Hierarchical Counterexamples for Discrete-Time Markov Chains*

Nils Jansen[1], Erika Ábrahám[1], Jens Katelaan[1], Ralf Wimmer[2], Joost-Pieter Katoen[1], and Bernd Becker[2]

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

Abstract. This paper introduces a novel *counterexample generation* approach for the verification of discrete-time Markov chains (DTMCs) with two main advantages: (1) We generate *abstract* counterexamples which can be refined in a *hierarchical* manner. (2) We aim at minimizing the number of states involved in the counterexamples, and compute a *critical subsystem* of the DTMC whose paths form a counterexample. Experiments show that with our approach we can reduce the size of counterexamples and the number of computation steps by several orders of magnitude.

1 Introduction

Discrete-time Markov chains (DTMCs) are a well-known modeling formalism for probabilistic systems. The *probabilistic computation tree logic (PCTL)* [6] is suited to express bounds on the probability mass of all paths satisfying some properties. Efficient algorithms and tools are available to *verify* PCTL properties of DTMCs. Prominent model checkers like Prism [9] and Mrmc [8] offer methods based on the solution of linear equation systems [6].

If verification reveals that a system does not fulfill a required property, the ability to provide diagnostic information is crucial for bug fixing. A *counterexample* carries an explanation why the property is violated. E. g., for Kripke structures and linear temporal logic (LTL) formulae, a counterexample is a path that violates the property, which can be generated by LTL model checking as a *by-product* without additional overhead. State-of-the-art model checking algorithms for probabilistic systems do not exhibit this feature. After model checking, current techniques have to apply additional methods to *generate* probabilistic counterexamples.

Even for large state spaces, a counterexample consisting of a single path gives an intuitive explanation why the property is violated. In the probabilistic setting, instead of a *single path* we need a *set of paths* whose total probability mass violates

* 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 the Research Training Group AlgoSyn (1298).

the bound specified by the PCTL formula [5]. It is much harder to understand the behavior represented by such a probabilistic counterexample as it may consist of a large or even infinite number of paths. To ease understanding, most approaches aim at finding counterexamples with a small number of paths having high probabilities. To generate more compact counterexamples, also the usage of regular expressions [5], the detection of loops [11], and the abstraction of strongly connected components (SCCs) [4] have been proposed, as well as diagnostic subgraphs [3], which is most related to our counterexample representation.

We suggested in [2] a model checking approach based on the hierarchical abstraction of SCCs. We abstract each SCC by a small loop-free graph in a recursive manner by the abstraction of sub-SCCs. The result is an abstract DTMC consisting of a single initial state and absorbing states, and transitions carrying the total probabilities of reaching target states. In [2] we also gave an idea of how to use the SCC-based model checking result for counterexample generation. In this paper we generalize this approach and suggest a novel method which computes a *critical subsystem* whose paths induce a counterexample. While other methods concentrate on minimizing the *number of paths*, our computation regards the system structure and aims at reducing the *number of involved states and transitions.*

Critical subsystems are computed *hierarchically*. We refine a critical subsystem by concretizing abstract states and reducing the concretized parts, such that the reduced subsystem still induces a counterexample. This hierarchical approach increases the usability of counterexamples for large state spaces. Concretization of only the user-relevant parts of the abstract critical subsystem allows for an intuitive approach for error correction.

The computation of critical subsystems is based on finding most probable paths or path fragments to be contained in the critical subsystem. We propose two approaches. The *global* method searches for paths through the entire system. Our main contribution is the *local* search which aims at connecting most probable *path fragments*. In contrast to most of the other approaches, our method is *complete*, i. e., termination is always guaranteed.

Experiments for two well-known case studies show that our approach reduces the size of counterexamples and the number of computation steps by several orders of magnitude.

The paper is structured as follows: Section 2 contains some preliminaries. We recall our model checking algorithm in Section 3. Section 4 describes our counterexample generation method, for which we give some experimental results in Section 5. A more detailed version of this paper, including examples and illustrations, can be found in [1].

2 Preliminaries

Definition 1. *Assume a set AP of atomic propositions. A* discrete-time Markov chain (DTMC) *is a tuple $M = (S, I, P, L)$ with a non-empty finite state set S, an initial discrete probability distribution $I : S \to [0, 1]$ with $\sum_{s \in S} I(s) = 1$, a* *transition probability matrix* $P : S \times S \to [0,1]$ *with* $\sum_{s' \in S} P(s,s') = 1$ *for all* $s \in S$*, and a labeling function* $L : S \to 2^{AP}$.

To reduce notation, we refer to the components of a DTMC M_l^u by S_l^u, I_l^u, P_l^u, and L_l^u. E.g., we use S' to denote the state set of the DTMC M'. Assume in the following a set AP of atomic propositions and a DTMC $M = (S, I, P, L)$.

We say that there is a *transition* 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 say that the transitions (s_i, s_{i+1}) are *contained* in the path π, written $(s_i, s_{i+1}) \in \pi$. 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, and $\mathit{Paths}^M_{\mathit{fin}}(s,t)$ of those starting in s and ending in t. A state t is called *reachable* from another state s iff $\mathit{Paths}^M_{\mathit{fin}}(s,t) \neq \emptyset$.

A state set $S' \subseteq S$ is called *absorbing in* M iff there is a state in S' from which no state outside S' is reachable in M. We call S' *bottom in* M if this holds for all states in S'. States $s \in S$ with $P(s,s) = 1$ are also called *absorbing states*.

We call M *loop-free*, if all of its loops are self-loops on absorbing states. A set $S' \subseteq S$ is *strongly connected in* M iff for all $s, t \in S'$ there is a path from s to t visiting states from S' only. A *strongly connected component (SCC)* of M is a maximal strongly connected subset of S.

The probability measure for finite paths $\pi \in \mathit{Paths}^M_{\mathit{fin}}$ is defined by $\mathit{Pr}^M_{\mathit{fin}}(\pi) = \prod_{(s_i, s_{i+1}) \in \pi} P(s_i, s_{i+1})$. For a set $R \subseteq \mathit{Paths}^M_{\mathit{fin}}$ of paths we have $\mathit{Pr}^M_{\mathit{fin}}(R) = \sum_{\pi \in R'} \mathit{Pr}^M_{\mathit{fin}}(\pi)$ with $R' = \{\pi \in R \mid \forall \pi' \in R.\ \pi' \text{ is no prefix of } \pi\}$.

The syntax of *probabilistic computation tree logic (PCTL)* [6] is given by[3]

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

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

For a property $\mathbb{P}_{\leq\lambda}(\varphi_1 \, U \, \varphi_2)$ refuted by M, a *counterexample* is a set $C \subseteq \mathit{Paths}^M_{\mathit{fin}}$, $\mathit{Pr}^M_{\mathit{fin}}(C) > \lambda$ of finite paths starting in an initial state and *satisfying* $\varphi_1 \, U \, \varphi_2$. For $\mathbb{P}_{<\lambda}(\varphi_1 \, U \, \varphi_2)$, the probability mass has to be at least λ. We consider upper probability bounds; see [5] for the reduction of lower bounds to this case.

Model checking of PCTL properties can be reduced to checking properties of the form $\mathbb{P}_{\sim\lambda}(\Diamond\varphi)$. The φ-states are also called *target* states. We concentrate on this case and assume DTMCs to have single initial and target states. Note that each DTMC can be equivalently transformed to satisfy these requirements.

3 SCC-based Model Checking

Next we recall our model checking algorithm from [2]. Given a DTMC M, we are interested in the total probability of reaching its target state from its initial

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

state. Each non-bottom SCC S' of M induces a DTMC M_{ind}: those states of the SCC through which paths may enter it are the initial states of M_{ind}; we call them *input states*. Those states outside the SCC to which paths may exit, the so-called *output states*, are absorbing states in M_{ind}. The remaining graph of M_{ind} is defined by the SCC's structure. We use $Inp^M(S') = \{t \in S' \mid I(t) > 0 \vee \exists s \in S \backslash S'.\ P(s,t) > 0\}$ and $Out^M(S') = \{t \in S \backslash S' \mid \exists s \in S'.\ P(s,t) > 0\}$ for the set of input respectively output states, and call states from S' *inner states*. Let in the following $M = (S, I, P, L)$ be a DTMC and $S' \subseteq S$ a not absorbing state set in M.

Definition 2. *The* DTMC induced by S' in M, *written* $DTMC(S', M)$, *is* $M_{ind} = (S_{ind}, I_{ind}, P_{ind}, L_{ind})$ *with*

1. $S_{ind} = S' \cup Out^M(S')$,
2. $\forall s \in S_{ind}.\ \big(I_{ind}(s) > 0 \leftrightarrow s \in Inp^M(S')\big)$,
3. $P_{ind}(s,t) = \begin{cases} P(s,t) & \textit{for } s \in S' \textit{ and } t \in S_{ind}, \\ 1 & \textit{for } s = t \in Out^M(S'), \\ 0 & \textit{else.} \end{cases}$
4. $\forall s \in S_{ind}.\ L_{ind}(s) = L(s)$.

We use the notation $Inp(M_{ind}) = \{s \in S_{ind} \mid I_{ind}(s) > 0\}$ *and* $Out(M_{ind}) = \{s \in S_{ind} \mid P_{ind}(s,s) = 1\}$.

The model checking procedure replaces inside M the subgraph M_{ind} by a smaller subgraph M_{abs} with the input and output states as state set and transitions from each input state s to each output state t carrying the total probability mass $Pr^{M_{ind}}\big(Paths_{fin}^{M_{ind}}(s,t)\big)$.

Definition 3. *Let* $DTMC(S', M) = M_{ind} = (S_{ind}, I_{ind}, P_{ind}, L_{ind})$ *and*

$$p_{s,t} = Pr_{fin}^{M_{ind}}\big(\{ss_1 \dots s_n t \in Paths_{fin}^{M_{ind}} \mid \forall 1 \le i \le n.\ s_i \neq s \wedge s_i \neq t\}\big)$$

for all $s \in Inp(M_{ind})$ *and* $t \in Out(M_{ind})$. *We define the* abstraction of M_{ind}, *written* $Abs(M_{ind})$, *to be the DTMC* $M_{abs} = (S_{abs}, I_{abs}, P_{abs}, L_{abs})$ *with*

1. $S_{abs} = Inp(M_{ind}) \cup Out(M_{ind})$,
2. $I_{abs}(s) = I_{ind}(s)$ *for all* $s \in S_{abs}$,
3. $P_{abs}(s,t) = \begin{cases} p_{s,t} / \left(\sum_{t' \in Out(M_{ind})} p_{s,t'}\right) & \textit{for } s \in Inp(M_{ind}),\ t \in Out(M_{ind}), \\ 1 & \textit{for } s = t \in Out(M_{ind}), \\ 0 & \textit{else.} \end{cases}$
4. $L_{abs}(s) = L_{ind}(s)$ *for all* $s \in S_{abs}$.

Next we formalize the abstraction and the concretization of an SCC.

Definition 4. *Let* $DTMC(S', M) = M_1 = (S_1, I_1, P_1, L_1)$, *and* $M_2 = (S_2, I_2, P_2, L_2)$ *a DTMC satisfying* $S_2 \cap (S \backslash S_1) = \emptyset$ *such that either* $M_2 = Abs(M_1)$ *or* $M_1 = Abs(M_2)$. *Then the* result of the substitution of M_1 by M_2 in M, *written* $M[M_2/M_1]$, *is the DTMC* $M_{sub} = (S_{sub}, I_{sub}, P_{sub}, L_{sub})$ *with*

Algorithm 1

Model_check(DTMC $M = (S, I, P, L)$, PCTL-formula $\mathbb{P}_{\sim\lambda}(\lozenge p)$)
begin
  $(M, \mathit{Sub}) :=$ Abstract_SCC$(M, \emptyset)$;
  $\mathit{result} := \Big(\sum_{s \in \mathit{Inp}(M)} \sum_{t \in \mathit{Out}(M)} (I(s) \cdot P(s,t)) \sim \lambda\Big)$;
  return $(\mathit{result}, M, \mathit{Sub})$
end

Abstract_SCC(DTMC $M = (S, I, P, L)$, Abstractions Sub)
begin
  for all non-bottom SCCs K in $\mathit{DTMC}(S \backslash \mathit{Inp}(M), M)$ **do**
    $M_K := \mathit{DTMC}(K, M)$; $(M_K^{abs}, \mathit{Sub}) :=$ Abstract_SCC(M_K, Sub);
    $M := M[M_K^{abs}/M_K]$
  end for
  $M^{abs} := \mathit{Abs}(M)$; $\mathit{Sub} := \mathit{Sub} \cup \big\{(M, M^{abs})\big\}$;
  return (M^{abs}, Sub)
end

1. $S_{sub} = (S \backslash S_1) \cup S_2$,
2. $I_{sub}(s) = I(s)$ *for* $s \in S_{sub}$ *and* 0 *otherwise,*
3. $P_{sub}(s,t) = P_2(s,t)$ *for* $s \in \big(S_2 \backslash \mathit{Out}(M_2)\big)$ *and* $t \in S_2$*, and* $P(s,t)$ *otherwise,*
4. $L_{sub}(s) = L_2(s)$ *for* $s \in S_2$ *and* $L(s)$ *otherwise.*

The replacement of an SCC by its abstraction and vice versa does not affect the total probabilities of reaching a target state from an initial state in M [1].

To compute the abstraction M_{abs} of an induced DTMC M_{ind}, we determine the probabilities $p_{s,t}$ recursively as follows. We detect all non-bottom SCCs in M_{ind} that do not contain any input states of M_{ind}, and replace them by their abstractions recursively. The result is a DTMC M'_{ind} which is loop-free in case M_{ind} has a single input state (multiple input states need a special treatment, see [2]), such that the probabilities $p_{s,t}$ can be computed easily.

The model checking algorithm is shown in Algorithm 1. We use a global variable *Sub* to store the pairs of abstracted DTMCs and their abstractions for the concretization during counterexample generation.[4]

4 Counterexample Generation

Our computation is based on the detection of single *paths*, which we use to determine a subgraph (*closure*) of the original system. We call the closure a *critical subsystem* if its paths form a counterexample for the violated property.

The closure is computed according to a *selection* $m \subseteq S \times S$. We use $\mathit{extend}^M : (2^{S \times S} \times \mathit{Paths}^M_{\mathit{fin}}) \to 2^{S \times S}$ defined by $\mathit{extend}(m, \pi) = \big\{(s, s') \in S \times S \;\big|\; (s, s') \in m \vee (s, s') \in \pi\big\}$ to extend a selection m with the transitions of a path π.

[4] Instead of copying, the implementation uses different markings to specify sub-graphs.

Algorithm 2

SearchAbstractCex(DTMC M, PCTL-formula $\mathbb{P}_{\sim\lambda}(\Diamond\, p)$)
begin
(10) $(\text{result}, M_{ce}, Sub) := \text{ModelCheck}(M,\, \mathbb{P}_{\sim\lambda}(\Diamond\, p))$;
(11) **if** result = true **then return** $\perp$
(12) **else**
(13) $m_{max} := \{(s_0, t)\}$;
(14) **while** true **do**
(15) $m_{min} := m_{max}$;
(16) $(\text{ready}, M_{ce}, m_{min}, m_{max}) := \text{Concretize}(M_{ce}, m_{min}, m_{max}, Sub)$;
(17) **if** (ready = true) **then return** $\mathit{closure}^{M_{ce}}(m_{max})$
(18) **else** $m_{max} := \text{CriticalSubsystem}\big(M_{ce}, m_{min}, m_{max}, \mathbb{P}_{\sim\lambda}(\Diamond\, p)\big)$;
(19) **end if**
(20) **end while**
(21) **end if**
end

Definition 5 (Closure). *For a DTMC $M = (S, I, P, L)$, target state t, and a selection $m \subseteq S \times S$, the* closure $\mathit{closure}^M(m) = (S_{cl}, I_{cl}, P_{cl}, L_{cl})$ of m in M *is given by $S_{cl} = S \uplus \{s_\perp\}$, $I_{cl}(s) = I(s)$, $L_{cl}(s) = L(s)$ for $s \in S$ and $I_{cl}(s_\perp) = 0, L_{cl}(s_\perp) = \emptyset$ and*

$$P_{cl}(s, s') = \begin{cases} P(s, s') & \text{for } (s, s') \in m, \\ 1 - \sum_{(s,s'') \in m} P(s, s'') & \text{for } s \in S \backslash \{t\} \text{ and } s' = s_\perp, \\ 1 & \text{for } s = s' = t \text{ or } s = s' = s_\perp, \\ 0 & \text{otherwise.} \end{cases}$$

Given a PCTL property φ, we call a DTMC M' a critical subsystem of M for φ *if $M' = \mathit{closure}^M(m)$ for some selection m and M' violates φ.*

4.1 The Basic Hierarchical Algorithm

We compute counterexamples in a hierarchical manner (see Algorithm 2): Intuitively, at first we compute a critical subsystem for the resulting abstract DTMC of the model checking procedure. Then we refine the DTMC stepwise hand in hand with its critical subsystem. For each refinement step, the abstract and the refined critical subsystems differ only in states and transitions affected by the refinement step.

The initial critical subsystem is given by the closure $\mathit{closure}^{M_{ce}}(m_{max})$ where the selection m_{max} contains the only transition from the initial state s_0 to the target state t of M_{ce} (line 13). Note that this initial subsystem represents *all* paths of M from its initial to its target state.

The *Concretize* method (Algorithm 3) concretizes some heuristically determined abstract states in M_{ce}. Thereby we remove all transitions from m_{max} that were removed by the concretization and add all transitions added by the

Algorithm 3

Concretize(DTMC M_{ce}, Selection m_{min}, Selection m_{max}, Abstractions *Sub*)
begin
(22) first = true;
(23) **while** true **do**
(24) $s_a :=$ ChooseAbstractState($closure^{M_{ce}}(m_{max})$);
(25) **if** ($s_a = \bot$) **then return** (first, M_{ce}, m_{min}, m_{max})
(26) **else**
(27) first := false;
(28) Let $(M_{abs}, M_{con}) \in Sub$ s. t. $s_a \in Inp(M_{abs})$;
(29) $Tr_{abs} := \{(s, s') \in S_{abs} \times S_{abs} \mid s \notin Out(M_{abs}) \wedge P_{abs}(s, s') > 0\}$;
(30) $Tr_{con} := \{(s, s') \in S_{con} \times S_{con} \mid s \notin Out(M_{con}) \wedge P_{con}(s, s') > 0\}$;
(31) $m_{min} := m_{min} \setminus Tr_{abs}$; $m_{max} := (m_{max} \setminus Tr_{abs}) \cup Tr_{con}$;
(32) $M_{ce} := M_{ce}[M_{con}/M_{abs}]$;
(33) **end if**
(34) **end while**
end

Algorithm 4 Global Search

CriticalSubsystem(DTMC M_{ce}, Selection m_{min}, Selection m_{max}, Formula $\mathbb{P}_{\sim\lambda}(\Diamond p)$)
begin
(35) $k := 0$; $M_{max} := closure^{M_{ce}}(m_{max})$;
(36) Let s_0 be the initial and t the target state of M_{max};
(37) **repeat**
(38) $k := k + 1$; $\pi :=$ FindNextPath(s_0, t, M_{max}, k); $m_{min} := extend(m_{min}, \pi)$;
(39) **until** ModelCheck($closure^{M_{ce}}(m_{min})$, $\mathbb{P}_{\sim\lambda}(\Diamond p)$) reports violation;
(40) **return** m_{min};
end

concretization (line 31). If the closure of m_{max} in M_{ce} represents a counterexample, then also the closure of the updated selection m_{max} in the concretization of M_{ce} represents a counterexample with the same probability. However, this counterexample may be unnecessarily large. *CriticalSubsystem* searches for a smaller selection included in m_{max} that still contains all transitions that were not affected by the concretization.

4.2 Search Algorithms

Global search. An implementation for *CriticalSubsystem*, which we call the *global search* algorithm, is proposed in Algorithm 4. Similarly to [5], we search for most probable paths from the initial state to the target state in the subsystem $M_{max} = closure^{M_{ce}}(m_{max})$ (line 35). After a next most probable path has been found (line 38), the algorithm extends m_{min} with the found path (line 38). This procedure is repeated until the closure of m_{min} is large enough to represent a counterexample (line 39).

Algorithm 5 Local Search

CriticalSubsystem(DTMC M_{ce}, Selection m_{min}, Selection m_{max},
PCTL-formula $\mathbb{P}_{\sim\lambda}\left(\Diamond\, p\right)$)
begin
$M_{cl} := \mathit{closure}^{M_{ce}}(m_{min})$;
while ModelCheck($M_{cl}, \mathbb{P}_{\sim\lambda}\left(\Diamond\, p\right)$) reports satisfaction **do**
$M_{search} := \mathit{closure}^{M_{ce}}(m_{max} \backslash m_{min})$;
$\Pi := \left\{ \pi' \in \mathit{Paths}_{\mathit{fin}}^{M_{search}}(s,t) \;\middle|\; s \in \mathit{Inp}(M_{search}) \wedge t \in \mathit{Out}(M_{search}) \right\}$;
$\pi := \arg\max_{\pi \in \Pi} \mathit{Pr}_{\mathit{fin}}(\pi)$;
$m_{min} := \mathit{extend}(m_{min}, \pi)$; $M_{cl} := \mathit{closure}^{M_{ce}}(m_{min})$;
end while
return m_{min}
end

Local search. The global search is complete, but it may find most probable paths which do not extend the minimal selection m_{min}. This can be time-consuming, e. g., when many different traversals of loops are considered.

Our second implementation for *CriticalSubsystem* (Algorithm 5), which we call the *local search*, overcomes this problem and finds only paths that extend the minimal selection and increase the target reachability probability of its closure. Instead of searching for paths from the initial to the target state, it aims at finding most probable *path fragments* that connect fragments of already found paths to new paths. The path fragments should, as the paths for the global search, lie in the closure of m_{max}. But this time they should (1) start at states reachable from an initial state via transitions of m_{min}, (2) end in states from which the target state is reachable via transitions from m_{min}, and (3) contain transitions from $m_{max} \backslash m_{min}$ only. I. e., we only search for path fragments in the subgraphs inserted by the last concretization step, which connect path fragments in the closure of m_{min} to whole paths from the initial to the target state.

5 Experimental results

We developed a C++ implementation with exact arithmetic for both search algorithms, and used it to run experiments on a 2.4 GHz dual core CPU with 4 GB RAM. We used Prism [9] to generate models for different instances of the parametrized *synchronous leader election protocol* [7] and the *crowds protocol* [10].

The global and the local search work on hierarchical data types. However, they can also directly be applied to concrete models. We consider this non-hierarchical approach to obtain a fair comparison to [5]. Table 1 compares the global method with the k-shortest path search for the leader election protocol, where the probability of reaching a target state is always 1. Table 2 depicts results for the crowds benchmark additionally containing the local search. The global search finds paths in the same order as k-sp, but due to the closure computation *earlier termination*, a significantly *smaller number of needed paths*, and therefore a

Table 1. Results for the leader benchmark on concrete models (TO > $1h$)

states		3902			12302		
transitions		5197			16397		
prob. threshold		0.92	0.93	0.95	0.95	0.96	0.97
k-sp	# paths	1193	8043	41636	3892	53728	-TO-
	# states	3593	3903	3903	11690	12302	12302
global	# paths	1193	1301	1850	3892	4360	5870
	# states	3593	3634	3676	11690	11815	11941
	prob.	0.9205	0.9302	0.9501	0.9502	0.9600	0.9700

Table 2. Results for the crowds benchmark on concrete models (TO > $1h$)

states		396		3515					18817	
transitions		576		6035					32677	
total prob.		0.1891		0.2346					0.4270	
prob. threshold		0.12	0.15	0.1	0.12	0.15	0.21	0.23	0.2	0.25
k-sp	# paths	1301	26184	3974	26981	488644	-TO-	-TO-	-TO-	-TO-
	# states	133	133	671	831	1071	-TO-	-TO-	-TO-	-TO-
global	# paths	38	76	91	220	935	3478	151639	3007	56657
	# closures	24	29	58	73	181	364	623	302	767
	# states	89	93	143	169	631	671	1071	663	2047
	prob.	0.1339	0.1514	0.1014	0.1203	0.1501	0.2101	0.2300	0.2002	0.2500
local	# paths	26	32	60	68	98	326	665	202	798
	# states	55	67	99	104	171	670	900	326	1439
	prob.	0.1238	0.1509	0.1018	0.1211	0.1525	0.2101	0.2300	0.2001	0.2508

smaller number of *computation steps* are achieved. For probability thresholds near the total probability, the number of paths for k-sp is several orders of magnitude larger. The number of considered states can also be reduced significantly. The local search not only leads to smaller critical subsystems in most cases, but also needs a much smaller number of found path fragments in comparison to the global search. The probability mass for all types of counterexamples is always very close to the specified probability threshold. Note that for our methods we model check only extended subsystems, while for the local search actually every new path extends the system.

The search for hierarchical counterexamples is motivated by their usefulness and understandability. The results in Table 3 show that the hierarchical search leads to critical subsystems of comparable size (the third last column is the hierarchical version of the global search in the second last column of Table 2). The number of found paths is much larger in the hierarchical approach, because we have to search at each abstraction level. However, due to abstraction, the found paths are shorter, especially for the local search, and the concretization up to the concrete level seems not necessary for many cases. We did experiments using different heuristics for the number of abstract states that are concretized in one step (e. g., either a single one or $\sqrt{n}$ with n the number of abstract states). We

Table 3. Results for a crowds instance (18817 states, 32677 transitions, 0.2 probability threshold) on the hierarchical model

search type	global				local	
# abstract states to concretize in one step	$\surd$		single		$\surd$	single
heuristic to choose the next abstract state	prob	none	prob	none	prob	prob
# paths	13525	912455	38379	594881	496	545
# closures	728	730	728	729	496	545
# states	457	457	458	457	319	347
# refinements	13	10	37	37	9	28

also tried two different heuristics for the choice of the next abstract state, either being just the next one found ("none"), or the one whose outgoing transitions have the maximal average probability ("prob").

References

1. Ábrahám, E., Jansen, N., Katelaan, J., Wimmer, R., Katoen, J.P., Becker, B.: Hierarchical counterexamples for discrete-time Markov chains. Tech. rep., RWTH Aachen University (2011), http://sunsite.informatik.rwth-aachen.de/Publications/AIB/2011/2011-11.pdf
2. Ábrahám, E., Jansen, N., Wimmer, R., Katoen, J.P., Becker, B.: DTMC model checking by SCC reduction. In: Proc. of QEST. pp. 37–46. IEEE CS (2010)
3. 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)
4. Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Proc. of HVC. LNCS, vol. 5394, pp. 129–148. Springer (2008)
5. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering 35(2), 241–257 (2009)
6. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
7. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation 88(1), 60–87 (1990)
8. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. In: Proc. of QEST. pp. 167–176. IEEE CS (2009)
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. of CAV. LNCS, Springer (2011), (to appear)
10. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM Trans. on Information and System Security 1(1), 66–92 (1998)
11. Wimmer, R., Braitling, B., Becker, B.: Counterexample generation for discrete-time Markov chains using bounded model checking. In: Proc. of VMCAI. LNCS, vol. 5403, pp. 366–380. Springer (2009)