RWTH Aachen

Department of Computer Science
Technical Report

Hierarchical Counterexamples for Discrete-Time Markov Chains

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

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2011-11

RWTH Aachen · Department of Computer Science · May 2011

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
[2] Albert-Ludwigs-University Freiburg, Germany

Abstract. In this paper we introduce a novel *counterexample generation* approach for 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 orders of magnitude.

1 Introduction

A wide range of safety-critical systems exhibit probabilistic behavior. *Discrete-time Markov chains (DTMCs)* are a well-known modeling formalism for probabilistic systems. To describe properties of DTMCs we consider the unbounded fragment of *probabilistic computation tree logic (PCTL)* [5], an adaptation of CTL to probabilistic systems, 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 [5].

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 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 the bound specified by the PCTL formula [4]. 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 [4], the detection of loops [12], and the abstraction of strongly connected components (SCCs) [3] have been proposed, as well as diagnostic subgraphs [2], which is most related to our counterexample representation.

We suggested in [1] 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 [1] we also gave an idea of how to use the SCC-based model checking result for counterexample generation. In this paper we first generalize the formalisms underlying our model checking algorithm. Then we use these formalisms to suggest a novel counterexample generation method, which computes a *critical subsystem* whose paths induce a counterexample. Compared to other approaches, the induced counterexamples are essentially different: Whereas other methods concentrate on minimizing the *number of paths*, our computation is structurally oriented 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 those parts of the abstract critical subsystem that are of interest for the user allows more intuition for error correction.

The computation is based on finding most probable paths or path fragments to be contained in the critical subsystem. We propose two different methods for the search. The *global* method searches, similarly to other approaches, for paths through the whole system. One of our main contributions is the *local* search which aims at connecting most probable *path fragments*. In contrast to most of the other approaches, our method is *complete*, and it terminates even if an infinite number of paths is needed for a counterexample. In the local search we strictly avoid to find paths that only differ in the number of unrollings of loops.

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

The remaining part of the paper is structured as follows: Section 2 contains some standard definitions and notations. We provide the theoretical background for counterexample generation and 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. We conclude the paper in Section 6.

2 Preliminaries

In this section we introduce discrete-time Markov chains and probabilistic computation tree logic, and define counterexamples in this probabilistic setting.

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, in the rest of the paper we sometimes refer to the components of a DTMC M_l^u by $(S_l^u, I_l^u, P_l^u, L_l^u)$ without explicitly defining them. For example, we use S', S_1, ... to denote the state sets of the DTMCs M', M_1, ...

Assume in the following a set AP of atomic propositions and a DTMC $M = (S, I, P, L)$. We also call a state $s \in S$ with $p \in L(s)$ a p-state.

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

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.

A finite path $\pi \in Paths_{fin}^M$ has an associated *cylinder set* $Cyl(\pi) = \{\pi' \in Paths_{inf}^M \mid \pi \text{ is a prefix of } \pi'\}$. The unique *probability measure* Pr^M of a DTMC M is defined on the associated smallest σ-algebra which contains the cylinder sets of finite paths. We set $Pr^M\big(Cyl(s_0 \ldots s_n)\big) = \prod_{i=0}^{n-1} P(s_i, s_{i+1})$, and for $\pi \in Paths_{fin}^M$ let $Pr_{fin}^M(\pi) = Pr^M\big(Cyl(\pi)\big)$. For a set $R \subseteq Paths_{fin}^M$ we define $Pr_{fin}^M(R) = \sum_{\pi \in R'} Pr_{fin}^M(\pi)$ with $R' = \{\pi \in R \mid \forall \pi' \in R.\ \pi' \text{ is not a prefix of } \pi\}$. Note that $Pr_{fin}^M(\pi_1 s \pi_2) = Pr_{fin}^M(\pi_1 s) \cdot Pr_{fin}^M(s \pi_2)$ for $\pi_1 s \pi_2 \in Paths_{fin}^M$. Similarly for sets R_1 and R_2 of finite paths, $R_i' = \{\pi \in R_i \mid \forall \pi' \in R_i.\ \pi' \text{ is not a prefix of } \pi\}$, $i = 1, 2$, if all paths in R_1' end in the same state s and all paths in R_2' start in s, then $Pr_{fin}^M\big(\{\pi_1 s \pi_2 \mid \pi_1 s \in R_1' \wedge s \pi_2 \in R_2'\}\big) = Pr_{fin}^M(R_1) \cdot Pr_{fin}^M(R_2)$.

The *probabilistic computation tree logic (PCTL)* [5] is an adaptation of CTL to probabilistic systems with the abstract syntax[3]

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

for (state) formulae with $p \in AP$ an atomic proposition, $\lambda \in [0, 1] \subseteq \mathbb{R}$ a probability threshold, and $\sim \ \in \{<, \leq, \geq, >\}$ a comparison operator. The probability operator $\mathbb{P}$ allows to express probability thresholds on the probability mass of all paths starting in a certain state and satisfying a (path) formula. The U is the classical "until" operator. As usual, we define additional operators like $\Diamond\varphi := \text{true } U \ \varphi$ and $\mathbb{P}_{\leq\lambda}(\Box\varphi) = \mathbb{P}_{\geq 1-\lambda}(\Diamond\neg\varphi)$ as syntactic sugar.

In case a property $\mathbb{P}_{\leq\lambda}(\varphi_1 \, U \, \varphi_2)$ is refuted by a DTMC M, a *counterexample* is a set $C \subseteq Paths_{fin}^M$ of finite paths starting in initial states such that all infinite paths from their associated cylinder sets *satisfy* $\varphi_1 \, U \, \varphi_2$ and $Pr_{fin}^M(C) > \lambda$. For $\mathbb{P}_{<\lambda}(\varphi_1 \, U \, \varphi_2)$ the probability mass has to be at least λ. In this paper we consider only formulae with upper probability bounds; a method in [4] can be used to reduce properties with lower bounds to the former case.

In order to check a property $\mathbb{P}_{\sim\lambda}(\varphi_1 \ U \ \varphi_2)$, usually (1) a labeling for the subformulae φ_1 and φ_2 is generated, possibly by recursively invoking probabilistic

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

model checking for subformulae, (2) the DTMC is reduced by making all states satisfying $\varphi_2 \vee (\neg\varphi_1 \wedge \neg\varphi_2)$ absorbing, and (3) the probability of reaching a φ_2-state from an initial state in the reduced DTMC is computed. The φ_2-states are also called *target* states. In this paper we concentrate on the last point (3), for which we need to consider reduced DTMCs and the temporal operator $\lozenge$ only.

It is possible to transform a reduced DTMC with multiple initial or target states to a DTMC with a single initial and a single target state, without changing the probability of reaching a target state. Adding an auxiliary initial state allows us to transform a DTMC into another one that has no loops containing the initial state.

The above observations allow us to consider in the following w. l. o. g. only (1) formulae of the form $\mathbb{P}_{\sim\lambda}(\lozenge p)$ with $\sim\, \in \{\leq, <\}$ and (2) DTMCs with a single initial and a single absorbing target state having no loops on the initial state.

3 SCC-based Model Checking

Next we describe our model checking algorithm for DTMCs and PCTL properties presented in [1]. Although the algorithm is basically the same as in [1], we need to define a more general formalization in order to handle counterexamples in the next section. The proof of correctness is given in [1].

Given a DTMC M, we are interested in the total probability of reaching its target state from its initial state. If M has no non-bottom SCCs, the probability is easy to compute. Otherwise, each non-bottom SCC S' of M induces a DTMC M_{ind} as follows. 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 the notation $Inp^M(S') = \{t \in S' \mid I(t) > 0 \vee \exists s \in S \backslash S'.\ P(s,t) > 0\}$ 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*.

Definition 2. *Let $M = (S, I, P, L)$ be a DTMC and $S' \subseteq S$ not absorbing in M. Then 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. $P_{ind}(s,t) = \begin{cases} P(s,t) & \text{for } s \in S' \text{ and } t \in S_{ind}, \\ 1 & \text{for } s = t \in Out^M(S'), \\ 0 & \text{else.} \end{cases}$
4. $\forall s \in S_{ind}.\ L_{ind}(s) = L(s)$.

In the above definition we do not require that S' is an SCC, only that it is not absorbing; this way we can use this definition also for the concretization later. Note that the initial distribution of the induced DTMC is not uniquely specified; in fact, we only need to specify the input states as initial states and can choose any distribution satisfying this requirement. Note furthermore that the output states are the only absorbing states of the induced system, since S' is required to be not absorbing. Thus the initial states of M_{ind} are the input

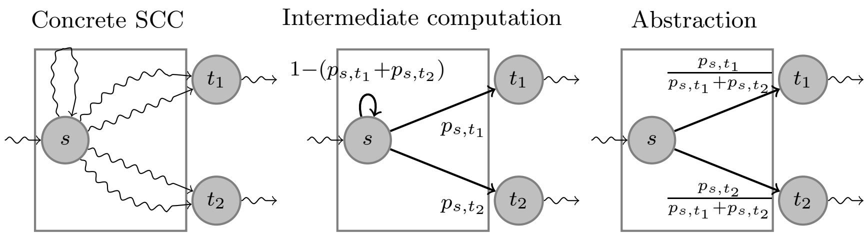


Fig. 1. Abstraction of an SCC

states of S' in M, and the absorbing states of M_{ind} are the output states of S' in M. We also use the notation $Inp(M_{ind}) = \{s \in S_{ind} \mid I_{ind}(s) > 0\}$ and $Out(M_{ind}) = \{s \in S_{ind} \mid P_{ind}(s,s) = 1\}$. Note that I_{ind} is defined to be an arbitrary initial distribution that specifies the input states as initial states.

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 $Pr^{M_{ind}}\big(Paths_{fin}^{M_{ind}}(s,t)\big)$ (see Fig. 1). This total probability mass is determined in two steps: First, we compute for each input state s and output state t the total probability mass $p_{s,t}$ of all finite paths from s to t that have no loop containing s. Since S' is not absorbing, the probability to eventually reach an output state in M_{ind} is 1. Therefore, the probability of a self-loop on an input state s is $1 - \sum_{t' \in Out(M_{ind})} p_{s,t'}$. Thus the probability of the transition from s to t in M_{abs} is determined by $p_{s,t}/(\sum_{t' \in Out(M_{ind})} p_{s,t'})$.

Definition 3. *Let $M = (S, I, P, L)$ be a DTMC and $S' \subseteq S$ not absorbing. Assume furthermore $DTMC(S', M) = M_{ind} = (S_{ind}, I_{ind}, P_{ind}, L_{ind})$ and*

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

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. $P_{abs}(s,t) = \begin{cases} p_{s,t} \ / \left(\sum_{t' \in Out(M_{ind})} p_{s,t'}\right) & \text{for } s \in Inp(M_{ind}),\ t \in Out(M_{ind}), \\ 1 & \text{for } s = t \in Out(M_{ind}), \\ 0 & \text{else.} \end{cases}$
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 $M = (S, I, P, L)$ be a DTMC, $S' \subseteq S$ a not absorbing state set, $DTMC(S', M) = M_1 = (S_1, I_1, P_1, L_1)$, and $M_2 = (S_2, I_2, P_2, L_2)$ a DTMC satisfying $S_2 \cap (S \setminus S_1) = \emptyset$ 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*

1. $S_{sub} = (S \setminus S_1) \cup S_2$,
2. $I_{sub}(s) = I(s)$ *for* $s \in S_{sub}$ *and* 0 *otherwise,*

3. $P_{sub}(s,t) = P_2(s,t)$ *for* $s \in \big(S_2 \backslash Out(M_2)\big)$ *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:

Theorem 1. *Let* $M = (S, I, P, L)$ *be a DTMC,* $S' \subseteq S$ *a not absorbing state set,* $DTMC(S', M) = M_1 = (S_1, I_1, P_1, L_1)$, *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 for* $M' = M[M_2/M_1]$ *all initial states* s *resp. target states* t *of* M *are also initial resp. target states of* M', *and it holds that*

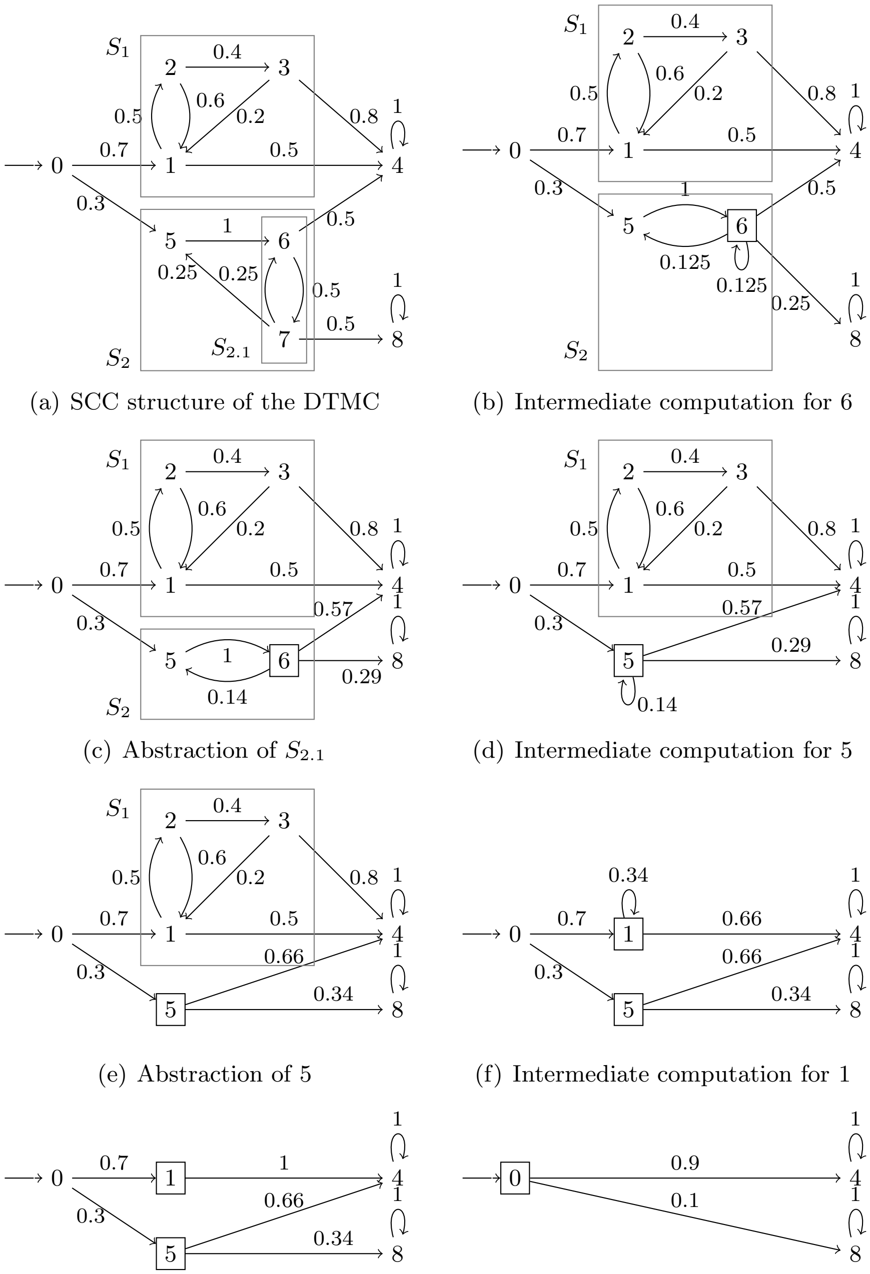
$$Pr^{M}_{fin}\big(Paths^{M}_{fin}(s,t)\big) = Pr^{M'}_{fin}\big(Paths^{M'}_{fin}(s,t)\big) \ .$$

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 [1]), such that the probabilities $p_{s,t}$ can easily be computed.

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]

Example 1. The example in Fig. 2 illustrates the model checking procedure. On the topmost level, SCCs named S_1 and S_2 are found in the input DTMC. By ignoring their input states we only detect the SCC $S_{2.1}$ inside S_2 (Fig. 2(a)). At this bottom level of the recursion, all cycles in $S_{2.1}$ go through its single input state 6. The total probabilities of reaching the output states 4, 5, resp. 8 from the input state 6 without looping back to 6, are 0.5, 0.125, resp. 0.25. Looping on 6 is thus possible with probability 0.125 (Fig. 2(b)). We replace $S_{2.1}$ by the state $\boxed{6}$ with transitions to 4, 5, resp. 8 labeled with the probabilities 0.57, 0.14, resp. 0.29 (Fig. 2(c)), and continue at the next higher level, where for the abstraction of S_2 the previous computation is used. As the probabilities of reaching the output states 4 and 8 are 0.57 and 0.29, the probability of looping on $\boxed{5}$ is 0.14 (Fig. 2(d)). The abstraction yields transitions to 4 and 8 with probabilities 0.66 and 0.34 (Fig. 2(e)). As SCC S_1 has only one output state, the replacement state $\boxed{1}$ has only one outgoing transition to 4 with probability 0.66 and a self-loop with probability 0.34 (Fig. 2(f)). The probability of eventually reach state 4 from state 1 is therefore 1 (Fig. 2(g)). The final abstraction of this loop-free system only consists of edges from input state $\boxed{0}$ to the absorbing states. The transitions leading to states 4 and 8 with probabilities 0.9 and 0.1 depict the model checking result for unbounded reachability to these states (Fig. 2(h)).

[4] The implementation uses different markings to specify sub-graphs in order to store a single graph without copying subgraphs. The same holds for edge selections which will be introduced later.



(a) SCC structure of the DTMC

(b) Intermediate computation for 6

(c) Abstraction of $S_{2.1}$

(d) Intermediate computation for 5

(e) Abstraction of 5

(f) Intermediate computation for 1

(g) Abstraction of 1

(h) Path abstraction

Fig. 2. SCC-based model checking

Algorithm 1

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

4 Counterexample Generation

The result of the model checking procedure is an abstract and refinable DTMC. In this section we present the subsequent computation of a *hierarchical counterexample* in case the property was refuted.

4.1 Critical Subsystems

Similarly to other counterexample generation approaches, our computation is based on the detection of single *paths*. However, instead of just collecting found paths, we *select* all transitions appearing in the found paths and build a DTMC called *closure*, containing exactly the selected transitions and the involved states. We call the closure a *critical subsystem* if its paths form a counterexample for the violated property. In our approach we do not aim at minimizing the number of paths in the counterexample, but at minimizing the number of involved states and transitions and representing them intuitively as a subsystem.

A *selection* of $M = (S, I, P, L)$ is a relation $m \subseteq S \times S$. Selections can be extended with the transitions of a path using the function $\mathit{extend}^M : (2^{S\times S} \times \mathit{Paths}^M_{\mathit{fin}}) \to 2^{S\times S}$ defined by $\mathit{extend}(m, \pi) = \{(s, s')\in S\times S \mid (s, s')\in m \vee (s, s')\in\pi\}$.

Definition 5 (Closure). *For a DTMC $M = (S, I, P, L)$ and a selection $m \subseteq S \times S$, the* closure of m in M*, written $\mathit{closure}^M(m)$, is given by the DTMC $M_{cl} = (S_{cl}, I_{cl}, P_{cl}, L_{cl})$ with*

1. $S_{cl} = S \uplus \{s_\perp\}$,
2. $I_{cl} = I$,
3. $P_{cl}(s, s') = \begin{cases} P(s, s') & \text{for } (s, s') \in m, \\ 1 - \sum_{(s,s'')\in m} P(s, s'') & \text{for } s \in S\backslash\{t\} \text{ and } s' = s_\perp, \\ 1 & \text{for } s = s' = t \text{ or } s = s' = s_\perp, \\ 0 & \text{otherwise,} \end{cases}$
4. $L_{cl}(s) = L(s)$.

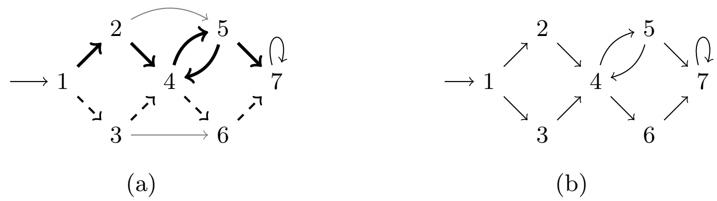


Fig. 3. (a) Selection for two intersecting paths and (b) closure for the selection

Algorithm 2

SearchAbstractCex(DTMC M, PCTL-formula $\mathbb{P}_{\sim\lambda}(\Diamond p)$)
begin
  (result, M_{ce}, *Sub*) := ModelCheck(M, $\mathbb{P}_{\sim\lambda}(\Diamond p)$); (11)
  if result = true **then** (12)
    return $\perp$ (13)
  else (14)
    $m_{max} := \{(s_0, t)\}$; (15)
    while true **do** (16)
      $m_{min} := m_{max}$; (17)
      (ready, M_{ce}, m_{min}, m_{max}) :=
        Concretize($M_{ce}, m_{min}, m_{max}, Sub$); (18)
      if (ready = true) **then** (19)
        return $closure^{M_{ce}}(m_{max})$ (20)
      else (21)
        m_{max} :=
          FindCriticalSubsystem$\big(M_{ce}, m_{min}, m_{max}, \mathbb{P}_{\sim\lambda}(\Diamond p)\big)$; (22)
      end if (23)
    end while (24)
  end if (25)
end

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

To give an intuition, if for the system in Fig. 3 the paths $1 \to 2 \to 4 \to 5 \to 4 \to 5 \to 7$ and $1 \to 3 \to 4 \to 6 \to 7$ are selected, e. g., all paths of the form $1 \to 3 \to (4 \to 5)^+ \to 7$ with arbitrarily many traversals of the loop $4 \to 5 \to 4$ will be contained in the counterexample. This property guarantees the termination of our approach even for counterexamples that need an infinite number of certain loop traversals. Missing transitions are implicitly leading to $s_\perp$.

4.2 The Basic Hierarchical Algorithm

We compute counterexamples in a hierarchical manner: First we compute a critical subsystem for the abstract DTMC that is the result 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 method *SearchAbstractCex*, depicted in Algorithm 2, calls the model checking procedure (line 11) yielding an abstract DTMC $M_{ce} = (S_{ce}, I_{ce}, P_{ce}, L_{ce})$ and a set of abstraction pairs *Sub*. If the property holds, the algorithm terminates

Algorithm 3

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

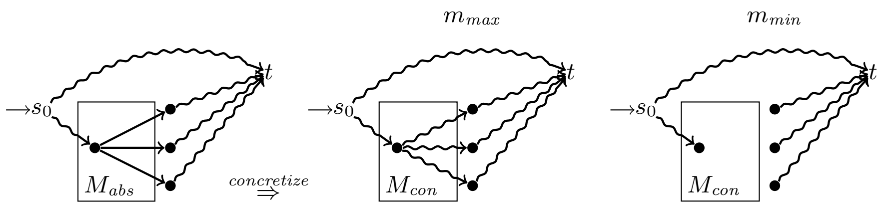


Fig. 4. Concretization of M_{abs} and resulting selections m_{min} and m_{max}

(lines 12–13). Otherwise it computes an initial critical subsystem for the abstract DTMC and refines it iteratively (lines 14–25).

The initial critical subsystem is given by the closure $closure^{M_{ce}}(m_{max})$ where the selection m_{max} contains the only transition from the initial state s_0 to the target state t of M_{ce} (line 15). Note that this initial subsystem represents *all* paths of M from its initial to its target state.

The concretization (line 18) is done by the *Concretize* method, listed in Algorithm 3, which determines heuristically a sequence of abstract states and concretizes them in M_{ce}. During this step, we remove all transitions from m_{max} that were removed by the concretization and add all transitions that were added by the concretization (line 37). It is easy to see that if the closure of m_{max} in M_{ce} represents a counterexample, then also the closure of the updated m_{max} in the concretization of M_{ce} represents a counterexample with the same probability. However, this counterexample is often unnecessarily large. For example, the concretized initial subsystem would still contain all paths from the initial to the target state. Therefore we search for a selection *included* in m_{max}. It should be smaller if possible, but, in order to generate hierarchical counterexamples, it should still *contain* all transitions that were not affected by the concretization step. To assure the latter requirement, we store a copy of m_{max} in m_{min} before the concretization, and during the concretization we remove concretized transitions from m_{min} (line 36). This way m_{min} puts a lower and m_{max} an upper bound on the selection inducing the concretized critical subsystem (see Fig. 4).

Algorithm 4 Global Search

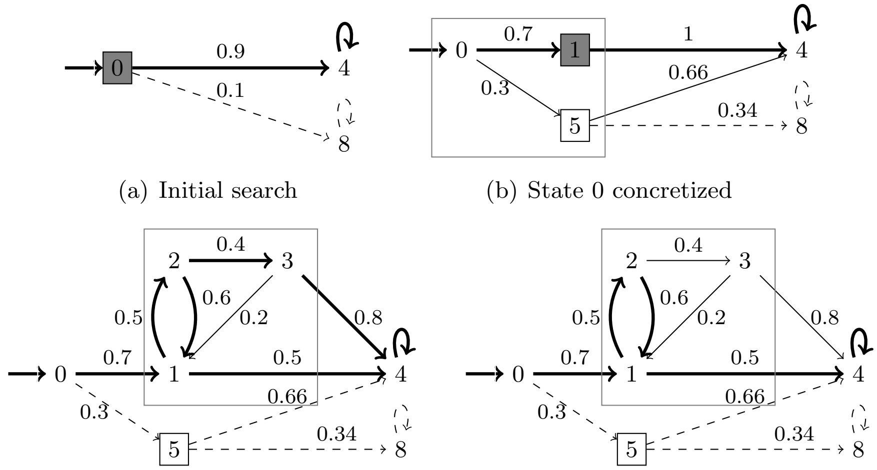
FindCriticalSubsystem(DTMC M_{ce}, Selection m_{min}, Selection m_{max},
PCTL-formula $\mathbb{P}_{\sim\lambda}(\Diamond p)$)
begin
(41) $M_{max} := closure^{M_{ce}}(m_{max})$;
(42) Let s_0 be the initial and t the target state of M_{max};
(43) $k := 0$;
(44) **repeat**
(45) $k := k + 1$;
(46) $\pi :=$ FindNextPath(s_0, t, M_{max}, k);
(47) $m_{min} := extend(m_{min}, \pi)$;
(48) **until** ModelCheck($closure^{M_{ce}}(m_{min})$, $\mathbb{P}_{\sim\lambda}(\Diamond p)$) reports violation;
(49) **return** m_{min};
end

If no concretization was required, the closure of m_{max} in M_{ce} is the final result (lines 19–20). Otherwise *FindCriticalSubsystem* (line 22) is invoked to determine a selection containing m_{min} and being contained in m_{max}, whose closure induces a counterexample after concretization.

4.3 Global Search

We propose an implementation for *FindCriticalSubsystem*, listed in Algorithm 4, which we call the *global search* algorithm. It searches for most probable paths from the initial state to the target state in the subsystem $M_{max} = closure^{M_{ce}}(m_{max})$ (line 41). For this search we follow Han et al. [4] and utilize a k-shortest paths algorithm [7] to accomplish an ordering on paths w. r. t. their probability. After a next most probable path has been found (line 46), the algorithm extends m_{min} with the found path (line 47). This procedure is repeated until the closure of m_{min} is large enough to represent a counterexample which is determined by our SCC-based model checking (line 48). Correctness of the whole method is ensured by applying model checking for the critical subsystem.

Example 2. Fig. 5 illustrates the global search for the example system of Fig. 2 violating the upper probability bound 0.5 of reaching state 4. Dashed lines denote transitions that are not in the closure of m_{max}, solid lines (both thick and thin) the closure of m_{max}, and thick lines the closure of m_{min}. The initial critical subsystem, depicted in Fig. 5(a), is the closure of $m_{max} = \{(0,4)\}$, and has a probability mass 0.9 to reach state 4. State 0 is grey, indicating that it will be concretized in the next step. After concretization, the global search determines $0 \to 1 \to 4$ as the most probable path, and adds $(0,1)$ to m_{min}, whose closure (Fig. 5(b)) has now a sufficient probability mass of 0.7. Note that the dashed transitions were ignored for the search. After concretizing state 1, the most probable paths $0 \to 1 \to 4$, $0 \to 1 \to 2 \to 3 \to 4$, and $0 \to 1 \to 2 \to 1 \to 4$ in the closure of m_{max} are sufficient to extend m_{min} to have a closure in which state 4 is reached with probability 0.66 (Fig. 5(c)).



(a) Initial search (b) State 0 concretized

(c) State 1 concretized, global search (d) State 1 concretized, local search

Fig. 5. Example applications of the global and the local search

4.4 Local Search

Though the global search is complete, it has one disadvantage: 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.

In this section we introduce a second implementation for *FindCriticalSubsystem* which we call the *local search* (see Algorithm 5), and which overcomes this problem. In contrast to the global search, the local search 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 search for path fragments only 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.

Example 3. We consider the computation of a counterexample for the same system as in Example 2, but this time using the local search. Applying local search to the result of the concretization of state 0 yields the same subsystem as applying global search (Fig. 5(b)). After concretizing state 1, we search for shortest path fragments that start at 1, end at 4, and visit states inside the concretized component (surrounded by a box) only. It is sufficient to extend m_{min} with the two shortest path fragments $1 \to 4$ and $1 \to 2 \to 1$: its closure after the extension is a critical subsystem with a sufficient probability mass of 0.583. Fig. 5(d) depicts the closures of the final minimal and maximal selections.

Algorithm 5 Local Search

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

5 Experimental results

We developed a C++ implementation with exact arithmetic for our local and global search algorithms. We used this tool to run some experiments on a 2.4 GHz Intel Core2 Duo CPU with 4 GB RAM. We used Prism [9] to generate models for different instances of the parametrized *synchronous leader election protocol* [6] and the *crowds protocol* [10].

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. The crowds protocol aims at anonymous communication in networks, where n users are divided in $g \cdot n$ good members and $(1-g)\cdot n$ 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. A *session* is the delivery of a message to a sender and the number of sessions is r. A user who was identified twice by a bad member, is *positively* identified, for this user no anonymity is guaranteed. In the corresponding DTMC model, such states are labeled with Pos. We verify the property $\mathbb{P}_{\leq p}(\lozenge Pos)$ while we fix $g = 0.833$ and $p_f = 0.8$. The models are parametrized by r and n. We also used different probability bounds p.

The global and the local search, introduced in the previous section, work on hierarchical data types. However, they can also directly be applied to concrete models. We consider this non-hierarchical approach because this allows a fair comparison to the k-shortest path approach of [4]. Furthermore, we can demonstrate on the one hand the advantage of incrementally computing the closure instead of building sets of paths, and on the other hand the improvements achieved by connecting most probable path fragments as done in our local search.

For this non-hierarchical application, Table 1 compares the global method with the k-shortest path search for the leader election protocol, where the prob-

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

ability 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 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 also 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 neccessary 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 also tried two different heuristics for the choice of the next abstract state, either

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

being just the next one found ("none"), or the one whose outgoing transitions have the maximal average probability ("prob").

6 Related Work and Conclusion

In this paper we introduced two approaches to generate counterexamples for DTMCs and unbounded PCTL properties. Most related to our work are [4], [3] and [2]. In [4] two methods are introduced. The first one applies a k shortest paths search on a DTMC to find the k most probable paths that form a counterexample. The second approach, based on state elimination, computes a regular expression, such that the set of paths whose state sequences are in the language of the regular expression is a counterexample. Although we also use a shortest paths algorithm, we generate structural counterexamples in form of critical subsystems. Furthermore, our local search method does not always consider shortest paths, but focuses on small subsystems.

The work [3] determines the SCCs in the graph of the DTMC. They use standard PCTL model checking for every non-bottom SCC and every input-output state pair of the SCC to compute the probabilities of reaching the output state from the input state, and replace the SCC by a minimal subgraph. An abstract counterexample is determined on the resulting acyclic DTMC. In contrast to our hierarchical approach, this abstraction technique allows only a one-level concretization.

In [2], a method for the generation of counterexamples for DTMCs and Continuous Time Markov Chains is proposed, that uses a directed explicit state search and represents counterexamples as a *diagnostic subgraphs*, which is similar to our representation as a critical subsystem. Bounded model checking in combination with loop detection was used in [12] to find most probable paths. Whereas we focus on PCTL, [11] deals with counterexamples for probabilistic LTL properties.

In this paper we introduced a global and a local method for the hierarchical computation of counterexamples for DTMCs and unbounded PCTL properties. Experimental results showed the advantage of the proposed methods. Currently we are working on the visualization of the critical subsystems and its refinement process and on the development of more sophisticated heuristics for the local search. In the future we will experimentally compare our method with other, both explicit and symbolic, methods. We also plan to develop a symbolic approach for the computation of critical subsystems.

References

1. 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)
2. 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)
3. 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)
4. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering 35(2), 241–257 (2009)
5. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
6. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation 88(1), 60–87 (1990)
7. Jiménez, V.M., Marzal, A.: Computing the k shortest paths: A new algorithm and an experimental comparison. In: Int'l Workshop on Algorithm Engineering (WAE). LNCS, vol. 1668, pp. 15–29. Springer (1999)
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. 23rd International Conference on Computer Aided Verification (CAV'11). 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. Schmalz, M., Varacca, D., Völzer, H.: Counterexamples in probabilistic LTL model checking for Markov chains. In: Proc. of CONCUR. LNCS, vol. 5710, pp. 587–602. Springer (2009)
12. 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)

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from `http://aib.informatik.rwth-aachen.de/`**. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email:** `biblio@informatik.rwth-aachen.de`

2008-01 * Fachgruppe Informatik: Jahresbericht 2007
2008-02 Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04 Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05 Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06 Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07 Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08 George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09 George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10 George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11 George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13 Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14 Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15 Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16 Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17 Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18 Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19 Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications

2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices

2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation

2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs

2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata

2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms

2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting

2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs

2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut

2010-11 Martin Zimmermann: Parametric LTL Games

2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut

2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems

2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination

2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles

2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten

2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit

2010-19 Felix Reidl: Experimental Evaluation of an Independent Set Algorithm

2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.