Logical Methods in Computer Science
Vol. 11(1:15)2015, pp. 1–23
www.lmcs-online.org

Submitted Jan. 6, 2014
Published Mar. 31, 2015

# HIGH-LEVEL COUNTEREXAMPLES FOR PROBABILISTIC AUTOMATA

RALF WIMMER [a], NILS JANSEN [b], ERIKA ÁBRAHÁM [c], AND JOOST-PIETER KATOEN [d]

[a] Albert-Ludwigs-Universität Freiburg, Germany
*e-mail address*: wimmer@informatik.uni-freiburg.de

[b,c,d] RWTH Aachen University, Germany
*e-mail address*: {nils.jansen | abraham | katoen}@informatik.rwth-aachen.de

Abstract. Providing compact and understandable counterexamples for violated system properties is an essential task in model checking. Existing works on counterexamples for probabilistic systems so far computed either a large set of system runs or a subset of the system's states, both of which are of limited use in manual debugging. Many probabilistic systems are described in a guarded command language like the one used by the popular model checker `PRISM`. In this paper we describe how a smallest possible subset of the commands can be identified which together make the system erroneous. We additionally show how the selected commands can be further simplified to obtain a well-understandable counterexample.

## 1. Introduction

One of the main strengths—perhaps *the* key feature—of model checking is its possibility to automatically generate a counterexample in case a model refutes a given property [1]. Counterexamples provide essential diagnostic information for debugging purposes. They also play an important role in counterexample-guided abstraction refinement (CEGAR) [2], a successful technique in software verification. In this iterative abstraction-refinement process, abstractions that are too coarse are refined with the help of counterexamples. Single system runs—typically acquired during model checking—suffice as counterexamples for linear-time properties. For branching-time logics such as CTL and CTL$^*$, more general shapes are necessary, such as tree-like counterexamples [3].

*2012 ACM CCS:* [**Theory of computation**]: Logic—Verification by model checking.

*Key words and phrases:* Probabilistic automata, Counterexamples, Guarded command language, Mixed integer linear programming.

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center AVACS (SFB/TR 14), the DFG project CEBug (AB 461/1-1), and the EU-FP7 IRSES project MEALS. Also funded by the Excellence Initiative of the German federal and state government.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(1:15)2015

This paper focuses on counterexamples for *probabilistic automata* [4], shortly labeled transition systems in which transitions yield distributions over states (rather than just states). A violating behavior in this setting entails that the (maximal) probability that a certain property $\varphi$ holds, such as a deadlock state is reachable, is outside of some required bounds. For probabilistic reachability properties, it suffices to treat violations of upper bounds [5]. Counterexamples consist of a finite set of finite runs that all satisfy the property $\varphi$ while their combined probability mass exceeds the required upper bound. In contrast to some traditional model checking algorithms for LTL and CTL, probabilistic counterexamples are not obtained as a by-product of the verification process. Instead, dedicated counterexample generation algorithms are used.

In the last years, several approaches have been proposed for probabilistic counterexample generation. Enumerative approaches [6, 5, 7] generate a set of finite paths based on $k$ shortest path algorithms possibly enhanced with heuristic search and/or SAT-based techniques. Such counterexamples can be succinctly represented by, e. g., regular expressions [5] or in a hierarchical manner [8, 9] using a graph decomposition of the Markov chain into strongly connected components. An alternative is to use so-called *critical sub-systems* [6, 9]. The key idea of this approach is to obtain a—preferably small—connected fragment of the Markov chain that itself already violates the property at hand. This sub-system can thus be viewed as another representation of the set of runs that all satisfy the property $\varphi$ whose probability mass exceeds the required upper bound. In [10, 11] we suggested to obtain *minimal* critical sub-systems. Here, minimality refers to the state space size of the sub-Markov chain. Whereas [6, 9] use heuristic approaches to construct small (but not necessarily minimal) critical sub-systems, [10, 11] advocates the use of mixed integer linear programming (MILP) [12]. The MILP-approach is applicable to $\omega$-regular properties (that include reachability) for both Markov chains and Markov decision processes (MDPs) [11], which are a slightly variant of probabilistic automata.[1] A more detailed overview of probabilistic counterexamples is given in [13].

Despite the algorithmic differences, all counterexample generation algorithms published so far have one thing in common—they are all *state* based. This means that path-based approaches yield paths in the Markov model, whereas the critical sub-system techniques obtain fragments of the Markov model. They do not obtain diagnostic information in terms of the modeling formalism in which these models are described. This seriously hampers the comprehensibility of counterexamples and is a significant obstacle in debugging the system model (description). In addition, as most practical systems consist of various components running concurrently, counterexamples in terms of the underlying (potentially huge) state space are often too large to be used effectively. Although critical sub-systems often are orders of magnitude smaller than the original system, they may still be very large, rendering manual debugging practically impossible.

To overcome these deficiencies, this paper focuses on obtaining *counterexamples at the level of the modeling formalism.* The basic idea is to determine a minimal critical model description that acts as a counterexample—preferably in a fully automated manner. Intuitively speaking, our approach determines the fragments of the model description that are relevant for the violation of the property at hand. Having a human-readable specification language, it seems natural that a user should be pointed to the part of the system description

[1] Obtaining minimal critical sub-systems for MDPs is NP-complete [6], as is solving MILP problems.

which causes the error. This is exactly what our approach for *high-level counterexamples* attempts to accomplish.

We assume Markov models are described using a stochastic version of Alur and Henzinger's reactive modules [14]. This is the modeling formalism adopted by the popular probabilistic model checker `PRISM` [15]. In this setting, a probabilistic automaton [4] model is typically specified as a parallel composition of modules. The behavior of a single module is described using a probabilistic extension [16] of Dijkstra's *guarded command language* [17]. Modules communicate by shared variables or synchronization on common actions. Our approach however is also applicable to other modeling formalisms for probabilistic automata such as the process algebraic approach in [18] by using the similarities between linearised process descriptions and the guarded command language.

This paper considers the problem of determining a—preferably small—set of guarded commands that together induce a critical sub-system. In order to correct the system description, at least one of the returned guarded commands has to be changed. We show how to simplify the commands by removing command branches which are not necessary to obtain a counterexample. We present this as a special case of a method where the number of different transition labels for a probabilistic automaton is minimized. This offers great flexibility in terms of human-readable counterexamples. We show that obtaining a *minimal* command set that acts as a counterexample is NP-complete and advocate the usage of MILP techniques to obtain such a smallest critical label set. Besides the theoretical principles of our technique, we illustrate its practical feasibility by showing the results of applying a prototypical implementation to various examples from the `PRISM` benchmark suite.

*Structure of the paper.* The first section briefly reviews the necessary foundations. Section 3 presents the theoretical framework to obtain minimal counterexamples. Section 3.6 introduces several simplification steps for reactive modules. After presenting some experimental results in Section 4, we conclude the paper in Section 5.

This paper is an extended version of [19]. The extensions include (1) a more general labeling (branches instead of transitions) enabling more simplification steps, (2) minimization of variable values and intervals, and (3) detailed correctness proofs for the MILP formulation.

## 2. Foundations

Let $S$ be a countable set. A *sub-distribution* on $S$ is a function $\mu : S \to [0,1] \subseteq \mathbb{Q}$ such that $0 < \sum_{s \in S} \mu(s) \leq 1$ with support $\mathrm{supp}(\mu) = \{s \in S \,|\, \mu(s) > 0\}$. We use the notation $\mu(S') = \sum_{s \in S'} \mu(s)$ for $S' \subseteq S$. A sub-distribution with $\mu(S) = 1$ is called a *probability distribution*. We denote the set of all probability distributions on $S$ by $\mathrm{Distr}(S)$ and analogously the set of sub-distributions by $\mathrm{SubDistr}(S)$.

### 2.1. Probabilistic Automata.

**Definition 2.1** (Probabilistic automaton)**.** A *probabilistic automaton (PA)* is a tuple $\mathcal{M} = (S, s_{\text{init}}, \mathrm{Act}, P)$ such that $S$ is a finite set of states, $s_{\text{init}} \in S$ is an initial state, Act is a finite set of actions, and $P : S \to (2^{\mathrm{Act} \times \mathrm{Distr}(S)} \setminus \{\emptyset\})$ is a probabilistic transition relation such that $P(s)$ is finite for all $s \in S$.

For notational convenience, we use all notations defined for (sub-)distributions also for action-distribution pairs with the natural meaning, e. g., $\mathrm{supp}(\eta) = \mathrm{supp}(\mu)$ and $\eta(s) = \mu(s)$ for $\eta = (\alpha, \mu) \in \mathrm{Act} \times \mathrm{Distr}(S)$. We call $\eta \in P(s)$ a *transition*, while a tuple $(s, \eta, s')$ with $\eta \in P(s)$ and $\eta(s') > 0$ is called a *branch* of the transition.

For a state $s \in S$, a successor state is determined as follows: A transition $\eta \in P(s)$ is chosen non-deterministically. Then, $s' \in \mathrm{supp}(\eta)$ is determined probabilistically according to the distribution in $\eta$. This process can be repeated infinitely often starting with the initial state $s_{\mathrm{init}}$. To prevent deadlocks we assume $P(s) \neq \emptyset$ for all $s \in S$.

An *infinite path* of a PA $\mathcal{M}$ is an infinite sequence $s_0\eta_0 s_1\eta_1 s_2 \ldots$ with $s_i \in S$, $\eta_i \in P(s_i)$ and $s_{i+1} \in \mathrm{supp}(\eta_i)$ for all $i \geq 0$. A *finite path* $\pi$ of $\mathcal{M}$ is a finite prefix $s_0\eta_0 s_1\eta_1 \ldots s_n$ of an infinite path of $\mathcal{M}$ with $\mathrm{last}(\pi) = s_n$. The set of all finite paths of $\mathcal{M}$ is $\mathrm{Paths}^{\mathrm{fin}}_{\mathcal{M}}$.

A *sub-PA* is like a PA, but it allows sub-distributions instead of distributions in the definition of $P$.

**Definition 2.2** (Sub-PA)**.** A *sub-PA* is a tuple $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ with $S$, $s_{\mathrm{init}}$, and Act as in Definition 2.1 and $P : S \to 2^{\mathrm{Act} \times \mathrm{SubDistr}(S)}$ is a probabilistic transition relation such that $P(s)$ is finite for all $s \in S$.

A sub-PA $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ can be transformed into a PA as follows: We add a new state $s_\perp \notin S$ and a new action $\tau \notin \mathrm{Act}$, extend all sub-distributions into probability distributions by defining $\mu(s_\perp) = 1 - \mu(S)$ for each $s \in S$ and $(\alpha, \mu) \in P(s)$, and set $P(s) = \{(\tau, \mu) \in \{\tau\} \times \mathrm{Distr}(S \cup \{s_\perp\}) \mid \mu(s_\perp) = 1\}$ for each $s \in \{s_\perp\} \cup \{s' \in S \mid P(s') = \emptyset\}$. This allows for applying all methods we use for PAs also for sub-PAs.

**Definition 2.3** (Subsystem)**.** A sub-PA $\mathcal{M}' = (S', s'_{\mathrm{init}}, \mathrm{Act}', P')$ is a *subsystem* of a sub-PA $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$, written $\mathcal{M}' \sqsubseteq \mathcal{M}$, iff $S' \subseteq S$, $s'_{\mathrm{init}} = s_{\mathrm{init}} \in S'$, $\mathrm{Act}' \subseteq \mathrm{Act}$ and for all $s \in S'$ there is an injective function $f : P'(s) \to P(s)$ such that for all $(\alpha', \mu') \in P'(s)$ with $f((\alpha', \mu')) = (\alpha, \mu)$ we have that $\alpha' = \alpha$ and for all $s' \in S'$ either $\mu'(s') = 0$ or $\mu'(s') = \mu(s')$.

In this paper we are interested in *probabilistic reachability properties*: Is the probability to reach a set $T \subseteq S$ of target states from $s_{\mathrm{init}}$ at most a given bound $\lambda \in [0, 1] \subseteq \mathbb{R}$? This property is denoted by $\mathcal{P}_{\leq\lambda}(\Diamond\, T)$. Note that checking arbitrary $\omega$-regular properties can be reduced to checking reachability properties, see [20, Chapter 10.3]. To define a suitable probability measure on PAs, the nondeterminism has to be resolved. This is done by an oracle called *scheduler*.

**Definition 2.4** (Scheduler)**.** A memoryless deterministic *scheduler*[2] for a sub-PA $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ is a partial function $\sigma : S \nrightarrow \mathrm{Act} \times \mathrm{SubDistr}(S)$ with $\sigma(s) \in P(s)$ for all $s \in \mathrm{dom}(\sigma)$. We use $\mathrm{Sched}_{\mathcal{M}}$ to denote the set of all memoryless deterministic schedulers of $\mathcal{M}$. $\mathrm{Sched}^{+}_{\mathcal{M}} \subseteq \mathrm{Sched}_{\mathcal{M}}$ is the set of all schedulers that are total functions $\sigma : S \to \mathrm{Act} \times \mathrm{SubDistr}(S)$. Such schedulers are called *deadlock-free*.

As a scheduler resolves the nondeterminism for a PA, this induces a fully probabilistic model, for which a standard probability measure can be defined. We refer to [20, Chapter 10.1] for more details on schedulers and measure theory.

**Definition 2.5** (Sub-PA induced by scheduler)**.** For a sub-PA $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ and a scheduler $\sigma \in \mathrm{Sched}_{\mathcal{M}}$, the *sub-PA induced by* $\mathcal{M}$ *and* $\sigma$ is given by $\mathcal{M}^\sigma = (S^\sigma, s_{\mathrm{init}}, \mathrm{Act}^\sigma, P^\sigma)$

[2]Note that schedulers in their full generality are functions mapping finite paths of the PA to distributions over the outgoing transitions of each state. For unbounded probabilistic reachability properties memoryless deterministic schedulers suffice to obtain maximal (and minimal) probabilities [20, Lemma 10.102].

with $S^\sigma = S$, $\mathrm{Act}^\sigma = \{\alpha \in \mathrm{Act} \mid \exists s \in S.\, \exists \mu \in \mathrm{SubDistr}(S).\, \sigma(s) = (\alpha, \mu)\}$, and $P^\sigma(s) = \{\sigma(s)\}$ for all $s \in \mathrm{dom}(\sigma)$ and $P^\sigma(s) = \emptyset$ for all $s \in S \setminus \mathrm{dom}(\sigma)$.

Note that in $\mathcal{M}^\sigma$, $|P^\sigma(s)| \leq 1$ holds for all $s \in S$. In fact, the actions could be removed, which would yield a *discrete-time Markov chain*. For details, we again refer to [20].

For a sub-PA $\mathcal{M}$ with a fixed scheduler $\sigma$, the probability $\Pr_{\mathcal{M}^\sigma}(s_{\mathrm{init}}, \Diamond T)$ can now be computed by solving a linear equation system. The property $\mathcal{P}_{\leq\lambda}(\Diamond T)$ is satisfied by a sub-PA $\mathcal{M}$ if $\Pr_{\mathcal{M}^\sigma}(s_{\mathrm{init}}, \Diamond T) \leq \lambda$ holds for all schedulers $\sigma$ for $\mathcal{M}$. To check this, it suffices to compute the maximal probability to reach $T$ from $s_{\mathrm{init}}$ over all schedulers, which we denote by $\Pr^+_{\mathcal{M}}(s_{\mathrm{init}}, \Diamond T)$. This probability is given by the unique solution of the following equation system:

$$\Pr^+_{\mathcal{M}}(s, \Diamond T) = \begin{cases} 1 & \text{if } s \in T, \\ 0 & \text{if } T \text{ is unreachable from } s \text{ under all schedulers}, \\ \max\limits_{\eta \in P(s)} \sum\limits_{s' \in S} \eta(s') \cdot \Pr^+_{\mathcal{M}}(s', \Diamond T) & \text{otherwise.} \end{cases} \tag{2.1}$$

It can be solved by either rewriting it into a linear program, by applying a technique called value iteration, or by iterating over the possible schedulers (policy iteration) (see, e. g., [20, Chapter 10.6]). A memoryless deterministic scheduler can be obtained easily from the solution of the equation system (cf. [20, Lemma 10.102]).

2.2. **PRISM's Guarded Command Language.** For a set Var of bounded integer variables, let $\mathcal{A}_{\mathrm{Var}}$ denote the set of all variable assignments, i. e., of functions $\nu : \mathrm{Var} \to \mathbb{Z}$ such that $\nu(\xi) \in \mathrm{dom}(\xi)$ for all $\xi \in \mathrm{Var}$. We assume that the domains of all variables are finite.

**Definition 2.6** (Model, module, command)**.** A *model* is a tuple $(\mathrm{Var}, s_{\mathrm{init}}, \{M_1, \ldots, M_k\})$ where Var is a finite set of Boolean variables, $s_{\mathrm{init}} \in \mathcal{A}_{\mathrm{Var}}$ an initial assignment, and $\{M_1, \ldots, M_k\}$ a finite set of modules.

A *module* is a tuple $M_i = (\mathrm{Var}_i, \mathrm{Act}_i, C_i)$ with $\mathrm{Var}_i \subseteq \mathrm{Var}$ a set of variables such that $\mathrm{Var}_i \cap \mathrm{Var}_j = \emptyset$ for $i \neq j$, $\mathrm{Act}_i$ a finite set of synchronizing actions, and $C_i$ a finite set of commands. The action $\tau$ with $\tau \notin \bigcup_{i=1}^k \mathrm{Act}_i$ denotes the internal non-synchronizing action. A *command* $c \in C_i$ has the form

$$c \;=\; [\alpha]\; g \to\; p_1 : f_1 + \ldots + p_n : f_n$$

with $\alpha \in \mathrm{Act}_i \dot\cup \{\tau\}$, $g$ a Boolean predicate ("guard") over the variables in Var, $p_j \in [0,1]$ a rational number with $\sum_{j=1}^n p_j = 1$, and $f_j : \mathcal{A}_{\mathrm{Var}} \to \mathcal{A}_{\mathrm{Var}_i}$ being a variable update function. We refer to the action $\alpha$ of command $c$ by $\mathrm{act}(c)$.

Note that each module may only change the values of its own variables while their new values may depend on variables of other modules. Each model with several modules is equivalent to a model with a single module, which is obtained by computing the parallel composition of these modules. We give a short intuition on how this composition is built. For more details we refer to the documentation of PRISM.

Assume two modules $M_1 = (\mathrm{Var}_1, \mathrm{Act}_1, C_1)$ and $M_2 = (\mathrm{Var}_2, \mathrm{Act}_2, C_2)$ with $\mathrm{Var}_1 \cap \mathrm{Var}_2 = \emptyset$. We first define the composition $c \otimes c'$ of two commands $c$ and $c'$: For $c = [\alpha]\; g \to p_1 : f_1 + \ldots + p_n : f_n \in C_1$ and $c' = [\alpha]\; g' \to\; p'_1 : f'_1 + \ldots + p'_m : f'_m \in C_2$ we have:

$$c \otimes c' \;=\; [\alpha]\; g \wedge g' \quad\to\quad \sum_{i=1}^{n} \sum_{j=1}^{m} p_i \cdot p'_j : f_i \otimes f'_j \;.$$

Here, for $f_i : \mathcal{A}_{\mathrm{Var}} \to \mathcal{A}_{\mathrm{Var}_1}$ and $f'_j : \mathcal{A}_{\mathrm{Var}} \to \mathcal{A}_{\mathrm{Var}_2}$ we define $f_i \otimes f'_j : \mathcal{A}_{\mathrm{Var}} \to \mathcal{A}_{\mathrm{Var}_1 \cup \mathrm{Var}_2}$ such that for all $\nu \in \mathcal{A}_{\mathrm{Var}}$ we have that $(f_i \otimes f'_j)(\nu)(\xi)$ equals $f_i(\nu)(\xi)$ for each $\xi \in \mathrm{Var}_1$ and $f'_j(\nu)(\xi)$ for each $\xi \in \mathrm{Var}_2$.

Using this, the *parallel composition* $M = M_1 \parallel M_2 = (\mathrm{Var}, \mathrm{Act}, C)$ is given by $\mathrm{Var} = \mathrm{Var}_1 \cup \mathrm{Var}_2$, $\mathrm{Act} = \mathrm{Act}_1 \cup \mathrm{Act}_2$, and

$$\begin{aligned} C \;=\; \{ \quad c \quad & \mid c \in C_1 \cup C_2 \wedge \mathrm{act}(c) \in \{\tau\} \cup (\mathrm{Act}_1 \setminus \mathrm{Act}_2) \cup (\mathrm{Act}_2 \setminus \mathrm{Act}_1) \quad \} \cup \\ \{ \quad c \otimes c' & \mid c \in C_1 \wedge c' \in C_2 \wedge \mathrm{act}(c) = \mathrm{act}(c') \in \mathrm{Act}_1 \cap \mathrm{Act}_2 \quad \} \; . \end{aligned}$$

Intuitively, commands labeled with non-synchronizing actions are executed on their own, while for synchronizing actions a command from each synchronizing module is executed simultaneously. Note that if a module has an action in its synchronizing action set but no commands labeled with this action, this module will block the execution of commands with this action in the composition. This is considered to be a modeling error and the corresponding commands are ignored.

The PA-semantics of a model is as follows. Assume a model $(\mathrm{Var}, s_{\mathrm{init}}, \{M\})$ with a single module $M = (\mathrm{Var}, \mathrm{Act}, C)$ which will not be subject to parallel composition any more and $\mathrm{Var} = \{\xi_1, \ldots, \xi_m\}$. The *state space* $S$ of the corresponding PA $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ is given by the set of all possible variable assignments $\mathcal{A}_{\mathrm{Var}}$, i. e., a state $s$ is a vector $(v_1, \ldots, v_m)$ with $v_i$ being a value of the variable $\xi_i \in \mathrm{Var}$. To construct the transitions, we observe that the guard $g$ of each command

$$c \;=\; [\alpha]\; g \to \;\; p_1 : f_1 + \ldots + p_n : f_n \;\; \in C$$

defines a subset of the state space $S_c \subseteq \mathcal{A}_{\mathrm{Var}}$ with $s \in S_c$ iff $s$ satisfies $g$. For each state $s \in S_c$ we define a probability distribution $\mu_{c,s} : \mathcal{A}_{\mathrm{Var}} \to [0, 1]$ with

$$\mu_{c,s}(s') = \sum_{\{1 \le i \le n \mid f_i(s) = s'\}} p_i$$

for each $s' \in \mathcal{A}_{\mathrm{Var}}$. The probabilistic transition relation $P : \mathcal{A}_{\mathrm{Var}} \to 2^{\mathrm{Act} \times \mathrm{Distr}(\mathcal{A}_{\mathrm{Var}})}$ is given by $P(s) = \{(\alpha, \mu_{c,s}) \mid c \in C \wedge \mathrm{act}(c) = \alpha \wedge s \in S_c\}$ for all $s \in \mathcal{A}_{\mathrm{Var}}$.

**Example 2.7.** We consider the shared coin protocol of a randomized consensus algorithm [21]. The protocol returns a preference between two choices with a certain probability. A shared integer variable[3] `c` is incremented or decremented by each process depending on the internal result of a coin flipping. If the value of `c` becomes lower than a threshold `left` or higher than a threshold `right`, the result is `heads` or `tails`, respectively.

The protocol, which is the same for each participating process, has the following local variables: `coin` which is either 0 or 1, `flip` which is `true` iff the coin shall be flipped, `flipped` which is `true` iff the coin has already been flipped, `check` which is `true` iff the value of `c` shall be checked. Initially, `c` has a value between `left` and `right`, `flip` is true, and `flipped` and `check` are false. Consider a simplified version of the original `PRISM` code:

[3]Our simplified definition of a model does not support variables which are written by more than one module. The language actually implemented by PRISM, however, allows such variables with the restriction that they may be written only by non-synchronizing commands in order to avoid writing conflicts.

$$
\begin{aligned}
&[\tau]\ \texttt{flip} && \to 0.5 : \texttt{coin=0} \,\&\, \texttt{flip=false} \,\&\, \texttt{flipped=true} \\
&&& \quad + 0.5 : \texttt{coin=1} \,\&\, \texttt{flip=false} \,\&\, \texttt{flipped=true} && (2.2)\\
&[\tau]\ \texttt{flipped} \,\&\, \texttt{coin=0} \,\&\, \texttt{c}{\leq}\texttt{right} && \to 1 : \ \texttt{c=c-1} \,\&\, \texttt{flipped=false} \,\&\, \texttt{check=true} && (2.3)\\
&[\tau]\ \texttt{flipped} \,\&\, \texttt{coin=1} \,\&\, \texttt{left}{\leq}\texttt{c} && \to 1 : \ \texttt{c=c+1} \,\&\, \texttt{flipped=false} \,\&\, \texttt{check=true} && (2.4)\\
&[\tau]\ \texttt{c<left} && \to 1 : \ \texttt{heads=true} && (2.5)\\
&[\tau]\ \texttt{c>right} && \to 1 : \ \texttt{tails=true} && (2.6)\\
&[\tau]\ \texttt{check} \,\&\, \texttt{c}{\leq}\texttt{right} \,\&\, \texttt{c}{\geq}\texttt{left} && \to 1 : \ \texttt{check=false} \,\&\, \texttt{flip=true} && (2.7)
\end{aligned}
$$

Command 2.2 sets `coin` to 0 or 1, each with probability 0.5. Commands 2.3 and 2.4 increment or decrement the shared counter `c` depending on the value of `coin`. Commands 2.5 and 2.6 check whether the value of `c` is above or below the boundaries `left` and `right` and return `heads` or `tails`, respectively. If no boundary is violated, Command 2.7 sets `flip` to `true` which enables Command 2.2 again.

2.3. **Mixed Integer Programming.** A *mixed integer linear program* optimizes a linear objective function under a condition specified by a conjunction of linear inequalities. A subset of the variables in the inequalities is restricted to take only integer values, which makes solving MILPs NP-hard [22, Problem MP1].

**Definition 2.8** (Mixed integer linear program)**.** Let $A \in \mathbb{Q}^{m\times n}$, $B \in \mathbb{Q}^{m\times k}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $d \in \mathbb{Q}^k$. A *mixed integer linear program* (MILP) consists in computing $\min c^T x + d^T y$ such that $Ax + By \leq b$ and $x \in \mathbb{R}^n,\ y \in \mathbb{Z}^k$.

MILPs are typically solved by a combination of a branch-and-bound algorithm and the generation of so-called cutting planes. These algorithms heavily rely on the fact that relaxations of MILPs which result from removing the integrality constraints can be efficiently solved. MILPs are widely used in operations research, hardware-software co-design, and numerous other applications. Efficient open source as well as commercial implementations are available like `Scip` [23], and `Gurobi` [24]. We refer to the textbook [12] for more information on solving MILPs.

## 3. Computing Counterexamples

In this section we show how to compute a smallest critical command set of a given model, i. e.,, a smallest subset of the model's commands which lead to an erroneous system independent of the other commands. For this, we introduce a generalization of this problem, namely smallest critical label sets, state the complexity, and specify an MILP formulation to solve this problem.

3.1. **Smallest Critical Label Sets.** Let $\mathcal{M} = (S, s_{\text{init}}, \text{Act}, P)$ be a PA, $T \subseteq S$, and Lab a finite set of labels. Assume furthermore a partial labeling function $L : S \times \text{Act} \times \text{Distr}(S) \times S \nrightarrow 2^{\text{Lab}}$ such that $L(s, \eta, s')$ is defined iff $\eta \in P(s)$ and $s' \in \text{supp}(\eta)$.

Let $\text{Lab}' \subseteq \text{Lab}$ be a subset of the labels. The *sub-PA induced by* $\text{Lab}'$ is $\mathcal{M}_{|\text{Lab}'} = (S, s_{\text{init}}, \text{Act}, P_{|\text{Lab}'})$ such that for all $s \in S$ we have

$$P_{|\text{Lab}'}(s) \;=\; \{(\alpha, \mu_{|\text{Lab}'}) \mid (\alpha, \mu) \in P(s) \wedge \exists s' \in S.\, L(s, \alpha, \mu, s') \subseteq \text{Lab}'\},$$

where $\mu_{|\text{Lab}'} \in \text{SubDistr}(S)$ with $\mu_{|\text{Lab}'}(s') = \mu(s')$ if $L(s, \alpha, \mu, s') \subseteq \text{Lab}'$ and $\mu_{|\text{Lab}'}(s') = 0$ otherwise for each $s' \in S$. Thus in $\mathcal{M}_{|\text{Lab}'}$ all branches have been removed whose labeling is not a subset of $\text{Lab}'$.

**Definition 3.1** (Smallest critical label set (SCL) problem)**.** Let $\mathcal{M}$, $T$, Lab, and $L$ be defined as above and $\mathcal{P}_{\leq\lambda}(\Diamond T)$ be a reachability property that is violated by $s_{\text{init}}$ in $\mathcal{M}$. A label set $\text{Lab}' \subseteq \text{Lab}$ and its induced sub-PA $\mathcal{M}_{|\text{Lab}'} = (S, s_{\text{init}}, \text{Act}, P')$ are called *critical* if $\Pr^{+}_{\mathcal{M}_{|\text{Lab}'}}(s_{\text{init}}, \Diamond T) > \lambda$.

Given a weight function $w : \text{Lab} \to \mathbb{R}^{\geq 0}$, the *smallest critical label set (SCL) problem* is to determine a critical subset $\text{Lab}' \subseteq \text{Lab}$ such that $w(\text{Lab}') = \sum_{\ell \in \text{Lab}'} w(\ell)$ is minimal among all critical subsets of Lab.

**Theorem 3.2.** *To decide whether there is a critical label set* $\text{Lab}' \subseteq \text{Lab}$ *with* $w(\text{Lab}') \leq k$ *for a given integer* $k \geq 0$ *is NP-complete.*

The proof of this theorem is a reduction from exact 3-cover (X3C) [22], similar to a proof in [25]. For the aid of the reviewers, we give the proof in Appendix A.

The concept of smallest critical label sets gives us a flexible description of counterexamples being minimal with respect to different quantities. We will now list different kinds of counterexamples that can be computed using an SCL.

**Commands.** In order to minimize the number of commands that together induce an erroneous system, i. e., form a *critical command set*, let $\mathcal{M} = (S, s_{\text{init}}, \text{Act}, P)$ be a PA generated by modules $M_i = (\text{Var}_i, \text{Act}_i, C_i)$, $i = 1, \ldots, k$. For each module $M_i$ and each command $c \in C_i$ we introduce a unique label[4] $\ell_{c,i}$ with weight 1 and define the labeling function $L : S \times \text{Act} \times \text{Distr}(S) \times S \nrightarrow 2^{\text{Lab}}$ such that the labels in $L(s, \eta, s')$ correspond to the set of commands which together generate this transition $\eta \in P(s)$.[5] Note that in case of synchronization several commands together create a certain transition. An SCL then corresponds to a *smallest critical command set*.

**Modules.** We can also minimize the number of modules involved in a counterexample by using the same label for all commands in a module. Often systems consist of a number of copies of the same module, containing the same commands, only with the local variables renamed, plus a few extra modules. Consider for example a wireless network: $n$ nodes want to transmit messages using a protocol for medium access control [26]. All nodes run the same protocol. Additionally, there may be a module describing the channel. When fixing an erroneous system, one wants to preserve the identical structure of the nodes. Therefore the selected commands should contain the same subset of commands from all identical modules.

[4]In the following we write short $\ell_c$ instead of $\ell_{c,i}$ if the index $i$ is clear from the context.

[5]If several command sets generate the same transition, we introduce copies of the transition.

This can be obtained by assigning the same label to all corresponding commands from the symmetric modules and using the number of symmetric modules as its weight.

**Deletion of unnecessary branches.** The SCL problem can also be used to simplify commands. For this we identify a smallest set of command branches that need to be preserved, such that the induced sub-PA still violates the property under consideration. The resulting command branches can be removed, still yielding an erroneous system. Given a command $c_i$ of the form $[\alpha]\ g \to p_1 : f_1 + p_2 : f_2 + \cdots + p_n : f_n$, we assign to each command branch $p_j : f_j$ a unique label $\ell_{i,j}$ with weight 1. Let Lab be the set of all such labels. When the parallel composition of modules is computed, see Section 2.2, we build the union of the labelings of the synchronizing command branches being executed together. When computing the corresponding PA $\mathcal{M}$, we transfer this labeling to the transition branches of $\mathcal{M}$: We define the labeling function $L$ such that $L(s, \eta, s')$ contains the labels of all command branches that are involved in generating the branch from $s$ to $s'$ via the transition $\eta$.

**States.** The state-minimal critical subsystems as introduced in [10] can be also obtained as special case of smallest critical label sets: For each state $s \in S$ introduce a label $\ell_s$ with weight 1 and set $L(s, \eta, s') = \{\ell_{s'}\}$ for all $s \in S$, $\eta \in P(s)$ and $s' \in \mathrm{supp}(\eta)$. Then a smallest critical label set $\mathrm{Lab}' \subseteq \mathrm{Lab} = \{\ell_s \mid s \in S\}$ induces a state-minimal critical subsystem.

**Variable domains.** Smallest critical label sets can also be used to reduce the domains of the variables in the `PRISM` program. Let Var be the set of variables of a `PRISM` program and $\mathcal{M} = (S, s_{\mathrm{init}}, \mathrm{Act}, P)$ the corresponding PA. Note that each state $s \in S$ corresponds to an assignment of the variables in Var. For a variable $\xi \in \mathrm{Var}$ we denote by $s(\xi)$ the value of $\xi$ in state $s$. Let $\mathrm{Lab} = \{\ell_{\xi,v} \mid \xi \in \mathrm{Var} \wedge v \in \mathrm{dom}(\xi)\}$ be the set of labels, each with weight 1. We define the labeling of transition branches by corresponding variable values as $L(s, \eta, s') = \{\ell_{\xi,v} \mid \xi \in \mathrm{Var} \wedge s'(\xi) = v\}$. A smallest critical labeling induces a critical subsystem with a minimum number of variable values. If we restrict the variable domains to these values, we still obtain an erroneous system.

**Variable intervals.** The previous reduction technique removes a maximum number of values from the variables' domains. Originally the domains are intervals in $\mathbb{Z}$. Minimization, however, yields sets that are in general not intervals anymore. We can also minimize the size of the intervals instead. To do so we need to impose further constraints on the valid label sets Lab$'$. Details will be presented in Section 3.7.

**Remark 3.3.** The various applications require label sets of very different sizes. For the minimization of commands, modules, and branches the number of labels is linear in the size of the model description. In contrast, the number of different labels for state minimization is linear in the number of (reachable) states of the described PA, which can be exponential in the size of its description. The same holds for the minimization of variable domains and intervals.

3.2. **Computing Smallest Critical Label Sets.** We now explain how smallest critical label sets can be computed. First, the notions of *relevant* and *problematic* states are introduced. Intuitively, state $s$ is relevant if it is on a path from the initial state to a target state. A relevant state $s$ is problematic, if there exists a deadlock-free scheduler under which no target state is reachable from $s$.

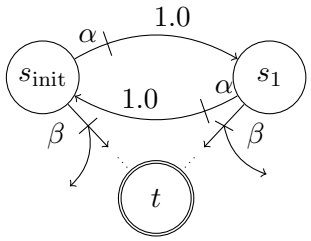


Figure 1: Example for problematic states

**Definition 3.4** (Relevant and problematic states and transitions)**.** Let $\mathcal{M}$, $T$, and $L$ be as above. The *relevant states of* $\mathcal{M}$ *for* $T$ are given by $S_T^{\text{rel}} = \{s \in S \mid \exists \sigma \in \text{Sched}_{\mathcal{M}}.\ \Pr^{\sigma}_{\mathcal{M}}(s_{\text{init}}, \Diamond\{s\}) > 0 \wedge \Pr^{\sigma}_{\mathcal{M}}(s, \Diamond T) > 0\}$. A label $\ell$ is *relevant* for $T$ if there are $s \in S_T^{\text{rel}}$, $\eta \in P(s)$, and $s' \in \text{supp}(\eta) \cap S_T^{\text{rel}}$ such that $\ell \in L(s, \eta, s')$.

The states in $S_T^{\text{prob}} = \{s \in S_T^{\text{rel}} \mid \exists \sigma \in \text{Sched}^{+}_{\mathcal{M}}.\ \Pr^{\sigma}_{\mathcal{M}}(s, \Diamond T) = 0\}$ are *problematic states* and the set $P_T^{\text{prob}} = \left\{(s,\eta) \in S_T^{\text{prob}} \times \text{Act} \times \text{Distr}(S) \,\middle|\, \eta \in P(s) \wedge \text{supp}(\eta) \subseteq S_T^{\text{prob}}\right\}$ are *problematic state-transition-pairs* regarding $T$.

States that are not relevant can be removed from the PA together with all their incident branches without changing the probability of reaching $T$ from $s_{\text{init}}$. Additionally, all labels that do not occur in the relevant part of the PA can be deleted. We therefore assume that the (sub-)PA under consideration contains only states and labels that are relevant for $T$. Note that the relevant states and labels can be computed in linear time using graph algorithms [27]. In our computation, we need to ensure that from each problematic state a non-problematic state is reachable under the selected scheduler, otherwise the probability of problematic states is not well-defined by the constraints as in [28].

**Example 3.5.** The PA in Figure 1 illustrates the issues with problematic states. Assume $t$ is a target state. States $s_{\text{init}}$ and $s_1$ are both problematic since the scheduler which selects $\alpha$ in both $s_{\text{init}}$ and $s_1$ prevents reaching a target state, but all other schedulers do not. We cannot remove the outgoing transitions belonging to action $\alpha$ in a preprocessing step since a scheduler may choose $\alpha$ in one state and $\beta$ in the other one. However, if a scheduler chooses $\alpha$ in both states, we obtain according to Equation (2.1) the following equation system for model checking:

$$p_{s_{\text{init}}} = 1.0 \cdot p_{s_1} \qquad p_{s_1} = 1.0 \cdot p_{s_{\text{init}}}$$

For all $\kappa \in [0,1]$ we obtain a solution by setting $p_{s_{\text{init}}} = p_{s_1} = \kappa$, although the target state $t$ is not reachable at all.

We solve this problem by attaching a value $r_s$ to each problematic state $s \in S_T^{\text{prob}}$ and encoding that a transition of $s$ may be selected only if it has at least one successor state $s'$ with a value $r_{s'} > r_s$ attached to it. Since the sub-PA is finite, this requirement assures by induction that there is a (loop-free) *increasing path* from $s$ to a non-problematic or deadlock state, along which the values attached to the states are strictly increasing. This implies that the probability of each loop visiting problematic states only is always less than one.

To encode the computation of smallest critical label sets as MILP, we use the following variables with values assigned as described:

- for each $\ell \in \text{Lab}$ a variable $x_\ell \in \{0,1\} \subseteq \mathbb{Z}$ which is 1 iff $\ell$ is part of the critical label set;

- for each state $s \in S \setminus T$ and each transition $\eta \in P(s)$ a variable $\sigma_{s,\eta} \in \{0,1\} \subseteq \mathbb{Z}$ which is 1 iff $\eta$ is chosen in $s$ by the scheduler; the scheduler is free not to choose any transition;
- for each branch $(s,\eta,s')$ with $s \in S$, $\eta \in P(s)$ and $s' \in \operatorname{supp}(\eta)$ a variable $p_{s,\eta,s'} \in [0,1] \subseteq \mathbb{R}$ which is 0 if not all labels in $L(s,\eta,s')$ are contained in $\mathrm{Lab}'$, and at most $\eta(s')$ otherwise;
- for each state $s \in S$ a variable $p_s \in [0,1] \subseteq \mathbb{R}$ whose value is at most the probability to reach a target state from $s$ under the selected scheduler within the subsystem induced by the selected label set;
- for each problematic state $s \in S$ a variable $r_s \in [0,1] \subseteq \mathbb{R}$ for the encoding of increasing paths; and
- for each problematic state-transition pair $(s,\eta) \in P_T^{\mathrm{prob}}$ and each successor state $s' \in \operatorname{supp}(\eta)$ a variable $t_{s,\eta,s'} \in \{0,1\} \subseteq \mathbb{Z}$, where $t_{s,\eta,s'} = 1$ implies that the values attached to the states increase along the branch $(s,\eta,s')$, i. e., $r_s < r_{s'}$.

Let $w_{\min} = \min\{w(\ell) \mid \ell \in \mathrm{Lab} \wedge w(\ell) > 0\}$ be the smallest positive weight that is assigned to any label. The MILP for the smallest critical label set problem is then as follows:

$$\text{minimize} \quad -\frac{1}{2} w_{\min} \cdot p_{s_{\mathrm{init}}} + \sum_{\ell \in \mathrm{Lab}} w(\ell) \cdot x_\ell \tag{3.1a}$$

such that

$$p_{s_{\mathrm{init}}} > \lambda \tag{3.1b}$$

$$\forall s \in T. \quad p_s = 1 \tag{3.1c}$$

$$\forall s \in S \setminus T. \quad \sum_{\eta \in P(s)} \sigma_{s,\eta} \leq 1 \tag{3.1d}$$

$$\forall s \in S \setminus T. \quad p_s \leq \sum_{\eta \in P(s)} \sigma_{s,\eta} \tag{3.1e}$$

$$\forall s \in S \setminus T. \forall \eta \in P(s). \forall s' \in \operatorname{supp}(\eta). \forall \ell \in L(s,\eta,s'). \quad p_{s,\eta,s'} \leq x_\ell \tag{3.1f}$$

$$\forall s \in S \setminus T. \forall \eta \in P(s). \forall s' \in \operatorname{supp}(\eta). \quad p_{s,\eta,s'} \leq \eta(s') \cdot p_{s'} \tag{3.1g}$$

$$\forall s \in S \setminus T. \forall \eta \in P(s). \quad p_s \leq (1 - \sigma_{s,\eta}) + \sum_{s' \in \operatorname{supp}(\eta)} p_{s,\eta,s'} \tag{3.1h}$$

$$\forall (s,\eta) \in P_T^{\mathrm{prob}}. \quad \sigma_{s,\eta} = \sum_{s' \in \operatorname{supp}(\eta)} t_{s,\eta,s'} \tag{3.1i}$$

$$\forall (s,\eta) \in P_T^{\mathrm{prob}}. \forall s' \in \operatorname{supp}(\eta). \quad r_s < r_{s'} + (1 - t_{s,\eta,s'}) \tag{3.1j}$$

We first explain the constraints in lines (3.1b)–(3.1j) of the MILP, which describe a critical label set. First, we ensure that the probability of the initial state exceeds the probability bound $\lambda$ (3.1b). The probability of target states is set to 1 (3.1c). For each state $s \in S \setminus T$ the scheduler selects at most one transition, encoded by setting at most one scheduler variable $\sigma_{s,\eta} \in P(s)$ to 1 (3.1d). Note that there may be states where no transition is chosen. In this case the probability of a state is set to 0 (3.1e). The next two constraints describe the probability contribution of an edge $\eta \in P(s)$ from $s$ to $s'$: If a label $\ell \in L(s,\eta,s')$ is not contained in the selected subset, the probability of the branch is set to

0 (3.1f). Otherwise this constraint is satisfied for all possible values of $p_{s,\eta,s'} \in [0,1] \subseteq \mathbb{R}$. Then the following constraint (3.1g) imposes an upper bound on the contribution of this branch, namely the probability $\eta(s')$ of this branch times the probability of the successor state $s'$. Constraint (3.1h) is trivially satisfied if $\sigma_{s,\eta} = 0$, i. e., if the scheduler does not select the current transition. Otherwise the probability $p_s$ of state $s$ is at most the sum of the probabilities of its outgoing branches.

The reachability of at least one deadlocking or non-problematic state is ensured by (3.1i) and (3.1j). First, if a problematic transition $\eta$ of a state $s$ is selected by the scheduler then exactly one transition branch flag must be activated. Second, for all paths along activated branches of problematic transitions, an increasing order on the problematic states is enforced. Because of this order, no problematic states can be revisited on an increasing path which enforces the final reachability of a non-problematic or a deadlocking state.

These constraints assure that each satisfying assignment of the label variables $x_\ell$ corresponds to a critical label set. By minimizing the total weight of the selected labels we obtain a smallest critical label set. By the additional term $-\frac{1}{2} w_{\min} \cdot p_{s_{\text{init}}}$ we obtain not only a smallest critical label set but one with maximal probability. The coefficient $-\frac{1}{2} w_{\min}$ is needed to ensure that the benefit from maximizing the probability is smaller than the loss by adding an additional label. Please note that any coefficient $c$ with $0 < c < w_{\min}$ could be used.

3.3. **Size of the MILP.** The number of integer variables in this MILP is in $O(l + m)$, the number of real variables in $O(n + m)$, and the number of constraints in $O(n + l \cdot m)$ where $l = |\text{Lab}|$ is the number of labels, $n = |S|$ the number of states, and $m$ the number of branches of PA $\mathcal{M}$, i. e., $m = \big|\{(s,\eta,s') \mid s \in S, \eta \in P(s), s' \in \text{supp}(\eta)\}\big|$.

**Remark 3.6.** In case the labeling $L(s,\eta,s')$ does not depend on the successor states $s'$, but only on the state $s$ and the selected transition $\eta$ or even only on $s$, then constraints (3.1f)–(3.1h) can be simplified. See [19] for details.

3.4. **Correctness of the MILP.** For the correctness of the MILP formulation (3.1a)–(3.1j) we need to show that for each critical label set there is a satisfying assignment of the MILP and that from each satisfying assignment of the MILP one can construct a critical label set. As setting we have again $\mathcal{M}$, $T$, $L$, Lab and $\mathcal{M}_{|\text{Lab}'}$ for $\text{Lab}' \subseteq \text{Lab}$.

**Lemma 3.7.** *For each critical label set* $\text{Lab}' \subseteq \text{Lab}$ *there is an assignment* $\nu$ *of the MILP variables with* $\nu(x_\ell) = 1$ *iff* $\ell \in \text{Lab}'$ *such that the constraints* (3.1b)–(3.1j) *are satisfied.*

*Proof.* Let $\text{Lab}' \subseteq \text{Lab}$ be a critical label set. Then $\Pr^+_{\mathcal{M}_{|\text{Lab}'}}(s_{\text{init}}, \Diamond T) > \lambda$ and a deterministic memoryless scheduler $\sigma$ for $\mathcal{M}_{|\text{Lab}'}$ exists with $\Pr_{\mathcal{M}^\sigma_{|\text{Lab}'}}(s_{\text{init}}, \Diamond T) = \Pr^+_{\mathcal{M}_{|\text{Lab}'}}(s_{\text{init}}, \Diamond T) > \lambda$ and $s \in \text{dom}(\sigma)$ iff $\Pr^+_{\mathcal{M}_{|\text{Lab}'}}(s, \Diamond T) > 0$, for all $s \in S \setminus T$. Let $G = (V, E)$ be the digraph with $V = \text{dom}(\sigma) \cup T$ and $E = \{(s,s') \in V \times V \mid s' \in \text{supp}(\sigma(s))\}$. Now consider a smallest (edge-minimal) subgraph $G' = (V, E')$ of $G$ containing for each state $s \in V$ a path from $s$ to $T$. Due to minimality, $G'$ is loop-free and contains for each state $s \in V \setminus T$ exactly one

outgoing edge. We set

$$\nu(x_\ell) = \begin{cases} 1 & \text{if } \ell \in \mathrm{Lab}', \\ 0 & \text{otherwise;} \end{cases} \qquad \nu(\sigma_{s,\eta}) = \begin{cases} 1 & \text{if } s \in \mathrm{dom}(\sigma) \text{ and } \sigma(s) = \eta, \\ 0 & \text{otherwise;} \end{cases}$$

$$\nu(p_s) = \Pr_{\mathcal{M}^\sigma_{|\mathrm{Lab}'}}(s, \Diamond T); \qquad \nu(p_{s,\eta,s'}) = \begin{cases} \eta(s') \cdot \nu(p_{s'}) & \text{if } L(s,\eta,s') \subseteq \mathrm{Lab}', \\ 0 & \text{otherwise;} \end{cases}$$

$$\nu(t_{s,\eta,s'}) = \begin{cases} 1 & \text{if } s \in V \cap S^{\mathrm{prob}}_T \text{ and } (s,s') \in E' \text{ and } \sigma(s) = \eta, \\ 0 & \text{otherwise;} \end{cases}$$

$$\nu(r_s) = \begin{cases} \frac{1}{2}\nu(r_{s'}) & \text{if } s \in V \cap S^{\mathrm{prob}}_T \text{ and} (s,s') \in E', \\ 1 & \text{otherwise.} \end{cases}$$

We now systematically check the constraints (3.1b) through (3.1j):

(3.1b) is satisfied by $\nu$ because $\nu(p_{s_{\mathrm{init}}}) = \Pr_{\mathcal{M}^\sigma_{|\mathrm{Lab}'}}(s_{\mathrm{init}}, \Diamond T) > \lambda$.

(3.1c) holds because $\Pr_{\mathcal{M}^\sigma_{|\mathrm{Lab}'}}(s, \Diamond T) = 1$ for all target states $s \in T$.

(3.1d) holds since the deterministic memoryless scheduler $\sigma$ selects at most one transition in each state.

(3.1e) is trivially satisfied if $\nu(\sigma_{s,\eta}) = 1$ for some $\eta \in P(s)$. Otherwise, if no action is chosen, $s$ is a deadlock state and the probability $\nu(p_s)$ to reach a target state is 0.

(3.1f) is satisfied as $\nu(p_{s,\eta,s'})$ is defined to be 0 if $\ell \in L(s,\eta,s')$ for some $\ell \notin \mathrm{Lab}'$.

(3.1g) holds by the definition of $\nu(p_{s,\eta,s'})$.

(3.1h) is trivially satisfied if $\nu(\sigma_{s,\eta}) = 0$. In case $\nu(\sigma_{s,\eta}) = 1$, the constraint reduces to $p_s \leq \sum_{s' \in \mathrm{supp}(\eta)} p_{s,\eta,s'} \leq \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \cdot p_{s'}$ with $\eta = \sigma(s)$. It is satisfied if $\nu(p_s) = 0$. Otherwise, since $\nu(p_s)$ is the reachability probability of $T$ in $\mathcal{M}^\sigma_{|\mathrm{Lab}'}$, it satisfies the following equation [20, Theorem 10.19]:

$$\nu(p_s) = \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \cdot \nu(p_{s'}) = \sum_{s' \in \mathrm{supp}(\eta)} \nu(p_{s,\eta,s'}).$$

$\mathcal{M}_{|\mathrm{Lab}'}$ contains exactly those branches $(s,\eta,s')$ of $\mathcal{M}^\sigma$ for which $L(s,\eta,s') \subseteq \mathrm{Lab}'$ and therefore $\nu(p_{s,\eta,s'}) = \eta(s') \cdot \nu(p_{s'})$. For all other branches $(s,\eta,s')$ in $\mathcal{M}^\sigma$, but not in $\mathcal{M}^\sigma_{|\mathrm{Lab}'}$, $\nu(p_{s,\eta,s'}) = 0$ holds. Hence we have $\nu(p_s) = \sum_{s' \in \mathrm{supp}(\eta)} \nu(p_{s,\eta,s'})$ and (3.1h) is satisfied.

(3.1i) holds if $\nu(\sigma_{s,\eta}) = 0$, since in this case by definition either $s \notin \mathrm{dom}(\sigma)$ or $\eta \neq \sigma(s)$ and therefore $\nu(t_{s,\eta,s'}) = 0$ for all $s' \in S$. Otherwise $\nu(\sigma_{s,\eta}) = 1$, i. e., $\sigma(s) = \eta$. By the construction of $G'$ there is exactly one $s' \in \mathrm{supp}(\eta)$ with $\nu(t_{s,\eta,s'}) = 1$.

(3.1j) is straightforward if $\nu(t_{s,\eta,s'}) = 0$. Otherwise by definition $r_s = \frac{1}{2} r_{s'}$, and since $\nu(r_s), \nu(r_{s'}) > 0$, the inequality holds. □

**Lemma 3.8.** *Let $\nu$ be a satisfying assignment of the MILP* (3.1b)–(3.1j). *Then* $\mathrm{Lab}' = \{\ell \in \mathrm{Lab} \,|\, \nu(x_\ell) = 1\}$ *is a critical label set.*

*Proof.* Let $\nu$ be a satisfying assignment and $\mathrm{Lab}' = \{\ell \in \mathrm{Lab} \,|\, \nu(x_\ell) = 1\}$ the induced label set. We define the scheduler $\sigma : S \nrightarrow \mathrm{Act} \times \mathrm{SubDistr}(S)$ by $\mathrm{dom}(\sigma) = \{s \in S \,|\, \exists \eta \in P(s).\, \nu(\sigma_{s,\eta}) = 1 \wedge \exists s' \in \mathrm{supp}(\eta).\, L(s,\eta,s') \subseteq \mathrm{Lab}'\}$ and for each $s \in \mathrm{dom}(\sigma)$ we set $\sigma(s) = \eta$ with $\nu(\sigma_{s,\eta}) = 1$. Due to constraint (3.1d) there is at most one transition $\eta \in P(s)$ for $\sigma_{s,\eta} = 1$. Therefore the scheduler is well defined. If $s \notin \mathrm{dom}(\sigma)$ then $s$ is a deadlock state under $\sigma$ with no outgoing transition.

Let $\sigma_{|\mathrm{Lab}'} : S \nrightarrow \mathrm{Act} \times \mathrm{SubDistr}(S)$ result from $\sigma$ by removing branches whose labels are not included in $\mathrm{Lab}'$, i. e., $\mathrm{dom}(\sigma_{|\mathrm{Lab}'}) = \mathrm{dom}(\sigma)$ and for each $s \in \mathrm{dom}(\sigma_{|\mathrm{Lab}'})$ and $s' \in S$, $\sigma_{|\mathrm{Lab}'}(s)(s') = \sigma(s)(s')$ if $L(s, \sigma(s), s') \subseteq \mathrm{Lab}'$, and $\sigma_{|\mathrm{Lab}'}(s)(s') = 0$ otherwise.

Let $U$ be the set of states from which $T$ is not reachable in $\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}$[6], $D$ the deadlock states in $U$, and $R$ the states in $U$ whose scheduled transitions got reduced by removing some branches due to the selected label set:

$$\begin{aligned} U &= \{s \in S \mid T \text{ is unreachable from } s \text{ in } \mathcal{M}^{\sigma}_{|\mathrm{Lab}'}\} \\ D &= U \setminus \mathrm{dom}(\sigma_{|\mathrm{Lab}'}) \\ R &= \{s \in U \cap \mathrm{dom}(\sigma_{|\mathrm{Lab}'}) \mid \sigma(s) \neq \sigma_{|\mathrm{Lab}'}(s)\}\,. \end{aligned}$$

The reachability probabilities $q_s = \Pr_{\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}}(s, \Diamond T)$ are the unique solution of the following linear equation system [20, Theorem 10.19]:

$$q_s = \begin{cases} 1 & \text{for } s \in T, \\ 0 & \text{for } s \in U, \\ \sum_{s' \in S} \sigma_{|\mathrm{Lab}'}(s)(s') \cdot q_{s'} & \text{otherwise.} \end{cases} \tag{3.2}$$

This equation system is well defined, since, if $\sigma_{|\mathrm{Lab}'}(s)$ is undefined, either $s$ is a target state or the target states are unreachable from $s$. In the following we prove

$$\nu(p_s) = 1 \qquad \text{for } s \in T, \tag{3.3}$$

$$\nu(p_s) = 0 \qquad \text{for } s \in U, \tag{3.4}$$

$$\nu(p_s) \leq q_s \qquad \text{otherwise}\,. \tag{3.5}$$

Thus $\nu(p_s) \leq q_s$ for each $s \in S$. With (3.1b) we get $q_{s_{\mathrm{init}}} > \lambda$, i. e., $\mathrm{Lab}'$ is critical.

It remains to show that (3.3)–(3.5) hold.

(3.3) is straightforward for target states $s \in T$ due to (3.1c).

(3.4) First we observe that from all states $s \in U$ a state in $D \cup R$ is reachable: Equations (3.1i) and (3.1j) assure that from each problematic state in $U$ we can reach a state from $D \cup R$ (proof by induction over the problematic states $s \in S^{\mathrm{prob}}_T$ with decreasing values $r_s$). From the non-problematic states in $U$ the target states $T$ are reachable in $\mathcal{M}$ under each scheduler. Therefore, the unreachability of $T$ from those states in $\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}$ is due to the selected label set, where certain branches on each path leading to $T$ are not available any more. Thus also from each non-problematic state in $U$ we can reach a state in $D \cup R$.

Now we show that $\nu(p_s) = 0$ for all $s \in U$. Assume the opposite and let $s \in U$ with $\nu(p_s) = \xi^{\max} = \max\{\nu(p_{s'}) \mid s' \in U\} > 0$. Then $s \in \mathrm{dom}(\sigma_{|\mathrm{Lab}'})$ by Equations (3.1e)–(3.1h), and for $\sigma_{|\mathrm{Lab}'}(s) = \eta$ we get:

$$\begin{aligned} \xi^{\max} \;=\; \nu(p_s) \;\leq\; & \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \cdot \nu(p_{s'}) \;\leq\; \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \cdot \xi^{\max} \\ =\; & \xi^{\max} \cdot \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \;\leq\; \xi^{\max}\,. \end{aligned} \tag{3.6}$$

[6]Note that the order of operations is not arbitrary here. We have $(\mathcal{M}^{\sigma})_{|\mathrm{Lab}'} = (\mathcal{M}_{|\mathrm{Lab}'})^{\sigma_{|\mathrm{Lab}'}}$.

Therefore all inequalities have to hold with equality. Since $\xi^{\max}$ is assumed to be positive, this is possible only if $\sum_{s' \in \mathrm{supp}(\eta)} \eta(s') = 1$, i. e., $s \in U \setminus R$, and $\nu(p_{s'}) = \xi^{\max}$ for all $s' \in \mathrm{supp}(\eta)$. By induction we conclude that $\nu(p_{s'}) = \xi^{\max}$ and $s' \in U \setminus R$ for all states $s'$ that are reachable from $s$ under $\sigma_{|\mathrm{Lab}'}$. We know that from each $s \in U$ either a state $s' \in D$ or a state $s' \in R$ is reachable. For the former case $s' \in D$, from (3.1e)–(3.1h) we imply $\nu(p_{s'}) = 0$, contradicting to $\nu(p_{s'}) = \xi^{\max} > 0$. In the latter case $s' \in R$, the definition of $R$ implies $\sum_{s'' \in \mathrm{supp}(\sigma_{|\mathrm{Lab}'}(s'))} \sigma_{|\mathrm{Lab}'}(s')(s'') < 1$, contradicting to $\sum_{s'' \in \mathrm{supp}(\sigma_{|\mathrm{Lab}'}(s'))} \sigma_{|\mathrm{Lab}'}(s')(s'') = 1$. Therefore our assumption was wrong and we have proven $\nu(p_s) = 0$ for each $s \in U$.

(3.5) Finally we show that $\nu(p_s) \leq q_s$ for each $s \in S \setminus (T \cup U)$. The constraints (3.1f)–(3.1h) can be simplified for the chosen action $\sigma_{|\mathrm{Lab}'}(s) = \eta$ to:

$$p_s \leq \sum_{s' \in \mathrm{supp}(\eta)} \eta(s') \cdot p_{s'} \tag{3.7}$$

Let $\nu_{\mathrm{opt}}$ be a satisfying assignment such that $\nu_{\mathrm{opt}}(p_s)$ is maximal among all satisfying assignments (this maximum exists, since the set of satisfying assignments is compact). We claim that for all $s' \in S \setminus (T \cup U)$ reachable from $s$ in $\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}$, Equation (3.7) is satisfied by $\nu_{\mathrm{opt}}$ with equality. Assume the converse is true, i. e., there is a state $s' \in S \setminus (T \cup U)$ that is reachable from $s$ in $\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}$ such that $\sigma_{|\mathrm{Lab}'}(s) = \eta$ and

$$0 < \varepsilon = \left( \sum_{s'' \in \mathrm{supp}(\eta)} \eta(s'') \cdot \nu_{\mathrm{opt}}(p_{s''}) \right) - \nu_{\mathrm{opt}}(p_{s'}) \ .$$

Let $s = s_0 \eta_0 s_1 \eta_1 \dots s_n = s'$ be an acyclic path in $\mathcal{M}^{\sigma}_{|\mathrm{Lab}'}$ from $s$ to $s'$. We could increase the value $\nu_{\mathrm{opt}}(p_{s_n})$ by at least $\varepsilon_n = \varepsilon$ (more, if $p_{s_n}$ also appears on the right-hand side; note that $0 \leq \eta_i(s_i) < 1$ holds for all $i = 0, \dots, n$). This would not violate any inequality, since in the inequalities for the other states $p_{s_n}$ appears only in upper bounds on the right-hand sides with a non-negative coefficient. Assume that, for some $i \leq n$, we have increased the value of $s_i$ by $\varepsilon_i$. Then the right-hand side of the inequality for $s_{i-1}$ increases by at least $\eta_{i-1}(s_i) \cdot \varepsilon_i > 0$. Therefore we could also increase the value of $p_{s_{i-1}}$ by $\eta_{i-1}(s_i) \cdot \varepsilon_i$. This could be continued along the path back to $s = s_0$, whose value could be increased by $\varepsilon_0 = \varepsilon \cdot \prod_{i=0}^{n-1} \eta_i(s_{i+1}) > 0$. But then $\nu_{\mathrm{opt}}(p_s)$ would not be optimal, contradicting our assumption $\varepsilon > 0$.

This means, the inequalities for all states that are reachable from $s$ are satisfied with equality for $\nu_{\mathrm{opt}}$, in other words, $\nu_{\mathrm{opt}}$ encodes the solution $\nu_{\mathrm{opt}}(p_s) = q_s$ to (3.2). Since $\nu_{\mathrm{opt}}$ is maximal for $s$, all other assignments satisfy $\nu(p_s) \leq q_s$. □

**Theorem 3.9.** *The MILP given in* (3.1a)–(3.1j) *yields a smallest critical label set.*

*Proof.* According to Lemmas 3.7 and 3.8, for each critical label set $\mathrm{Lab}'$ there is a satisfying assignment $\nu$ and vice versa. With $\mathrm{Lab}' = \{\ell \in \mathrm{Lab} \mid \nu(x_\ell) = 1\}$ and $w(\mathrm{Lab}') = \sum_{\ell \in \mathrm{Lab}'} w(\ell) = \sum_{\ell \in \mathrm{Lab}} w(\ell) \cdot \nu(x_\ell)$, for the objective function

$$w(\mathrm{Lab}') - w_{\min} < -\frac{1}{2} w_{\min} \cdot \nu(p_{s_{\mathrm{init}}}) + w(\mathrm{Lab}') < w(\mathrm{Lab}')$$

holds. By minimizing the objective function, we obtain a smallest critical label set. □

3.5. **Optimizations.** The constraints of the MILP describe *critical label sets*, whereas *minimality* is enforced by the objective function. In this section we describe how some additional constraints can be imposed, which explicitly exclude variable assignments that are either not optimal or encode label sets that are also encoded by other assignments. The branch & bound methods used for solving MILPs [12] obtain lower bounds on the optimal solution by solving a linear relaxation of the problem. The lower bounds are used to prune branches of the search space which cannot contain an optimal solution. Adding constraints that are redundant regarding the feasible solutions can improve the relaxation, yield larger lower bounds and let the solver thus prune more branches of the search space. This can reduce the computation time significantly in spite of the larger number of constraints.

**Scheduler cuts.** We want to exclude solutions of the constraint set for which a non-deadlocking state $s$ has only *deadlocking successors* under the selected scheduler. Note that such solutions would define $p_s = 0$. We add for all $s \in S \setminus T$ and all $\eta \in P(s)$ with $\mathrm{supp}(\eta) \cap T = \emptyset$ the constraint

$$\sigma_{s,\eta} \;\leq \sum_{s' \in \mathrm{supp}(\eta) \setminus \{s\}} \; \sum_{\eta' \in P(s')} \sigma_{s',\eta'} \;. \tag{3.8}$$

Analogously, we require for each non-initial state $s$ with a selected action-distribution pair $\eta \in P(s)$ that there is a selected action-distribution pair leading to $s$. Thus, we add for all states $s \in S \setminus \{s_{\mathrm{init}}\}$ the constraint

$$\sum_{\eta \in P(s)} \sigma_{s,\eta} \leq \sum_{s' \in \{s'' \in S \,|\, s'' \neq s \wedge \exists \eta \in P(s'').\, s \in \mathrm{supp}(\eta)\}} \; \sum_{\eta' \in \{\eta'' \in P(s') \,|\, s \in \mathrm{supp}(\eta'')\}} \sigma_{s',\eta'} \;. \tag{3.9}$$

As special cases of these cuts, we can encode that the initial state has at least one activated outgoing transition and that at least one of the target states has a selected incoming transition. These special cuts come with very few additional constraints and often have a great impact on the solution times.

**Label cuts.** In order to guide the solver to select the correct combinations of *labels* and *scheduler* variables, we want to enforce that for every selected label $\ell$ there is at least one scheduler variable $\sigma_{s,\eta}$ activated such that $\ell \in \bigcup_{s' \in \mathrm{supp}(\eta)} L(s, \eta, s')$:

$$x_\ell \;\leq \sum_{s \in S} \; \sum_{\eta \in \{\eta' \in P(s) \,|\, \exists s' \in \mathrm{supp}(\eta').\, \ell \in L(s,\eta',s')\}} \sigma_{s,\eta} \;. \tag{3.10}$$

**Synchronization cuts.** While scheduler and label cuts are applicable to the general smallest critical label set problem, synchronization cuts take the proper *synchronization of commands* into account and are applicable for the computation of smallest critical command sets only.

Let $M_i, M_j$ $(i \neq j)$ be two modules which synchronize on action $\alpha$, $c$ a command of $M_i$ with action $\alpha$, and $C_{j,\alpha}$ the set of commands with action $\alpha$ in module $M_j$. The following constraint ensures that if command $c$ is selected by activating the variable $x_{l_c}$ then at least one command $d \in C_{j,\alpha}$ is selected, too.

$$x_{\ell_c} \;\leq \sum_{d \in C_{j,\alpha}} x_{\ell_d} \;. \tag{3.11}$$

Similar constraints can be formulated for minimization of command branches.

3.6. **Further Simplification of Counterexamples.** Given a PA $\mathcal{M}$ as a PRISM program and a violated reachability property $\mathcal{P}_{\leq\lambda}(\Diamond T)$, we propose to extract a counterexample as follows:

(1) remove all *commands* which are not necessary for a violation of the property,
(2) remove all unnecessary *branches* of the remaining commands, and
(3) reduce the *domains* of the variables as much as possible.

These simplification steps are special cases of the SCL problem, as described in the previous section. As an alternative to the last step, the *intervals* of the variables can be reduced, which is also an application of the SCL problem, but with additional constraints.

3.7. **Reduction of Variable Intervals.** Using SCL to reduce variable domains as described in Section 3.1 can yield domains that are not intervals anymore. If intervals are desired, additional constraints are necessary to keep the domains connected.

For each variable $\xi \in \mathrm{Var}$ we encode an interval $[l_\xi, u_\xi] \subseteq \mathrm{dom}(\xi) \subseteq \mathbb{Z}$ by introducing two additional variables $h^u_{\xi,v}, h^l_{\xi,v} \in \{0,1\} \subseteq \mathbb{Z}$ for each $v \in \mathrm{dom}(\xi)$. The intuition is that $h^u_{\xi,v} = 1$ iff $v > u_\xi$, and $h^l_{\xi,v} = 1$ iff $v < l_\xi$. The remaining values $v \in \mathrm{dom}(\xi)$, for which $h^u_{\xi,v} = 0$ and $h^l_{\xi,v} = 0$ hold, form the interval. We add the following additional constraints to the MILP (3.1a)–(3.1j):

$$\forall \xi \in \mathrm{Var}.\, \forall v \in \mathrm{dom}(\xi).\, v \neq \min\big(\mathrm{dom}(\xi)\big). \quad h^l_{\xi,v} \leq h^l_{\xi,v-1} \tag{3.12a}$$

$$\forall \xi \in \mathrm{Var}.\, \forall v \in \mathrm{dom}(\xi).\, v \neq \max\big(\mathrm{dom}(\xi)\big). \quad h^u_{\xi,v} \leq h^u_{\xi,v+1} \tag{3.12b}$$

$$\forall \xi \in \mathrm{Var}.\, \forall v \in \mathrm{dom}(\xi). \quad h^l_{\xi,v} + h^u_{\xi,v} + x_{\ell_{\xi,v}} = 1 \tag{3.12c}$$

The first constraint takes care that, if a value $v$ is neglected (i. e., $h^l_{\xi,v} = 1$), also the value $v - 1$ is neglected. The same holds for $h^u_{\xi,v}$ and the successor value $v + 1$ in (3.12b). Constraint (3.12c) connects the decision variables $x_{\ell_{\xi,v}}$ for the labeling with the auxiliary variables $h^l_{\xi,v}$ and $h^u_{\xi,v}$: Exactly one of these three variables has to be set to 1—either $v$ is below the lower bound ($h^l_{\xi,v} = 1$) or above the upper bound ($h^u_{\xi,v} = 1$), or the label $\ell_{\xi,v}$ is contained in the computed label set.

## 4. Experiments

We have implemented the described techniques in C++ using the MILP solver `Gurobi` [24]. The experiments were performed on an Intel® Xeon® CPU E5-2643 with 3.3 GHz clock frequency and 32 GB of main memory, running Ubuntu 12.04 Linux in 64 bit mode. We focus on the minimization of the number of commands needed to obtain a counterexample and simplify them by deleting a maximum number of branchings and variable values. We do not consider symmetries in the models. We ran our tool with two threads in parallel and aborted any experiment which did not finish within 10 min (1200 CPU seconds). We conducted a number of experiments with benchmarks that are publicly available on the web page of `PRISM` [29]. We give a brief overview of the used models.

▶ `coin`-$N$-$K$ [30] models the shared coin protocol as in Example 2.7. The protocol is parameterized by the number $N$ of involved processes, which collectively undertake a random walk by flipping an unbiased coin and, depending on the outcome, incrementing or decrementing a shared counter. If the counter reaches a value greater than $KN$ for an integer constant $K > 1$ then the decision is heads, if it is less than $-KN$ then tails. We consider

the property $\mathcal{P}_{\leq\lambda}\big(\Diamond\,(\text{finished} \wedge \text{all\_coins\_equal\_0})\big)$, which is satisfied if the probability to finish the protocol with all coins equal to 0 is at most $\lambda$.

For $N = K = 2$, this probability is 0.5556. To show the applicability of our approach we introduce a "bug" by having a biased coin where the probability for `coin=0` is 0.8 for all processes. The probability is now 0.9999. If we search for a smallest critical command set for $\mathcal{P}$ with a probability bound of 0.5, which is the expected scenario, the command 2.4 as in Example 2.7 is not chosen. That means to observe faulty behavior it is not necessary to ever increment the counter. This gives us the hint that the fault is caused by the flipping command 2.2.

▶ `wlan`-$B$-$C$ models the two-way handshake mechanism of the IEEE 802.11 Wireless LAN protocol. Two stations try to send data, but run into a collision. Therefore they enter the randomized exponential backoff scheme. The parameter $B$ denotes the maximal allowed value of the backoff counter. We check the property $\mathcal{P}_{\leq\lambda}\big(\Diamond\,(\text{num\_collisions} = C)\big)$ putting an upper bound on the probability that a maximal allowed number $C$ of collisions occur.

▶ `csma`-$N$-$C$ concerns the IEEE 802.3 CSMA/CD network protocol. $N$ is the number of processes that want to access a common channel, $C$ is the maximal value of the backoff counter. We check $\mathcal{P}_{\leq\lambda}(\neg\text{collision\_max\_backoff}\,U\,\text{delivered})$ expressing that the probability that all stations successfully send their messages before a collision with maximal backoff occurs is at most $\lambda$.

▶ `fw`-$N$ models the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus (called "FireWire") [31]. It is a leader election protocol which is executed each time a node enters or leaves the network. The parameter $N$ denotes the delay of the wire as multiples of 10 ns. We check $\mathcal{P}_{\leq\lambda}(\Diamond\,\text{leader\_elected})$, i. e., that the probability of finally electing a leader is at most $\lambda$.

Some statistics of the models for different parameter values are shown in Table 1. The columns contain the name of the model ("Name"), its number of states ("#states"), transitions ("#trans"), modules ("#mod"), commands ("#comm"), branches of the commands ("#upd"), variables ("#var"), and different variable values ("#val"), i. e., $\sum_{\xi\in\text{Var}} |\,\text{dom}(\xi)|$. The values in braces are the number of commands and command branches, respectively, that are relevant for the considered property. Column "$\Pr^{+}(s_{\text{init}}, \Diamond\, T)$" contains the reachability probability and column "$\lambda$" the probability bound. The last column "MCS"shows the number of states in the minimal critical subsystem, i. e., the smallest subsystem of the PA such that the probability to reach a target state inside the subsystem is still above the bound. Entries which are marked with a star correspond to the smallest critical subsystem we could find within the time bound of 10 min using our tool `LTLSubsys` [10], but they are not necessarily optimal.

The results of our experiments computing a smallest critical command set are displayed in Table 2. The first column "Model" contains the name of the model. The following two blocks contain the results of runs without any cuts (cf. Section 3.5) and with the best combination of cuts: If there were cut combinations with which the MILP could be solved within the time limit, we report the fastest one. If all combinations timed out, we report the one that yielded the largest lower bound.

For the block without cuts, we give the number of variables ("Var.") and constraints ("Constr.") of the MILP, the computation time in seconds ("Time"), the memory consumption in MB ("Mem."), the number of commands in the critical command set ("$n$"), and, in case the time limit was exceeded, a lower bound on the size of the smallest critical command set ("lb"), which the solver obtains by solving a linear programming relaxation of the MILP.

Table 1: Model statistics

| Model | #states | #trans | #mod | #comm | #upd | #var | #val | $\Pr^+(s_{\text{init}}, \Diamond T)$ | $\lambda$ | MCS |
|---|---|---|---|---|---|---|---|---|---|---|
| coin2-1 | 144 | 252 | 2 | 14 (12) | 16 (12) | 5 | 19 | 0.6 | 0.4 | 13 |
| coin2-2 | 272 | 492 | 2 | 14 (12) | 16 (12) | 5 | 23 | 0.5556 | 0.4 | 25* |
| coin2-4 | 528 | 972 | 2 | 14 (12) | 16 (12) | 5 | 31 | 0.52940 | 0.4 | 55* |
| coin2-5 | 656 | 1212 | 2 | 14 (12) | 16 (12) | 5 | 35 | 0.52379 | 0.4 | 67* |
| coin2-6 | 784 | 1452 | 2 | 14 (12) | 16 (12) | 5 | 39 | 0.51998 | 0.4 | 83* |
| csma2-2 | 1038 | 1282 | 3 | 34 (34) | 42 (42) | 11 | 94 | 0.875 | 0.5 | 540 |
| csma2-4 | 7958 | 10594 | 3 | 38 (38) | 90 (90) | 11 | 122 | 0.99902 | 0.5 | 1769* |
| fw01 | 1743 | 2197 | 4 | 68 (64) | 72 (68) | 10 | 382 | 1.0 | 0.5 | 412 |
| fw04 | 5452 | 7724 | 4 | 68 (64) | 72 (68) | 10 | 394 | 1.0 | 0.5 | 412* |
| fw10 | 17190 | 29364 | 4 | 68 (64) | 72 (68) | 10 | 418 | 1.0 | 0.5 | 412* |
| fw15 | 33425 | 63379 | 4 | 68 (64) | 72 (68) | 10 | 438 | 1.0 | 0.5 | 412* |
| wlan0-2 | 6063 | 10619 | 3 | 70 (42) | 100 (72) | 13 | 91 | 0.18359 | 0.1 | 121 |
| wlan0-5 | 14883 | 26138 | 3 | 70 (42) | 100 (72) | 13 | 94 | 0.00114 | 0.001 | 952* |
| wlan2-1 | 28597 | 57331 | 3 | 76 (14) | 114 (14) | 13 | 100 | 1.0 | 0.5 | 7 |
| wlan2-2 | 28598 | 57332 | 3 | 76 (42) | 114 (72) | 13 | 101 | 0.18260 | 0.1 | 121* |
| wlan2-3 | 35197 | 70216 | 3 | 76 (42) | 114 (78) | 13 | 102 | 0.01793 | 0.01 | 514* |
| wlan3-1 | 96419 | 204743 | 3 | 78 (14) | 130 (14) | 13 | 110 | 1.0 | 0.5 | 7 |
| wlan3-2 | 96420 | 204744 | 3 | 78 (42) | 130 (72) | 13 | 111 | 0.18359 | 0.1 | 121* |

An entry "??" for the number of commands means that the solver was not able to find a non-trivial critical command set within the time limit. For the best cut combination, the last four columns specify the combination of cuts leading to the best running time. Here the column "$\sigma_f$" corresponds to scheduler forward cuts (3.8), "$\sigma_b$" to scheduler backward cuts (3.9), "$\ell$" to label cuts (3.10), and "$\|$" to synchronization cuts (3.11). An entry "$\surd$" indicates that the corresponding constraints have been added to the MILP, "$\times$" that they were not used.

Although we ran into timeouts for many instances, in particular without any cuts, in almost all cases a solution could be found within the time limit. We suppose that also the solutions of the aborted instances are optimal or close to optimal. It seems that the MILP solver is able to quickly find good (or even optimal) solutions due to sophisticated heuristics, but proving their optimality is hard. A solution is proven optimal as soon as the objective value of the best solution and the lower bound coincide. The additional cuts strengthen this lower bound considerably. Further experiments have shown that the scheduler forward cuts of Eq.(3.8) have the strongest effect on the lower bound. Choosing good cuts consequently enables the solver to obtain optimal solutions for more benchmarks.

Table 3 contains the results of the subsequent simplification steps: To the `PRISM` model corresponding to the smallest critical command set we applied minimization of the commands' branches and finally to its result the minimization of variable domains. For both we give, as before, the computation time in seconds, the memory consumption in megabytes, the resulting number of branches and variable values, respectively, and, if a time out occured, the computed lower bound. For both simplification steps we report only the running times obtained using the best combination of cuts. In all experiments that terminated within the time limit, the branch and variable domain minimization were faster than the previous command selection using the same combination of cuts.

Table 2: Results for command minimization (TO = 600 seconds)

| | no cuts | | | | | | best cut combination | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Var. | Constr. | Time | Mem. | $n$ | lb | Time | Mem. | $n$ | lb | $\sigma_f$ | $\sigma_b$ | $\ell$ | $\|$ |
| coin2-1 | 277 | 492 | TO | 855 | 9 | 8 | 69.05 | 42 | 9 | opt | √ | √ | √ | × |
| coin2-2 | 533 | 1004 | TO | 1140 | 9 | 6 | TO | 1111 | 9 | 8 | × | × | √ | √ |
| coin2-4 | 1045 | 2028 | TO | 553 | 9 | 6 | TO | 974 | 9 | 7 | × | √ | √ | √ |
| coin2-5 | 1301 | 2540 | TO | 768 | 9 | 5 | TO | 554 | 9 | 6 | × | √ | √ | √ |
| coin2-6 | 1557 | 3052 | TO | 698 | 9 | 5 | TO | 688 | 9 | 6 | × | √ | √ | × |
| csma2-2 | 2123 | 5990 | 7.30 | 26 | 32 | opt | 4.19 | 28 | 32 | opt | × | × | × | √ |
| csma2-4 | 15977 | 46882 | 196.11 | 215 | 36 | opt | 77.43 | 261 | 36 | opt | × | √ | √ | √ |
| fw01 | 3974 | 13121 | TO | 148 | 28 | 27 | 29.43 | 122 | 28 | opt | √ | × | × | × |
| fw04 | 13144 | 43836 | TO | 604 | 28 | 22 | 103.37 | 296 | 28 | opt | √ | √ | × | √ |
| fw10 | 46282 | 153764 | TO | 954 | 28 | 15 | 390.82 | 1102 | 28 | opt | √ | × | × | √ |
| fw15 | 96222 | 318579 | TO | 1494 | 28 | 10 | TO | 1861 | 28 | 19 | √ | √ | × | × |
| wlan0-2 | 7072 | 6602 | TO | 474 | 33 | 15 | TO | 373 | 33 | 31 | √ | √ | √ | √ |
| wlan0-5 | 19012 | 25808 | TO | 1083 | ?? | ?? | TO | 1083 | ?? | ?? | × | × | × | × |
| wlan2-1 | 28538 | 192 | 1.12 | 45 | 8 | opt | 0.82 | 45 | 8 | opt | √ | √ | × | × |
| wlan2-2 | 29607 | 6602 | TO | 517 | 33 | 15 | TO | 416 | 33 | 31 | √ | √ | × | √ |
| wlan2-3 | 36351 | 18922 | TO | 809 | 38 | 14 | TO | 394 | 38 | 32 | √ | √ | × | √ |
| wlan3-1 | 96360 | 192 | 1.98 | 142 | 8 | opt | 1.98 | 142 | 8 | opt | × | × | × | × |
| wlan3-2 | 97429 | 6602 | TO | 552 | 33 | 15 | TO | 518 | 33 | 31 | √ | √ | × | √ |

Table 3: Results for branch and variable domain minimization (TO = 600 seconds)

| | branch minimization | | | | var. domain minimization | | | |
|---|---|---|---|---|---|---|---|---|
| Model | Time | Mem. | $n$ | lb | Time | Mem. | $n$ | lb |
| coin2-1 | 3.78 | 16 | 11 | opt | 0.39 | 10 | 15 | opt |
| coin2-2 | TO | 3293 | 11 | opt | TO | 1142 | 19 | 17 |
| coin2-4 | TO | 1548 | 11 | 9 | TO | 938 | 25 | 15 |
| coin2-5 | TO | 814 | 11 | 9 | TO | 809 | 29 | 15 |
| coin2-6 | TO | 1186 | 11 | 9 | TO | 515 | 32 | 13 |
| csma2-2 | 0.46 | 15 | 39 | opt | 0.33 | 18 | 92 | opt |
| csma2-4 | TO | 135 | 70 | 69 | 3.16 | 102 | 120 | opt |
| fw01 | 0.11 | 11 | 27 | opt | 0.14 | 14 | 342 | opt |
| fw04 | 0.31 | 26 | 30 | opt | 0.33 | 32 | 342 | opt |
| fw10 | 0.81 | 58 | 29 | opt | 1.16 | 72 | 342 | opt |
| fw15 | 1.37 | 88 | 27 | opt | 2.30 | 112 | 342 | opt |
| wlan0-2 | 1.58 | 23 | 31 | opt | 0.96 | 241 | 50 | opt |
| wlan2-1 | 0.11 | 30 | 8 | opt | 1.01 | 260 | 25 | opt |
| wlan2-2 | TO | 1533 | 49 | 43 | 5.52 | 251 | 67 | opt |
| wlan2-3 | TO | 332 | 37 | 32 | TO | 865 | 51 | 40 |
| wlan3-1 | 0.47 | 91 | 8 | opt | 3.57 | 871 | 25 | opt |
| wlan3-2 | TO | 1624 | 49 | 42 | 8.68 | 889 | 66 | opt |

We have also ran the two simplification steps directly on the original `PRISM` model with all commands. There the computation times were comparable to those of determining a smallest critical command set (cf. Table 2). Thus we suppose that the much smaller times for simplification after selecting a smallest critical command set are due to the considerably reduced possibilities for simplification.

The experiments show that is feasible to use MILP formulations for counterexample computation, although solving the MILPs is costly and often optimality of the result cannot be proven by the solver within the given time limit. Additionally we can see that in all cases we are able to reduce the number of commands and to simplify the commands, in some cases considerably, compared to the original `PRISM` model.

## 5. Conclusion

We have presented a new type of counterexamples for probabilistic automata which are described using a guarded command language: We computed a smallest subset of the commands which alone induces an erroneous system. This requires the solution of a mixed integer linear program whose size is linear in the size of the state space of the PA. State-of-the-art MILP solvers apply sophisticated techniques to find small command sets quickly, but they are often unable to prove the optimality of their solution.

For the MILP formulation of the smallest critical labeling problem we both need decision variables for the labels and for the scheduler inducing the maximal reachability probabilities of the subsystem. On the other hand, model checking can be executed without any decision variables using a linear programming formulation. We therefore coupled a MAXSAT solver with a model checker for PAs. For many benchmark instances this reduced the computation time significantly. First results on this alternative method have been published in [32].

## References


[1] Clarke, E.M.: The birth of model checking. In: 25 Years of Model Checking – History, Achievements, Perspectives. Vol. 5000 of LNCS. Springer (2008) 1–26

[2] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. of CAV. Vol. 1855 of LNCS, Springer (2000) 154–169

[3] Clarke, E.M., Jha, S., Lu, Y., Veith, H.: Tree-like counterexamples in model checking. In: Proc. of LICS, IEEE Computer Society (2002) 19–29

[4] Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing **2**(2) (1995) 250–273

[5] Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering **35**(2) (2009) 241–257

[6] 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) (2010) 37–60

[7] Wimmer, R., Braitling, B., Becker, B.: Counterexample generation for discrete-time Markov chains using bounded model checking. In: Proc. of VMCAI. Vol. 5403 of LNCS, Springer (2009) 366–380

[8] Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Proc. of HVC. Vol. 5394 of LNCS, Springer (2008) 129–148

[9] Jansen, N., Ábrahám, E., Katelaan, J., Wimmer, R., Katoen, J.P., Becker, B.: Hierarchical counterexamples for discrete-time Markov chains. In: Proc. of ATVA. Vol. 6996 of LNCS, Springer (2011) 443–452

[10] Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J.P., Becker, B.: Minimal critical subsystems for discrete-time Markov models. In: Proc. of TACAS. Vol. 7214 of LNCS, Springer (2012) 299–314

[11] Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J.P., Becker, B.: Minimal counterexamples for linear-time probabilistic verification. Theoretical Computer Science **549** (2014) 61–100

[12] Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1986)

[13] Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.P., Wimmer, R.: Counterexample generation for discrete-time Markov models: An introductory survey. In: $14^{\text{th}}$ International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM), Advanced Lectures. Vol. 8483 of LNCS. Springer (2014) 65–121

[14] Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design **15**(1) (1999) 7–48
[15] Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. of CAV. Vol. 6806 of LNCS, Springer (2011) 585–591
[16] He, J., Seidel, K., McIver, A.: Probabilistic models for the guarded command language. Science of Computer Programming **28**(2–3) (1997) 171–192
[17] Dijkstra, E.W.: Guarded commands, non-determinacy and formal derivation of programs. Communications of the ACM **18**(8) (1975) 453–457
[18] Katoen, J.P., van de Pol, J., Stoelinga, M., Timmer, M.: A linear process-algebraic format with data for probabilistic automata. Theoretical Computer Science **413**(1) (2012) 36–57
[19] Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.P., Becker, B.: High-level counterexamples for probabilistic automata. In: Proc. of QEST. Vol. 8054 of LNCS, Springer (2013) 18–33
[20] Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
[21] Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. Journal of Algorithms **15**(1) (1990) 441–460
[22] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co Ltd (1979)
[23] Achterberg, T.: SCIP: Solving constraint integer programs. Mathematical Programming Computation **1**(1) (2009) 1–41
[24] Gurobi Optimization, Inc.: Gurobi optimizer reference manual. `http://www.gurobi.com` (2012)
[25] Chadha, R., Viswanathan, M.: A counterexample-guided abstraction-refinement framework for Markov decision processes. ACM Trans. on Computational Logic **12**(1) (2010) 1–45
[26] Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: Proc. of PAPM/PROBMIV. Vol. 2399 of LNCS, Springer (2002) 169–187
[27] Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Proc. of FSTTCS. Vol. 1026 of LNCS, Springer (1995) 499–513
[28] Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J.P., Becker, B.: Minimal counterexamples for refuting $\omega$-regular properties of Markov decision processes. Reports of SFB/TR 14 AVACS 88 (2012) ISSN: 1860-9821, `http://www.avacs.org`.
[29] Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. of QEST, IEEE Computer Society (2012) 203–204
[30] Kwiatkowska, M., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: Proc. of CAV. Vol. 2102 of LNCS, Springer (2001) 194–206
[31] Stoelinga, M.: Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 Root Contention Protocol. Formal Aspects of Computing **14**(3) (2003) 328–337
[32] Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.P.: Fast debugging of PRISM models. In: Proc. of ATVA. LNCS, Springer (2014) 146–162

# Appendix

## Appendix A. Proof of Theorem 3.2

Let $\mathcal{M} = (S, s_{\text{init}}, \text{Act}, P)$ be a PA, $L : S \times \text{Act} \times \text{Distr}(S) \times S \nrightarrow 2^{\text{Lab}}$ a labeling function and $w : \text{Lab} \to \mathbb{R}^{\geq 0}$ a weight function. Assume the reachability property $\mathcal{P}_{\leq\lambda}(\Diamond\, T)$ is violated.

**Theorem A.1.** *To decide whether there is a critical label set* $\text{Lab}' \subseteq \text{Lab}$ *with* $w(\text{Lab}') \leq k$ *is NP-complete.*

*Proof.* That the decision problem is in NP is obvious: we can guess a label set $\text{Lab}' \subseteq \text{Lab}$ and verify in polynomial time by computing the reachability probability $\Pr^+(s_{\text{init}}, \Diamond\, T)$, that $w(\text{Lab}') \leq k$ and that $\Pr^+(s_{\text{init}}, \Diamond\, T) > \lambda$.

To prove the NP-hardness, we give a reduction from exact 3-cover (X3C) [22], similar to [25]. X3C is defined as follows:

> Let $X$ be a finite set with $|X| = 3r$ and a collection $C \subseteq 2^X$ of subsets with $|c| = 3$ for all $c \in C$. Is there a collection of pairwise disjoint sets $B \subseteq C$ such that $X = \bigcup_{c \in B} c$?

We note that $X$ has an exact 3-cover iff it has a cover of size $|B| \leq r$.

Given $X$ and $C$, we construct a PA $\mathcal{M} = (S, s_{\text{init}}, \{\alpha\}, P)$ as follows: $S = X \,\dot\cup\, C \,\dot\cup\, \{s_{\text{init}}, t\}$ with two fresh states $s_{\text{init}}$ and $t$. We set $P(s_{\text{init}}) = \big\{(\alpha, \mu_X)\big\}$ with $\mu_X(x) = \frac{1}{|X|}$ for all $x \in X$ and $\mu_X(s) = 0$ for all $s \in S \setminus X$. For $x \in X$ we define $P(x) = \big\{(\alpha, \mu_c^1) \,\big|\, c \in C \text{ with } x \in c\big\}$, where $\mu_c^1(c) = 1$ and $\mu_c^1(s) = 0$ for all $s \neq c$. Finally, $P(c) = \big\{(\alpha, \mu_t^1)\big\}$ for all $c \in C$, and $P(t) = \emptyset$.

We use the following label set $\text{Lab} = \{\ell_{s_{\text{init}}}\} \cup \{\ell_x \,|\, x \in X\} \cup \{l_t\}$ and label the branches as follows: all branches in $P(s_{\text{init}})$ are labeled with $\ell_{s_{\text{init}}}$, the branches in $P(x)$ with $\ell_x$ for $x \in X$ and the branches in $P(c)$ with $\ell_t$ for $c \in C$. The weight function $w$ assigns 0 to $\ell_{s_{\text{init}}}$ and $\ell_t$, and 1 to all other labels. The set of target states is $\{t\}$, and the probability bound $\lambda = 1 - \frac{1}{|X|+1}$.

We claim that there is a critical label set with weight $\leq r$ iff $X$ has an exact 3-cover.

"$\Rightarrow$" Let $\text{Lab}' \subseteq \text{Lab}$ be a critical label set with $w(\text{Lab}') \leq r$. We can assume w. l. o. g. that $\ell_{s_{\text{init}}}$ and $\ell_t$ are contained in $\text{Lab}'$ since their weight is zero. We observe that from each state $x \in X$ there has to be a path to state $t$ in the induced sub-PA $\mathcal{M}_{|\text{Lab}'}$. Otherwise the maximal probability to reach $t$ from $s_{\text{init}}$ is $\leq 1 - \frac{1}{|X|} < 1 - \frac{1}{|X|+1}$. This means, for each $x \in X$ there is $c \in C$ with $x \in c$ and $\ell_c \in \text{Lab}'$. Let $B = \{c \in C \,|\, \ell_c \in \text{Lab}'\}$. $B$ is a cover of $X$. Since $w(\text{Lab}') \leq r$, $|B| \leq r$ and $B$ is an exact cover.

"$\Leftarrow$" Let $B \subseteq C$ be an exact cover of $X$. We set $\text{Lab}' = \{\ell_c \,|\, c \in B\} \cup \{\ell_{s_{\text{init}}}, \ell_t\}$. Then $w(\text{Lab}') = r$. For all $x \in X$ there is $c \in B$ such that $x \in c$, because $B$ is a cover. That means, for all $x \in X$ there is a path from $x$ to $t$ with probability 1 in $\mathcal{M}_{|\text{Lab}'}$. Since $\ell_{s_{\text{init}}} \in \text{Lab}'$, we have that $\Pr^+(s_{\text{init}}, \Diamond\, \{t\}) = 1$. Hence, $\text{Lab}'$ is critical. □