Regular Expressions for PCTL Counterexamples

Berteun Damman Tingting Han Joost-Pieter Katoen

Software Modeling and Verification, RWTH Aachen University, Germany
Formal Methods and Tools, University of Twente, The Netherlands

Abstract

Counterexamples for probabilistic reachability in Markov chains are sets of paths that all reach a goal state and whose cumulative likelihood exceeds a threshold. This paper is concerned with the issue of how to conveniently represent these sets. Experiments, partially substantiated with combinatorial arguments, show that the cardinality of such sets may be excessive. To obtain compact representations of counterexamples we suggest to use regular expressions. We present a simple algorithm to generate minimal regular expressions and adopt a recursive scheme to determine their likelihood. Markov chain reduction prior to counterexample generation may yield even shorter regular expressions. The feasibility of the approach is illustrated by means two protocols: leader election and the Crowds protocol.

1. Introduction

Model checking is a successful and widely applied technique for the automated verification of properties of system designs. Its popularity is witnessed by the rapidly increasing interest of industries to apply model checking to both hardware systems, as well as software products. Model checking is based on a systematic check of the validity of a system property, typically formulated in a temporal logic, in each state of a model of the system under consideration. The power of model checking is, however, not exhaustive verification, but its capability to generate useful diagnostic feedback in case a violation of the property is encountered. Due to this feature, model checking is seen as an effective and powerful bug-hunting technique: it does not only indicate that a property is refuted, but also indicates why. In fact, this also motivates recent developments such as bounded model checking that consider paths up to a certain maximum length only. Although these techniques just search a fragment of the state space and thus cannot guarantee the absence of flaws, their potential to find refuting behavior is considered of vital importance.

In the last decades, model-checking techniques have been adapted and extended to models that are widely used in performance and dependability analysis such as discrete-time and continuous-time Markov chains. Temporal logics have been defined that allow the specification of well-known performance and performability measures and efficient – both numerical and simulative techniques – algorithms have been developed and culminated into dedicated tools, such as PRISM [25] and MRMC [23], as well as extensions of existing tools such as GreatSPN, SPIN, PEPA Workbench, SMART, and STATEMATE. In contrast to traditional model-checking techniques, the support for diagnostic feedback in case a property is violated is rather limited; e.g., when the probability to reach a set of goal states (via legal paths) within 1,000 steps, say, exceeds the required threshold "at most 0.87", typically the feedback is just a list of states for which this is true, possibly accompanied with a curve showing the probability vs. the number of steps. It is left to the user to interpret these results, and to obtain more useful information about the cause of this refuting behavior.

One of the main reasons of this restricted form of feedback has been the absence of a clear notion of a counterexample in the probabilistic setting. Whereas it is clear that in case of a traditional safety property (e.g., always $x > 0$ for variable x), a single finite path that ends in a state where $x \leq 0$ suffices, this is no longer true for reachability probabilities. In fact, to show that the probability to reach a goal state exceeds 0.87, a *set* of paths is needed that all end in a goal state and whose total probability mass is larger than 0.87. In line with shortest counterexamples in classical model checking, preferably this set is as small as possible. Han and Katoen [15] have shown that the computation of the smallest set of paths that can act as a counterexample can be carried out using (small amendments of) k-shortest path algorithms [13][21], i.e., algorithms that compute k paths consisting of the shortest path, the one-but-shortest path, and so forth. These results have been recently generalized to CTMC counterexamples [16], and have been adopted to steer the refinement phase in a counterexample-guided abstraction refinement framework for MDPs [19], as well as

for counterexample generation for cpCTL, a variant of PCTL with means to reason about conditional probabilities [5]. An alternative approach proposed by Aljazzar and Leue [3] is to characterize counterexamples by rooted graphs, basically fragments of Markov chains where all paths from root to a leaf reach a goal state. Heuristic search algorithms are employed to generate counterexamples, see [2][3].

To be more precise, the shortest-path characterization in [15] yields a minimal counterexample for which there does not exist another equally-sized counterexample with higher probability mass. This paper reports on experiments to generate these so-called smallest counterexamples, and suggests to use regular expressions to represent them succinctly. Using the well-studied Itai and Rodeh's synchronous leader election protocol [20], we show that the size of a smallest counterexample, i.e., the minimal number of paths it contains, may be exponential in the input parameters of the protocol like the number of processes and the number of rounds. In order to obtain a better insight in this phenomenon, we provide a short mathematical analysis. The resulting closed-form expression confirms the double exponential growth of the size of counterexamples for this protocol.

In order to obtain a more comprehensible representation that may act as diagnostic feedback to the end-user, we propose to represent these counterexamples by regular expressions. The advantage of regular expressions is that they are commonly known, are easy to understand, and may yield very compact representations. In addition, they can be obtained by simple algorithms whose correctness is trivially checked. The idea is to represent a DTMC by a deterministic finite-state automaton (DFA, for short) and obtain regular expressions of sets of paths by applying state elimination. To obtain compact regular expressions, we adopt a recently proposed heuristic by Han and Wood [17] that determines the order in which states are eliminated. (Obtaining the optimal order is NP-hard.) This results in a simple algorithm for determining the minimal regular expression, i.e., an expression from which the elimination of any subexpression is not a counterexample. The probability of a regular expression is obtained using (a small amendment of) the approach advocated by Daws [10] for parametric model checking of DTMCs. This recursive evaluation is guaranteed to be exact (i.e., no rounding errors), provided the transition probabilities are rational. We provide the details of our approach and show its result when applied to Itai and Rodeh's leader election protocol, as well as the Crowds protocol [30], a protocol for anonymous web browsing that has been adapted, among others, to Bluetooth [32] and wireless Internet [4]. These examples show (as expected) that model reduction prior to counterexample generation may obtain even shorter regular expressions.

Summarizing, the main contributions of this paper are: 1) experiments with smallest counterexample generation; 2) advocating the use of regular expressions for counterexamples; 3) a simple algorithm to generate minimal counterexamples, and 4) the application of this approach to two protocols.

The paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents experimental results for the leader election protocol together with a brief combinatorial analysis. Section 4 details the use of regular expressions for counterexamples and is the core of the paper. Section 5 reports on counterexample generation for the Crowds protocol. Section 6 concludes the paper with future work.

2. Preliminaries

Definition 1 (DTMC) *A* discrete-time Markov chain (DTMC) $\mathcal{D}$ *is a triple* $(S, \mathbf{P}, \hat{s})$ *with* S *a finite set of states,* $\mathbf{P} : S \times S \to [0,1]$ *a stochastic matrix, and* $\hat{s}$ *the initial state.*

A state s is absorbing if $\mathbf{P}(s,s) = 1$, i.e., if s only has a self-loop. A finite path σ in $\mathcal{D}$ is a state sequence $s_0\, s_1 \ldots s_n$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$, for $0 \leqslant i < n$. The probability of σ, denoted $\mathbb{P}(\sigma) = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdots \mathbf{P}(s_{n-1}, s_n)$. We denote $\sigma[i]$ as the $(i+1)$-st state on σ; $|\sigma|$ the total number of transitions (or steps, or hops) of σ and $last(\sigma)$ to be the last state on σ. Let $Paths^{\mathcal{D}}(s)$ denote the set of all finite paths in $\mathcal{D}$ that start in state s. $Paths^{\mathcal{D}}$ is the set of all finite paths in $\mathcal{D}$. The superscript $\mathcal{D}$ will be omitted if is clear from the context.

Probabilistic (bounded) reachability We are mainly interested in *probabilistic (bounded) reachability* properties, i.e., given a goal state $\hat{t} \in S$, does the probability of reaching $\hat{t}$ meet the threshold p with or without a constraint on the maximal number h of steps (or hops) till reaching $\hat{t}$? In temporal logics, such as PCTL [18], this is formalized as $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant h}\, \hat{t})$, where $\bowtie \in \{<, \leqslant, >, \geqslant\}$, $0 \leqslant p \leqslant 1$, $h \in \mathbb{N} \cup \{\infty\}$ and $\Diamond^{\leqslant h}\, \hat{t}$ denotes the reachability of $\hat{t}$ within h hops.

We define a set of finite paths $Paths^{\leqslant h}(\hat{s}, \hat{t})$ to be $\{\sigma \in Paths(\hat{s}) \mid |\sigma| \leqslant h \wedge last(\sigma) = \hat{t} \wedge \forall i < |\sigma|.\ \sigma[i] \neq \hat{t}\}$. In words, for each path $\sigma \in Paths^{\leqslant h}(\hat{s}, \hat{t})$, $\hat{t}$ is the last state and the only goal state on σ. A state $\hat{s}$ satisfies $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant h}\, \hat{t})$, denoted $\hat{s} \models \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant h}\, \hat{t})$, iff $\mathbb{P}\big(Paths^{\leqslant h}(\hat{s}, \hat{t})\big) \bowtie p$. Note that $\mathbb{P}\big(Paths^{\leqslant h}(\hat{s}, \hat{t})\big) = \sum_{\sigma \in Paths^{\leqslant h}(\hat{s}, \hat{t})} \mathbb{P}(\sigma)$, which is due to the fact that paths are disjoint in $Paths^{\leqslant h}(\hat{s}, \hat{t})$, thus $\mathbb{P}$ is well-defined. In the rest of the paper, we will focus on formulas of the form $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\, \hat{t})$. We will discuss the $\mathcal{P}_{<p}(\Diamond^{\leqslant h}\, \hat{t})$ case later in Section 4. For the other PCTL formulas, refer to [15] for the reduction to the form $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\, \hat{t})$.

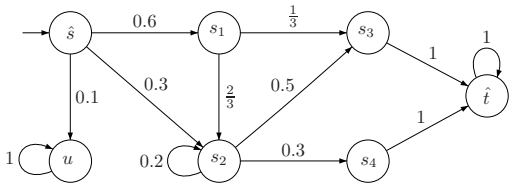


Figure 1. An example DTMC

Counterexample generation. For the violation of the formula $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$ in the initial state $\hat{s}$, we have:

$$\hat{s} \not\models \mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\hat{t}) \quad \text{iff} \quad \mathbb{P}\big(Paths^{\leqslant h}(\hat{s},\hat{t})\big) > p.$$

So, $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$ is refuted by state $\hat{s}$ if the probability of all $\hat{s}$-$\hat{t}$ paths of at most h hops exceeds p. This indicates that a counterexample for $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$ is a subset of $Paths^{\leqslant h}(\hat{s},\hat{t})$. Recall that σ ends at the first $\hat{t}$ state it meets.

Definition 2 (Evidence) *An* evidence *for the violation $\hat{s} \not\models \mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$ is a finite path $\sigma \in Paths^{\leqslant h}(\hat{s},\hat{t})$. A* strongest evidence *is an evidence σ' such that $\mathbb{P}(\sigma') \geqslant \mathbb{P}(\sigma)$ for any evidence σ.*

Definition 3 (Counterexample) *A* counterexample *for the violation $\hat{s} \not\models \mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$ is a set C of evidences such that $\mathbb{P}(C) > p$. C'' is a* smallest counterexample *if $|C''| \leqslant |C|$ for all counterexamples C and $\mathbb{P}(C'') \geqslant \mathbb{P}(C')$ for any counterexample C' with $|C'| = |C''|$.*

The intuition is that a smallest counterexample exceeds the required probability bound the most given that it has the smallest number of paths.

Example 1 *Consider the* DTMC *in Fig. 1 for which initial state s violates $\mathcal{P}_{\leqslant \frac{1}{2}}(\Diamond\,\hat{t})$. Evidences, among others, are: $\sigma_1 = \hat{s}s_1s_3\hat{t}$, $\sigma_2 = \hat{s}s_1s_2s_4\hat{t}$, and $\sigma_3 = \hat{s}s_2s_3\hat{t}$, $\sigma_4 = \hat{s}s_1s_2s_4\hat{t}$, and $\sigma_5 = \hat{s}s_2s_4\hat{t}$. Their respective probabilities are 0.2, 0.2, 0.15, 0.12, and 0.09. Strongest evidences are paths σ_1 and σ_2. The set $C_1 = \{\sigma_1,\ldots,\sigma_5\}$ with $\mathbb{P}(C_1) = 0.76$ is a counterexample, but not a smallest one, as the removal from either σ_1 or σ_2 also yields a counterexample. $C_2 = \{\sigma_1,\sigma_2,\sigma_3\}$ is a smallest counterexample with $\mathbb{P}(C_2) = 0.55$.*

In the sequel, we assume w.l.o.g. that goal state $\hat{t}$ is absorbing. To compute the strongest evidence and smallest counterexample, the DTMC $\mathcal{D}$ is transformed into a weighted digraph $\mathcal{G}_{\mathcal{D}} = (V,E,w)$, where V and E are finite sets of vertices and edges, respectively. V equals the state space of the DTMC, i.e., $V = S$ and $(v,v') \in E$ iff $\mathbf{P}(v,v') > 0$, and $w(v,v') = \log(\mathbf{P}(v,v')^{-1})$. Multiplication of transition probabilities is thus turned into the addition of edge weights along paths.

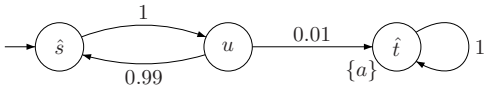
The weight of a finite path in $\mathcal{G}_{\mathcal{D}}$ is the sum of all the edge weights on it. A k-th shortest ($\hat{s}$-$\hat{t}$) path has the k-th least path weight than other $\hat{s}$-$\hat{t}$ paths, for $k \in \mathbb{N}_{>0}$ and if such paths exist. It is not necessarily unique. The same applies to k-th most probable paths in $\mathcal{D}$. Now:

Proposition 1 *[15] For any path $\sigma \in Paths^{\leqslant h}(\hat{s},\hat{t})$ in* DTMC *$\mathcal{D}$ and $k \in \mathbb{N}_{>0}$: σ is a k-th most probable path in $\mathcal{D}$ iff σ is a k-th shortest path in $\mathcal{G}_{\mathcal{D}}$.*

If $\hat{s} \not\models \mathcal{P}_{\leqslant p}(\Diamond^{\leqslant h}\,\hat{t})$, then a strongest evidence can be found by applying a shortest path (SP) algorithm to $\mathcal{G}_{\mathcal{D}}$. Similarly, a smallest counterexample can be determined by k-SP algorithms that allow k to be determined on-the-fly. The time complexity of the first algorithm is $\mathcal{O}(m+n\log n)$ and that of the second is $\mathcal{O}(m+n\log n+k)$, where $n = |S|$ and $m = |\mathbf{P}|$ is the number of non-zero entries in $\mathbf{P}$. For bounded until, hop-constrained SP (HSP) and k-HSP algorithms can be applied to compute the strongest evidences and smallest counterexamples, respectively. For details, cf. [13, 21, 15]. We have implemented the above algorithms and will report on some experimental results below.

3. Motivation

Smallest counterexamples may contain an excessive number of evidences, which is illustrated by the violation of $\hat{s} \models \mathcal{P}_{\leqslant 0.9999}(\Diamond\,\hat{t})$ in the following DTMC. The smallest counterexample consists of the evidences $\hat{s}(u\hat{s})^0u\hat{t}$, ..., $\hat{s}(u\hat{s})^{k-1}u\hat{t}$, where $(u\hat{s})^i$ is a short form of traversing the loop $\hat{s}u\hat{s}$ for i times and k is the smallest integer such that $1-0.99^{k-1} > 0.9999$ holds. As a result, the smallest counterexample has $k = 689$ evidences. In fact, the large number of evidences degrades the significance of each evidence.



To illustrate that such phenomena also occur in real-life case, we consider the generation of counterexamples for a more practical case study — *synchronous leader election protocol* [20]. In this protocol, N processes are arranged in a unidirectional ring to elect a leader. For this purpose, they randomly select an identity (id, for short) according to a uniform distribution on $\{1,\ldots,K\}$. We call each such selection by all processes a *configuration*. By means of synchronous message passing, processes send their ids around the ring till every process sees all the ids of the others, and can thus determine whether a leader (the one with the highest unique id) can be elected. If yes, the protocol terminates; if no, a new round will be started.

We intend to find a counterexample for the following formula: $\mathcal{P}_{\leqslant p}(\Diamond\, leader\_elected)$, where $leader\_elected$ characterizes the global state of the protocol in which a leader has been selected. It is clear that a leader will be elected

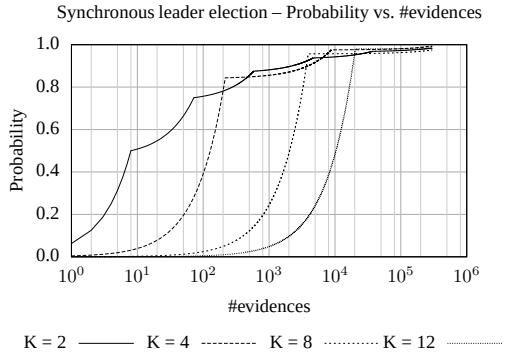


Figure 2. Probability vs. number of evidences for leader election (N = 4)

eventually. What interests us, is the number of evidences needed to converge to probability 1. Especially, we are interested in the relationship between the number of evidences and the bound p and R, where R is the round number. Starting a new round means that each process re-selects an id and repeats the procedure.

3.1. Experimental results

To find the number of evidences contained in a counterexample, we used the PRISM-model of the protocol [1] and ran the counterexample generation using our implemented algorithm. The results for a fixed N ($N = 4$) and varying K are depicted in Fig. 2 and partly in Table 1. In Fig. 2, the Y-axis is the accumulated probability and the X-axis (log-scale) is the number of evidences that are contained in a counterexample. The discontinuities in the curves correspond to the start of a new round, i.e., a new election, in the protocol. Due to the fact that the probability of all evidences in one round is the same, the curves in Fig. 2 are actually piecewise linear if the X-axis were not log-scale. The curves shift more to the right when K increases since there are more possible configurations and thus more evidences. The larger K, the more quickly the probability of the counterexample approaches 1. This is due to the fact that it is less probable that no process selects a unique id. All curves approach 1, which indicates that eventually a leader will be elected. The number of evidences in a counterexample, however, grows drastically to millions.

Table 1 shows some detailed numbers. The results in those $^+$-marked rounds are obtained from our experiments. The $^-$-marked rounds run out of memory and the numbers in this case are obtained from our mathematical analysis (see below). Note that in each round, the probability of having elected a leader (Prob. mass) decreases drastically, while the number of evidences increases rapidly, thus the probability per-evidence decreases tremendously.

	$N=4, K=2$			
	#Evidences	Prob. mass	Per-Evi mass	Cum.Pr.
R1$^+$	8	0.5	0.0625	0.5
R2$^+$	64	0.25	0.0039063	0.75
R3$^+$	512	0.125	0.00024414	0.875
R4$^+$	4 096	0.0625	$1.52588\cdot10^{-05}$	0.9375
	$N=4, K=4$			
	#Evidences	Prob. mass	Per-Evi mass	Cum.Pr.
R1$^+$	216	0.84375	0.0039063	0.84375
R2$^+$	8 640	0.13184	$1.52588\cdot10^{-05}$	0.97559
R3$^+$	345 600	0.020599	$5.96046\cdot10^{-08}$	0.99619
R4$^-$	$1.3824\cdot10^{07}$	0.0032187	$2.32831\cdot10^{-10}$	0.99940
	$N=4, K=8$			
	#Evidences	Prob. mass	Per-Evi mass	Cum.Pr.
R1$^+$	3 920	0.95703	$2.4414\cdot10^{-04}$	0.95703
R2$^-$	689 920	0.041122	$5.9605\cdot10^{-08}$	0.99815
R3$^-$	$1.2143\cdot10^{08}$	0.0017669	$1.4552\cdot10^{-11}$	0.99992
R4$^-$	$2.1371\cdot10^{10}$	$7.5925\cdot10^{-05}$	$3.5527\cdot10^{-15}$	1
	$N=4, K=12$			
	#Evi	Prob. mass	Per-Evi mass	Cum.Pr.
R1$^+$	20 328	0.98032	$4.8225\cdot10^{-05}$	0.98032
R2$^-$	$8.2938\cdot10^{06}$	0.019289	$2.3257\cdot10^{-09}$	0.99961
R3$^-$	$3.3839\cdot10^{09}$	$3.79525\cdot10^{-04}$	$1.1216\cdot10^{-13}$	0.99999
R4$^-$	$1.3806\cdot10^{12}$	$7.4675\cdot10^{-06}$	$5.4088\cdot10^{-18}$	1

Table 1. Number of evidences and probability mass per round

3.2. Mathematical analysis

To obtain more insight into this rapid growth of the size of a counterexample, we carry out a brief combinatorial analysis. Let us first consider the number of possibilities (denoted $W(N,K)$) of putting N *labeled* balls into K *labeled* boxes such that each box contains at least two balls. Actually, $W(N,K)$ characterizes the number of possibilities of assigning K ids to N processes such that each id is assigned to more than one process, in which case a leader is not selected. $W(N,K)$ can be solved by using the "associated Stirling number of the second kind (S_2)" [9]:

$$W(N,K) = \sum_{j=1}^{\min(\lfloor N/2 \rfloor, K)} S_2(N,j)\frac{K!}{(K-j)!}, \quad (1)$$

where $S_2(N,K) = K\cdot S_2(N-1,K) + (N-1)\cdot S_2(N-2,K-1)$ indicates the number of ways to put N labeled balls into K *unlabeled* boxes. Obviously, it makes no sense to have more than $\lfloor N/2 \rfloor$ boxes, or else it would be impossible to allocate all the balls in the right way. The factor $\frac{K!}{(K-j)!}$ expresses that there are $K!$ ways to permute the boxes (including the empty ones); for these empty boxes the order does not matter, so we divide by $(K-j)!$.

The non-recursive equation for $S_2(N,K)$ is:

$$S_2(N,K) = \sum_{i=0}^{K} (-1)^i \binom{N}{i} \left(\sum_{j=0}^{K-i} (-1)^j \frac{(K-i-j)^{N-i}}{j!(K-i-j)!} \right). \quad (2)$$

For each round in the leader election protocol, the number of possibilities for a process to choose an id is K^N. Thus,

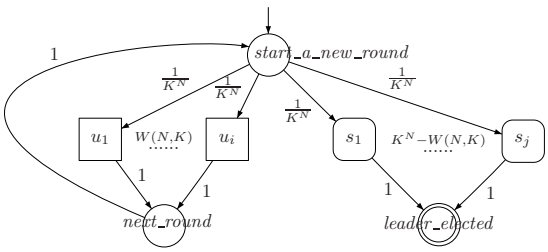


Figure 3. Abstract leader election model

the probability that N processes with K ids elect a leader in round R, denoted by $P(N,K,R)$, is:

$$P(N,K,R) = \left(\frac{W(N,K)}{K^N}\right)^{R-1} \frac{K^N - W(N,K)}{K^N}, \quad (3)$$

where $\left(\frac{W(N,K)}{K^N}\right)^{R-1}$ is the probability that a leader is not elected in the first $(R-1)$ rounds and $\frac{K^N-W(N,K)}{K^N}$ indicates the probability that a leader is elected in the R-th round.

We now calculate the probabilities of each evidence per round using equation (3). The model of the synchronous leader election protocol is depicted in Fig. 3. When we *start_a_new_round*, there are K^N possible configurations, among which $W(N,K)$ (square states, *u*nsuccessful) configurations no unique id will be selected. For these states, we start the *next_round*, while in $K^N - W(N,K)$ (round-angle states, *s*uccessful) configurations a unique id will be selected with a *leader_elected*. Thus:

Proposition 2 *The number of evidences that can reach the state* $leader\_elected$ *in round* R *is:*

$$\#Evi(N,K,R) = W(N,K)^{R-1} \cdot \left(K^N - W(N,K)\right).$$

Proposition 2 shows that the number of evidences is exponential in R. Note that $W(N,K)$ is exponential in N and K, which makes $\#Evi(N,K,R)$ doubly exponential. These results coincide with Table 1, where the $^-$-rounds are computed by the above closed form expression.

The number of evidences thus grows extremely fast. This results in two problems. First, it leads to the storage problem as counterexamples may simply get too large to be kept in memory. Secondly, and more importantly, counterexamples will be incomprehensible to the user. We therefore need to find ways to reduce the number of evidences in a counterexample, and to obtain a compact and user-friendly representation. To that purpose we suggest to use *regular expressions*.

4. Regular expressions for counterexamples

This approach is inspired by classical automata theory and is based on representing sets of paths by regular expressions. A major difference with usual regular expressions is that we need to keep track of the transition probabilities. To tackle this, we adopt the approach proposed by Daws [10]. He uses regular expressions to represent sets of paths and calculates the *exact rational value* of the probability measure in DTMC model checking. We adapt this approach to obtain compact representations of counterexamples. The main idea is to consider a counterexample as a set of probable branches (sub-expressions) that go from the initial state to the goal state and to provide a function to evaluate the probability measure of those expressions.

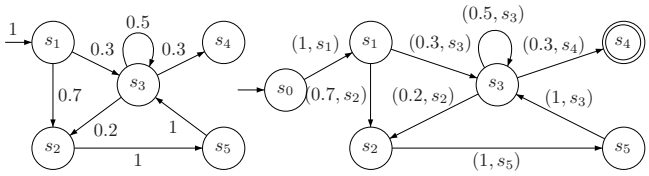


Figure 4. DTMC $\mathcal{D}$ **and its** DFA $\mathcal{A}_\mathcal{D}$

4.1. From DTMCs to automata

For DTMC $\mathcal{D} = (S, \mathbf{P}, \hat{s})$ and property $\Diamond^{\leqslant h}\, \hat{t}$, let deterministic finite automaton (DFA) $\mathcal{A}_\mathcal{D} = (S', \Sigma, \tilde{s}, \delta, \hat{t})$, where:

- $S' = S \cup \{\tilde{s}\}$ is the state space;
- $\Sigma \subset [0,1] \times S$ is the finite alphabet;
- $\delta \subseteq S' \times \Sigma \times S'$ is the transition relation such that $\delta\,(s_1, (p, s_2)) = s_2$, if $\mathbf{P}(s_1, s_2) = p$ and $\delta(\tilde{s}, (1, \hat{s})) = \hat{s}$;
- $\tilde{s} \notin S$ is the starting state;
- $\hat{t}$ is the accepting state.

$\tilde{s}$ is a new starting state in $\mathcal{A}_\mathcal{D}$ connecting to the initial state $\hat{s}$ of $\mathcal{D}$ with a transition labeled with $(1, \hat{s})$. Different from [10], for transition $s_1 \xrightarrow{p} s_2$ we add the target state s_2 together with the probability p as the symbol (p, s_2) to Σ. The transition probabilities are needed to calculate the probability of the paths (see Def. 4), while the target states are needed for keeping track of the visited states. The latter is required as a regular expression only records the labels on the transitions, but not the states, which are, however, important for evidences. For the rest of the paper, the probability label p is sometimes omitted for simplicity when p is of minor importance. The maximal cardinality of the alphabet is $m+1$, where $m = |\mathbf{P}|$ and 1 is due to $\hat{s}$. Since there is only one transition from state s_1 to s_2, $\delta\,(s_1, (p, s_2)) = s_2$, the derived automaton is deterministic.

Example 2 *Fig. 4 (left) depicts an abstract example of a* DTMC $\mathcal{D}$ *with initial state* $\hat{s} = s_1$ *and goal state* $\hat{t} = s_4$*, and its* DFA $\mathcal{A}_\mathcal{D}$ *(right). The new starting state is* $\tilde{s} = s_0$*, which has a transition equipped with symbol* $(1, s_1)$.

4.2. Evaluation of regular expressions

The set $\mathcal{R}(\Sigma)$ of regular expressions over the finite alphabet Σ is the set of expressions containing the elements of Σ, the empty word ε, and which is closed under union ($|$), concatenation (.) and Kleene star ($*$).

Let $\mathcal{L}(r)$ denote the regular language (a set of words) described by the regular expression $r \in \mathcal{R}(\Sigma)$ and $\mathcal{L}(\Sigma)$ denote the regular language that can be generated by any regular expression over Σ. For a word w, $|w|$ denotes the number of symbols in w. We sometimes omit . and write $r.r'$ as rr' for short. Note that in our setting, Σ contains elements of the form (p, s_1) where $p \in [0, 1]$ and $s_1 \in S'$.

Definition 4 ([10]) *The regular expressions can be evaluated by the function $val : \mathcal{R}(\Sigma) \mapsto \mathbb{R}$ as:*

$$val(\varepsilon) = 1 \quad val(r|r') = val(r) + val(r')$$
$$val((p, s)) = p \quad val(r.r') = val(r) \times val(r')$$

$$val(r^*) = \begin{cases} 1, & \text{if } val(r) = 1 \\ \frac{1}{1-val(r)}, & \text{otherwise} \end{cases}$$

If we limit the transition probabilities to be rational values, then we will obtain exact values. It can be proven that $val(r) = \mathbb{P}\big(Paths_{\mathcal{D}}^{\leqslant h}(\hat{s}, \hat{t})\big)$, for $h = \infty$ [10].

Definition 5 *r_1 is a* maximal union subexpression *(*MUS*) of a regular expression r if $r = r_1 \mid r_2$ modulo $(\mathbf{R_1})$-$(\mathbf{R_3})$, for some $r_2 \in \mathcal{R}(\Sigma)$, where:*

$$(\mathbf{R_1}) \quad r \equiv r \mid \varepsilon$$
$$(\mathbf{R_2}) \quad r_1 \mid r_2 \equiv r_2 \mid r_1$$
$$(\mathbf{R_3}) \quad r_1 \mid (r_2 \mid r_3) \equiv (r_1 \mid r_2) \mid r_3$$

r_1 is maximal because it is at the topmost level of a union operator. Note that if the topmost level operator is not union, then $r_1 = r$ (cf. $\mathbf{R_1}$). A regular expression represents a set of paths and each MUS can be regarded as a main branch from the initial state to the accepting state.

Example 3 *A regular expression for the automaton $\mathcal{A}_{\mathcal{D}}$ in Fig. 4 (right) is:*

$$r_0 = \underbrace{s_1 s_3 s_3^* s_4}_{r_1} \mid \underbrace{s_1(s_2|s_3 s_3^* s_2)(s_5 s_3 s_3^* s_2)^* s_5 s_3 s_3^* s_4}_{r_2}.$$

r_1 and r_2 are the MUS*s of r_0 with $val(r_1) = 1 \times 0.3 \times \frac{1}{1-0.5} \times 0.3 = 0.18$ and $val(r_2) = 0.82$. We can distribute $\mid$ over . in r_2 and obtain two more* MUS*s instead: $r_3 = s_1 s_2 (s_5 s_3 s_3^* s_2)^* s_5 s_3 s_3^* s_4$ and $r_4 = s_1 s_3 s_3^* s_2 (s_5 s_3 s_3^* s_2)^* s_5 s_3 s_3^* s_4$. r_1, r_3 and r_4 characterize all paths from s_1 to s_4, which fall into the above three branches. Note that r_1* cannot *be written as $s_1 s_3^+ s_4$, since from the full form of $r_1 = (1, s_1)(0.3, s_3)(0.5, s_3)^*(0.3, s_4)$, the probability of the first s_3 is different from that of s_3^*.*

4.3. From automata to regular expressions

The equivalence of DFAs and regular expressions, as well as converting DFAs to regular expressions has been widely studied. Several techniques are known, e.g., the transitive closure method [24], Brzozowski's algebraic method [7][6], or the state removal method [12][28]. The state removal approach identifies patterns within the automaton and removes states one by one, while building up regular expressions along each transition. It is suitable for manual inspection but less straightforward to implement. The transitive closure method gives a clear and simple implementation but tends to create very long regular expressions. The algebraic method is elegant and generates reasonably compact regular expressions. For a more detailed comparison, we refer to [29]. In our setting, in order to obtain a minimal counterexample *on-the-fly*, the state elimination method is taken. To be more precise, the algebraic method will not terminate before it generates the whole regular expression, however, it is possible that a regular expression has many MUSs (like in the leader election example shown below) and thus takes much longer time to terminate. On the other hand, the state elimination method can be terminated after each state elimination and be resumed if more MUSs are needed.

4.4. Regular expressions as counterexamples

By using regular expressions for representing counterexamples, we will, instead of obtaining evidences one by one, derive a larger number of evidences at a time, which hopefully yields a quick convergence to the required probability threshold and a clear explanation of the violation. As a result, we will not insist on obtaining the smallest counterexample but would instead prefer finding the branches (MUSs) with large probabilities and short length. In other words, three properties of the regular expressions are preferred:

1. shorter (wrt. the number of symbols it contains), to improve comprehensibility;
2. more probable, such that it is more informative and the algorithm will terminate with less MUSs;
3. minimal, where a counterexample is *minimal* if the omission of any of its MUSs would no longer result in a counterexample.

However, it has been recently proven that the size of a shortest regular expression of a given DFA cannot be efficiently approximated (if P≠PSPACE) [14]. Therefore, it is not easy to, e.g., by state elimination, compute an optimal removal sequence for state elimination in polynomial time [17]. We could adapt the heuristics proposed in e.g. [17][11] to get a better order to eliminate states. For 2), we could take the advantage of the k-SP or k-HSP algorithms as well as the model-checking results. The states on the more probable evidences should be eliminated first.

We take the following iterative strategy: In each iteration, we take the strongest evidence $\sigma{=}\tilde{s}\hat{s}s_1\cdots s_j\hat{t}$ in the remaining automaton — recall that this amounts to an SP problem — and eliminate all the intermediate states on σ (i.e., $s_1, \ldots, s_j$) one by one according to an order that is recently proposed in [17]. After eliminating each state, it is possible that a new MUS r_k is created and its $val(r_k)$ can be calculated, where k MUSs have been created so far. If $\sum_{i=1}^{k} val(r_i) > p$, then the algorithm terminates; else the transition labeled with r_k is removed from the automaton and either the next state is to be eliminated or a new evidence is to be found. The removal of r_k is to concentrate on the rest of the automaton that has not been explored yet. The sketch of the algorithm is shown in Algorithm 1.

Algorithm 1 *RegExpCE*$(\mathcal{A}_{\mathcal{D}}, p)$: Calculate the regular expression counterexample

Require: automaton $\mathcal{A}_{\mathcal{D}}$, probability bound p, initial state $\tilde{s}$, accepting state $\hat{t}$
Ensure: a regular expression r with $val(r) > p$

```
1: A := A_D; pr := 0; Priority queue q := ∅; k := 1;
2: while pr ⩽ p do                          \* the termination criterion *\
3:    σ := the strongest evidence in A;
4:    forall s' ∈ σ \ {s̃, ŝ, t̂} do  q.enqueue(s'); endforall;
5:    while q ≠ ∅ do
6:       A :=eliminate(q.dequeue());   r_k := the created MUS;
7:       pr := pr+val(r_k);   A :=eliminate(r_k);
8:       if (pr > p)  then break;  else k := k + 1;
9:    endwhile;
10: endwhile;
11: return r_1, ..., r_k;
```

Note that q is a priority queue whose elements are states to be eliminated in the current iteration. The order in which states are dequeued from q is defined by the heuristics provided in [17]. The function "eliminate(·)" can both eliminate states and regular expressions, where the latter is simply the deletion of the transitions labeled with the regular expressions.

Example 4 *Consider again Fig. 4. We suppose the formula is* $\mathcal{P}_{\leqslant 0.7}(\Diamond\, s_4)$ *and apply the algorithm on* $\mathcal{A}_{\mathcal{D}}$. *In the first iteration,* $\sigma_1{=}s_0s_1s_2s_5s_3s_4$ *is found as the strongest evidence. Suppose the order to eliminate the states by [17] is* s_5, s_2, s_3, *then we get the regular expression* $r_5{=}s_1(s_3|s_2s_5s_3)(s_3|s_2s_5s_3)^*s_4$ *with* $val(r_5){=}1$. *Since the states are eliminated and the threshold* 0.7 *is exceeded, the algorithm terminates. This expression gives a clear reason that infinitely many times traversing the cycles* s_3 *or* $s_2s_5s_3$ *exceeds* 0.7.

Let us change the elimination order. *If the elimination order is* s_5, s_3, s_2, *the regular expression is* $r_0 = s_1s_3s_3^*s_4 \mid s_1(s_2|s_3s_3^*s_2)(s_5s_3s_3^*s_2)^*s_5s_3s_3^*s_4$. *When* s_3 *has been eliminated, the first* MUS $r_1 = s_1s_3s_3^*s_4$ *is generated and the probability is* $0.18 < 0.7$. *The algorithm continues (i.e., eliminates* s_2*) to find more* MUSs, *till* r_0 *is found. Note that* r_0 *is longer than* r_5, *and thus less intuitive to comprehend. The cycles* s_3 *and* $s_3s_2s_5$ *are however indicated.*

Let us pick a less probable evidence $\sigma_2{=}s_0s_1s_3s_4$ *to be eliminated in the first iteration. After eliminating* s_3, *the resulting expression is* $r_1{=}s_1s_3s_3^*s_4$. *Then* r_1 *is removed from the automaton and the strongest evidence in the remaining automaton is* $\sigma_3{=}s_0s_1s_2s_5s_4$. *After eliminating* s_2, s_5, *we obtain the regular expression:* r_2, *as in Example 3. The final regular expression is again* r_0 *and the same analysis as in the last case applies.*

Proposition 3 *The regular expression counterexample generated by Alg. 1 is minimal.*

The above proposition is due to the fact that Alg. 1 terminates immediately when the cumulative probability exceeds the threshold.

To summarize, we usually obtain a better regular expression if we pick the most probable path first and eliminate the states on that path according to some known heuristics. Note that the regular expression representation is not applicable for formulae with nested probabilistic operators, e.g., $\mathcal{P}_{\leqslant p_1}\big(\Diamond\, \mathcal{P}_{\leqslant p_2}(\Diamond\, \hat{t})\big)$. However, it is not a real constraint in practice, since those formulas are rarely used. Whereas in [15], formulas with a strict probability bound such as $\mathcal{P}_{<p}(\Diamond^{\leqslant h}\, \hat{t})$ could not be treated as they may lead to infinite counterexamples, this restriction does not apply any more due to the Kleene star.

4.5. Bounded reachability

For bounded reachability formula $\Diamond^{\leqslant h}\, \hat{t}$, a regular expression, e.g. $r = r_1|r_2^*$, may not be valid because it is possible that the length of the words generated by r_1 or the expansion of r_2 exceeds h. Thus, $val(r)$ might be larger than the actual probability. In order to obtain a precise valuation, we extend the regular expressions to *constrained regular expressions*, and extend the valuation to these expressions.

Definition 6 (Constrained regular expressions) *For* $r \in \mathcal{R}(\Sigma)$ *and* $h \in \mathbb{N}$, $\mathcal{L}(r[h]) = \{w \in \mathcal{L}(r) \mid |w| \leqslant h\}$.

In fact, $\mathcal{L}(r[h]) \subseteq \mathcal{L}(r)$ and $r[h]$ can be expressed equivalently by a union of possible enumerations, namely $r[h] = r\langle 0\rangle|r\langle 1\rangle|\cdots|r\langle h\rangle$, where $r\langle i\rangle$ denotes the set of words generated by r and having exactly i symbols.

Usually a constrained regular expression is informative enough to show the reason for violation, because the cycle information is clear. Sometimes, however, it is also useful to calculate the probability (or valuation) of some constrained regular expression branches by the following function:

Definition 7 *For $r, r_1, r_2 \in \mathcal{R}(\Sigma)$ and $(p,t) \in \Sigma$, the (recursive) evaluation functions $val(r[h])$ and $val(r\langle h\rangle)$ for $r[h]$ and $r\langle h\rangle$ respectively are defined by:*

$$
\begin{aligned}
val(r[h]) &= \sum_{i=0}^{h} val(r\langle h\rangle) \\
val(\varepsilon\langle h\rangle) &= \begin{cases} 1, & \text{if } h = 0 \\ 0, & \text{otherwise} \end{cases} \\
val((p,t)\langle h\rangle) &= \begin{cases} p, & \text{if } h = 1 \\ 0, & \text{otherwise} \end{cases} \\
val(r_1|r_2\langle h\rangle) &= val(r_1\langle h\rangle) + val(r_2\langle h\rangle) \\
val(r_1.r_2\langle h\rangle) &= \sum_{i=0}^{h} val(r_1\langle i\rangle) \cdot val(r_2\langle h-i\rangle) \\
val(r^*\langle h\rangle) &= val(\varepsilon\langle h\rangle) + \sum_{i=1}^{h} val(r\langle i\rangle) \cdot val(r^*\langle h-i\rangle)
\end{aligned}
$$

Note that the complexity of the above evaluation function is, however, very high. The following theorem states that the evaluation of a regular expression obtained from DFA $\mathcal{A}_{\mathcal{D}}$ equals the probability of the set of paths $Paths^{\leqslant h}(\hat{s}, \hat{t})$ in the DTMC $\mathcal{D}$.

Theorem 8 *Let r be the regular expression for DFA $\mathcal{A}_{\mathcal{D}} = (S, \Sigma, \tilde{s}, \delta, \hat{t})$ where $\mathcal{D} = (S, \mathbf{P}, \hat{s})$ and $h \in \mathbb{N}$. Then,*

$$val(r[h]) = \mathbb{P}\big(Paths_{\mathcal{D}}^{\leqslant h}(\hat{s}, \hat{t})\big).$$

4.6. Leader election example

Let us reconsider the leader election protocol. For the original DTMC, the regular expression representation, denoted $r(N,K)$, where the probabilities are omitted, is:

$$start.\left((u_1|\cdots|u_i)\,.next.start\right)^*.\left(s_1|\cdots|s_j\right).leader,$$

where $start$, $next$ and $leader$ are the obvious short forms. The regular expression lists all the unsuccessful configurations, as well as the successful ones. As a result, $|r(N,K)| = K^N + 4$, where $|r|$ denotes the number of symbols it consists of. Compared to the number of evidences computed directly, i.e., $\sum_{i=1}^{R} \#Evi(N,K,i)$, $|r(N,K)|$ is much shorter, but it is still exponentially long. On the other hand, however, the structure of $r(N,K)$ clearly indicates the reason of violation, i.e., the repeated unsuccessful configurations followed by a successful one.

Model reduction. Regular expression counterexamples are feasible when the excessive number of evidences are caused by traversing the same loops for different times in the model, in which case the Kleene star compacts all those evidences to be one MUS. On the other hand, the large number of states may also result in a large-size regular expression counterexample. Consequently, if the model size is reduced prior to the counterexample generation, then the thus obtained regular expression counterexample would be of smaller size. Two strategies can be utilized to slim down the model size, namely, *bisimulation minimization* and SCC *minimization*. Bisimulation minimization [22, 26, 8] groups the bisimilar states together and hopefully derives a smaller quotient DTMC. Strongly-connected-component (SCC) minimization, instead, groups SCCs together [27]. Bisimulation minimization preserves both unbounded and bounded probabilistic reachability properties, while SCC minimization only preserves the former one.

For the leader election protocol, the regular expression counterexample on the bisimulation quotient DTMC is:

$$r_{\sim}(N,K) = start.\left(u.next.start\right)^*.s.leader,$$

where $u_1, \ldots, u_i$ are wrapped as u; $s_1, \ldots, s_j$ as s in Fig. 3. Note that $|r_{\sim}(N,K)| = 6$ is independent of N and K. The SCC-quotient DTMC is obtained by replacing the left half of the model (an SCC) by a self-loop on the initial state. The regular expression counterexample is:

$$r^{\text{SCC}}(N,K) = start.start^*.(s_1|\cdots|s_j).leader,$$

where the intuition of the self-loop is "still unsuccessful". We can gain more if both reduction techniques are applied, yielding:

$$r_{\sim}^{\text{SCC}}(N,K) = start.start^*.s.leader.$$

5. Case study – Crowds protocol

We now illustrate our techniques on a more serious example. The *Crowds* protocol [30] is aimed to provide users with a mechanism for anonymous Web browsing. The main idea behind Crowds is to hide each user's communication by routing randomly within a group of similar users. Even if a local eavesdropper or a corrupt (or bad) group member observes a message being sent by a particular user, it can never be sure whether the user is the actual sender, or is simply routing another user's message.

The protocol works in the following way: 1) The sender selects a crowd member at random (possibly itself), and forwards the message to it, encrypted by the corresponding pairwise key. 2) The selected router flips a biased coin. With probability $1 - PF$, where *PF* (forwarding probability) is a parameter of the system, it delivers the message directly to the destination. With probability *PF*, it selects a crowd member at random (possibly itself) as the next router in the path, and forwards the message to it, re-encrypted with the appropriate pairwise key. The next router repeats this step.

In our experiments, we assume that 1) if a sender has been observed by the bad member twice, then it has been *positively identified* (*Pos* for short), thus the anonymity is not preserved; 2) the bad member will deliver the message with probability 1 as in [31]. This protocol is executed every time one crowd member wants to establish an anonymous connection to a Web server. We call one run of the

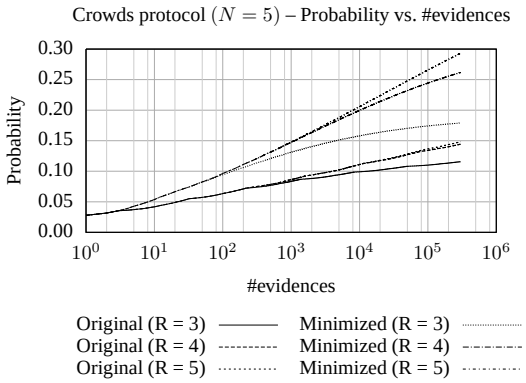


Figure 5. Probability vs. number of evidences

protocol a *session* and denote the number of sessions by R. Other parameters are the number of good members N and the number of corrupt members C.

We take the Crowds protocol modeled by Prism [1] and the property is $\mathcal{P}_{\leqslant p}(\Diamond\, \mathit{Pos})$ which characterizes the probability threshold that the original sender's id 0 is positively identified by the corrupt members. The relation between the number of evidences and the probability threshold for different number of sessions R is shown in Fig. 5 ($N = 5$, $C = 1$, $PF = 0.8$), both for the original and the bisimulation minimized DTMCs.

Regular expression representation Because we manually show the effect of the regular expression representation here, we choose a configuration with a small state space. We set $N = 2$, $C = 1$, $R = 2$, and $PF = 0.8$. The bisimulation minimization reduces the state space from 77 to 34. The state space of the quotient DTMC is shown in Fig. 6. To fit the figure on the page, we group a path of states with probability 1 by a square state. States i, G, B, *Del*, *Pos* represent *i*nitiating a new session, sending a message to a *G*ood member, to a *B*ad member, a message being *Del*ivered, a *Pos*itive result obtained, respectively. $G0$ and $G1$ are the two good members, where $G0$ is assumed always to be the original sender when a new session starts. $G0 \vee G1$ is a lumped state where either $G0$ or $G1$ is reached. The subscripts $a, b, \ldots$ are to distinguish the states in similar situations. Since the goal state *Pos* can be reached by only the gray states, the regular expression (thus the automaton) only depends on those states. Note that Del_a and Del_b denote the end of the first session, while Del_c and Del_b denote the end of the second. Only the case that two messages are both delivered by the bad member indicates a positive identification of the sender.

An intermediate automaton (see Fig. 7) can be derived after eliminating some states. The initial state $\acute{s}$ of the automaton is also omitted. This shows the basic structure of the model: i_a and i_c are the starting points of two sessions. The horizontal transitions indicate the observation of $G0$ by the bad member, which lead to *Pos*. In each session, a message can be forwarded to $G0$ or $G1$ many times (captured by the self loops). Once a message is delivered, a new session is assumed to be started (the transitions back to i_a and i_c). Thus, a regular expression that can be generated from the automaton is $r = r_0 r_1^* r_2 r_3^* r_4$, where:

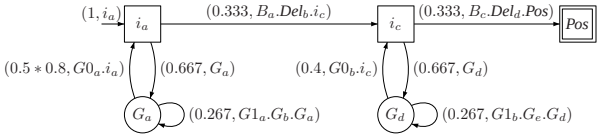


Figure 7. A more compact automaton

$$
\begin{aligned}
r_0 &= (1, i_a),\\
r_1 &= (0.667, G_a)(0.267, G1_a.G_b.G_a)^*(0.4, G0_a.i_a),\\
r_2 &= (0.333, B_a.\mathit{Del}_b.i_c),\\
r_3 &= (0.667, G_d)(0.267, G1_b.G_e.G_d)^*(0.4, G0_b.i_c),\\
r_4 &= (0.333, B_c.\mathit{Del}_d.\mathit{Pos}).
\end{aligned}
$$

If we omit the probabilities and the subscripts and merge the stuttering steps G, then we obtain:

$$r' = i\,\underbrace{(G.(G1.G)^*G0.i)^*}_{good}.\underbrace{(B.\mathit{Del}.i)}_{bad}.\underbrace{(G.(G1.G)^*G0.i)^*}_{good}.\underbrace{B}_{bad},$$

which is highly compact and informative in the sense that it indicates the observation of the bad members twice with arbitrary number of observing the good members. r' can be further compacted if the SCCs are identified and replaced by self-loops. In this case, $r'' = i.i^*.(B.\mathit{Del}.i).i^*.B$.

The probability of r is $val(r) = 0.274$, which coincides with the model checking result. These probabilities depend, among others, on the parameters of the protocol (N, C, R, *PF*, etc.). For instance, the probability of the strongest evidence is $(\frac{C}{N+C})^R = (\frac{1}{3})^2 = \frac{1}{9}$, which loops 0 times at r_1 and r_3. The probability of r_2 and r_4 is $\frac{a}{1-a} = \frac{4}{11}$, where a is the probability of the inner loop: $\frac{1}{N+C} \cdot \mathit{PF} \cdot (1 - \frac{C}{N+C}) = 0.267$, as is shown in the intermediate automaton. Note that this closed-form expression can now be used for arbitrary parameter values.

6. Conclusion

The contributions of this paper are: experimental results on the generation of counterexamples for model-checking DTMCs, partly substantiated with a mathematical analysis, together with the proposal to use the regular expressions to represent counterexamples in a compact way. The counterexample representation using regular expressions, is shown to be correct, and yields promising results. Bisimulation and SCC minimization may be explored to slim down the counterexample size. Constrained regular expressions for bounded-until formulae is a topic for further study as their valuation is expensive.

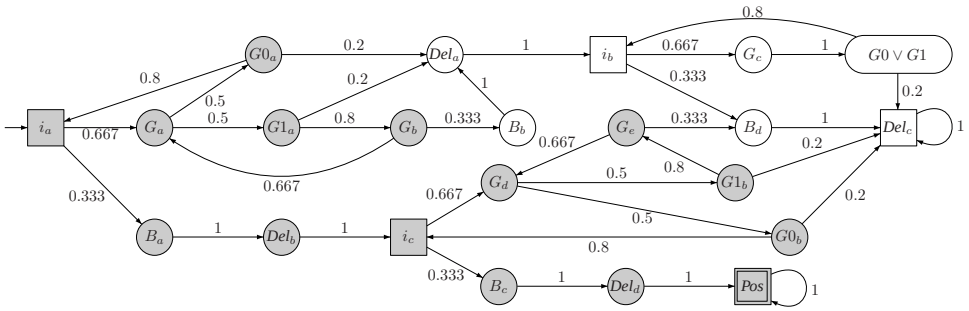


Figure 6. State space of the quotient DTMC **for the Crowds protocol (*N = 2*, *C = 1*, *R = 2*, *PF= 0.8*)**

Acknowledgement. Daniel Klink and Alexandru Mereacre are kindly acknowledged for their useful remarks. This research is financially supported by the Dutch NWO project QUPES, and the EU FP7 project QUASIMODO.

References

[1] PRISM website. `http://www.prismmodelchecker.org`.

[2] H. Aljazzar, H. Hermanns, and S. Leue. Counterexamples for timed probabilistic reachability. In *FORMATS, LNCS 3829*, pages 177–195, 2005.

[3] H. Aljazzar and S. Leue. Extended directed search for probabilistic timed reachability. In *FORMATS, LNCS 4202*, pages 33–51, 2006.

[4] C. Andersson, S. Fischer-Hübner, and R. Lundin. Enabling anonymity for the mobile Internet using the mCrowds system. In *IFIP WG 9.2, 9.6/11.7 Summer School on Risks and Challenges of the Network Society*, page 35, 2004.

[5] M. E. Andrés and P. van Rossum. Conditional probabilities over probabilistic and nondeterministic systems. In *TACAS, LNCS 4963*, pages 157–172, 2008.

[6] G. Berry and R. Sethi. From regular expressions to deterministic automata. *TCS*, 48(3):117–126, 1986.

[7] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[8] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *J. of Appl. Prob.*, pages 59–75, 1994.

[9] L. Comtet. *Advanced Combinatorics: The Art of Finite and Infinite Expansion*. D.Reidel Publishing Company, 1974.

[10] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC, LNCS 3407*, pages 280–294, 2004.

[11] M. Delgado and J. Morais. Approximation to the smallest regular expression for a given regular language. In *CIAA, LNCS 3317*, pages 312–314, 2004.

[12] D.-S. Du and K.-I. Ko. *Problem Solving in Automata, Languages, and Complexity*. John Wiley and Sons, 2001.

[13] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[14] G. Gramlich and G. Schnitger. Minimizing NFA's and regular expressions. *J. Comput. Syst. Sci.*, 73(6):908–923, 2007.

[15] T. Han and J.-P. Katoen. Counterexamples in probabilistic model checking. In *TACAS, LNCS 4424*, pages 72–86, 2007.

[16] T. Han and J.-P. Katoen. Providing evidence of likely being on time: Counterexample generation for CTMC model checking. In *ATVA, LNCS 4762*, pages 331–346, 2007.

[17] Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theor. Comput. Sci.*, 370(1-3):110–120, 2007.

[18] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *FACS*, 6(5):512–535, 1994.

[19] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *CAV, to appear*, 2008.

[20] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.

[21] V. M. Jiménez and A. Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *WAE, LNCS 1668*, pages 15–29, 1999.

[22] J.-P. Katoen, T. Kemna, I. Zapreev, and D. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *TACAS, LNCS 4424*, pages 87–101, 2007.

[23] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *QEST*, pages 243–244, 2005.

[24] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–42. Princeton Univ. Press, 1956.

[25] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *STTT*, 6(2):128–142, 2004.

[26] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

[27] H. le Guen and R. A. Marie. Visiting probabilities in non-irreducible Markov chains with strongly connected components. In *ESM*, pages 548–552, 2002.

[28] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartless Publishers, Sudbury, MA, 2001.

[29] C. Neumann. Converting deterministic finite automata to regular expressions. `http://neumannhaus.com/christoph/papers/2005-03-16.DFA_to_RegEx.pdf`, 2005.

[30] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM TISSEC*, 1(1):66–92, 1998.

[31] V. Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3-4):355–377, 2004.

[32] A. Vaha-Sipila and T. Virtanen. BT-Crowds: Crowds-style anonymity with Bluetooth and Java. In *HICSS*, 2005.