Debugging of Markov Decision Processes (MDPs) Models

Hichem Debbi
Department of Computer Science
University of M'sila
M'sila, Algeria
hichem.debbi@univ-msila.dz

In model checking, a counterexample is considered as a valuable tool for debugging. In Probabilistic Model Checking (PMC), counterexample generation has a quantitative aspect. The counterexample in PMC is a set of paths in which a path formula holds, and their accumulative probability mass violates the probability threshold. However, understanding the counterexample is not an easy task. In this paper we address the task of counterexample analysis for Markov Decision Processes (MDPs). We propose an aided-diagnostic method for probabilistic counterexamples based on the notions of causality, responsibility and blame. Given a counterexample for a Probabilistic CTL (PCTL) formula that does not hold over an MDP model, this method guides the user to the most relevant parts of the model that led to the violation.

1 Introduction

Probabilistic Model Checking (PMC) has appeared as an extension of model checking for analysing systems that exhibit stochastic behaviour. Several case studies in several domains have been addressed from randomized distributed algorithms and network protocols to biological systems and cloud computing environments. These systems are described usually using Discrete-Time Markov Chain (DTMC), Continuous Time Markov Chain (CTMC) and Markov Decision Process (MDP), and verified against properties specified in Probabilistic Computation Tree Logic (PCTL) [21] or Continuous Stochastic Logic (CSL) [8], [9].

One of the major advantages of model checking over other formal methods its ability to generate a counterexample when the model falsifies such specification. A counterexample is an error trace, by analyzing it, the user can locate the source of the error. Unlike the previous methods proposed for conventional model checking that generate the counterexample as a single path ending with bad states representing the failure, the task in PMC is quite different. The counterexample in PMC is a set of evidences or diagnostic paths that satisfy the path formula and their probability mass violates the probability bound. As it is in conventional model checking, the generated counterexample should be small and most indicative to be easy for understanding . In PMC, this task is more challenging, since the counterexample consists of multiple paths and it is probabilistic. In [6], the authors introduced a heuristic-based search method for generating counterexamples for DTMCs and CTMCs as what they refer to as diagnostic sub-graphs. In complementary work [5], they proposed an approach for counterexample generation for MDPs based on existing methods for DTMC. Based on these works, they built an open source tool, DiPro [4], for generating indicative counterexamples for DTMCs, CTMCs and MDPs. Similar to the previous works, [20] has proposed the notion of smallest most indicative counterexample that reduces to the problem of finding K shortest paths. Instead of generating path-based counterexamples, the authors in [28] have proposed a novel approach based on critical subsystems. Following this work, [24] proposed the COMICS tool for generating the critical subsystems that induce the counterexamples. Instead of relying on the state space search resulted from the parallel composition of the modules, [29] suggests

G. Gössler and O. Sokolsky (Eds.): 1st Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies (CREST'16)
EPTCS 224, 2016, pp. 25–39, doi:10.4204/EPTCS.224.4

© H.Debbi
This work is licensed under the
Creative Commons Attribution License.

to rely directly on the guarded command language used by the model checker, which is more likely and helpful for debugging purpose.

Generating small and indicative counterexamples only is not enough for understanding the error. Therefore, in conventional model checking many works have addressed the analysis of counterexamples to better understand the failure [18],[23],[10]. As it was done in conventional model checking, addressing the error explanation in probabilistic model checking is highly required, especially that probabilistic counterexample consists of multiple paths instead of single path and it is probabilistic. Debbi [13] has used the definition of causality [19] for debugging probabilistic models. It has been used with regression analysis in the aim to estimate the causal effect of the Boolean variables on the violation of the probabilistic property, and how these variables change their values dependently, the thing that can help to understand the behavior of the model. The same definition of causality has also been adapted to event orders for generating fault trees from probabilistic counterexamples , where the selection of traces forming the fault tree are restricted to some minimality condition [27]. To do so, they proposed the event order logic to reason about boolean conditions on the occurrence of events, where the cause of the hazard in their context is presented as a Event Order Logic (EOL) formula, which is a conjunction of events, and the event are simply actions leading from state to another. In [17], they extended their approach by integrating causality in explicit-state model checking algorithm to give a causal interpretation for sub and super-sets of execution traces. They proved the applicability of their approach to many industrial size PROMELA models. They aimed to extended the causality checking approach to probabilistic counterexamples by computing the probabilities of events combination [26], but they still consider the use of causality checking of qualitative PROMELA models. They proposed a symbolic version of causality checking [11] based on bounded model checking (BMC) and SAT solving.

In previous work [14], we have addressed the analysis of probabilistic counterexamples for DTMCs and CTMCs using the definition of causality by Halpern and Pearl [19] and its extension, responsibility [12]. Markov Decision Process (MDP) is a discrete time probabilistic model similar to DTMC, the only difference is that MDP is non-deterministic through the possible actions that can be taken at each state. Hence, for analyzing counterexamples for MDPs, we have to deal with actions as well. To this end, we adapt the definitions of causality and responsibility to reason about causes, in addition we adapt the definition of blame [12] to reason about actions. All the definitions are adapted in complementary way. Following that, we introduce an algorithm for identifying the relevant parts of the model that contribute the most to the violation of PCTL properties through computing responsibility and blame. We refer to the output of the algorithm as *diagnoses*. So, compared to the work [14], this paper does not deal only with the states and the Boolean variables that are considered as causes, but also takes in consideration the transitions leading to them, and under which actions the transitions are enabled. With all these information in hand, we can easily go-back to the model described in guarded command language of PRISM model checker and locate the commands that contribute the most to the violation. In this paper we will focus on probabilistic safety properties with upper threshold. The properties with lower threshold can be easily transformed to properties with upper threshold [6], [20].

The rest of this paper is organized as follows. In section 2, we present some preliminaries and definitions. In section 3, we show how the notions of causality, responsibility and blame can be adapted for explaining probabilistic counterexamples for MDPs models, following that an algorithm is presented. Experimental results are presented in section 4. At the end, we present conclusion and future works.

2 Preliminaries and Definitions

Definition 2.1 *(**Markov Decision Process (MDP)**) is a tuple $M = (S, s_{init}, A, P, L)$, where S is a finite set of states, $s_{init} \in S$ is the initial state, A is a set of actions, $P : S \times A \times S \to [0,1]$ is a probability transition function such that for every state $s \in S$ and an action $\alpha \in A$: $\sum_{s' \in S} P(s, \alpha, s') \in \{0,1\}$, and $L : S \to 2^{AP}$ is a labeling function, which assigns to each state $s \in S$ a subset of the finite set of atomic propositions AP.*

At each state s, the probability of moving to a successor state s' by taking an action α is given by $P(s, \alpha, s')$. We say that an action α is enabled in state s if and only if $\sum_{s' \in S} P(s, \alpha, s') = 1$, otherwise the action α is disabled. For each state $s \in S$ there is at least one action enabled. We denote the set of actions enabled from a state s by $A(s)$.

An infinite path σ is an infinite sequence $\sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \ldots$ with $\alpha_i \in A(s_i)$ such that $P(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 0$. We define the set of infinite paths starting from a state s_0 by $Paths(s_0)$. A finite path is a finite prefix of an infinite path. We denote by $FinitePaths(s_0)$ the finite paths starting from a state s_0. For Discrete-time Markov chains (DTMCs), the underlying σ-algebra is formed by the cylinder sets which are induced by $FinitePaths(s_0)$. For MDPs, computing the probabilities of paths must rely on the resolution of non-determinism, which is performed by a scheduler. A scheduler d resolves the non-determinism by taking in each state one of the enabled actions $\alpha \in A(s)$, thus resulting in DTMC for which the probability of paths is measurable. We refer to the set of infinite paths under this scheduler as $Paths_d(s_0)$. Then, the underlying σ-algebra is formed by the cylinder sets which are induced by finite paths under this scheduler denoted $FinitePaths_d(s_0)$. The probability of this cylinder set is computed by using the following formula:

$$P_d(\sigma \in FinitePaths_d(s_0) | \sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n) = \prod_{i \leq 0 < n} P(s_i, \alpha_i, s_{i+1}) \tag{1}$$

2.1 Probabilistic Computation Tree Logic (PCTL)

The Probabilistic Computation Tree Logic (PCTL) has appeared as an extension of CTL for the specification of systems that exhibit stochastic behaviour. PCTL state formulas are formed over the set of atomic propositions AP according to the following grammar:

$$\phi ::= true | a | \neg\phi | \phi_1 \wedge \phi_2 | \mathbf{P}_{\sim p}(\varphi)$$

Where $a \in AP$ is an atomic proposition, φ is a path formula, $\mathbf{P}$ is a probability threshold operator, $\sim \in \{<, \leq, >, \geq\}$ is a comparison operator, and p is a probability threshold. The path formulas φ are formed according to the following grammar:

$$\varphi ::= \phi_1 \mathbf{U} \phi_2 | \phi_1 \mathbf{W} \phi_2 | \phi_1 \mathbf{U}^{\leq n} \phi_2 | \phi_1 \mathbf{W}^{\leq n} \phi_2$$

Where ϕ_1 and ϕ_2 are state formulas and $n \in N$. The PCTL formula is a state formula, where path formulas only occur inside the operator $\mathbf{P}$.

The satisfaction of $\mathbf{P}_{\sim p}(\varphi)$ on DTMC depends on the probability mass of set of paths satisfying φ. This set is considered as a countable union of cylinder sets, so that, its measurability is ensured. A formula $\mathbf{P}_{\sim p}(\varphi)$ is satisfied on an MDP M if only if for every $d \in D$: $\mathbf{P}_d(\varphi) \sim p$, where D represents the set of all schedulers and $\mathbf{P}_d(\varphi)$ represents the probability of the set of all finite paths satisfying φ under the scheduler d.

The semantics of PCTL state and of path formulas for MDPs are defined as the same as for DTMCs as follows.

$s \models true \Leftrightarrow true$
$s \models a \Leftrightarrow a \in L(s)$
$s \models \neg\phi \Leftrightarrow s \not\models \phi$
$s \models \phi_1 \wedge \phi_2 \Leftrightarrow s \models \phi_1 \wedge s \models \phi_2$

Given a path $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} ...$ and an integer $j \geq 0$, where $\sigma[j] = s_j$, the semantics of PCTL path formulas is defined as for CTL as follows:

$\sigma \models \phi_1 \mathbf{U} \phi_2 \Leftrightarrow \exists j \geq 0.\sigma[j] \models \phi_2 \wedge (\forall 0 \leq k < j.\sigma[k] \models \phi_1)$
$\sigma \models \phi_1 \mathbf{W} \phi_2 \Leftrightarrow \sigma \models \phi_1 \mathbf{U} \phi_2 \vee (\forall k \geq 0.\sigma[k] \models \phi_1)$
$\sigma \models \phi_1 \mathbf{U}^{\leq n} \phi_2 \Leftrightarrow \exists 0 \leq j \leq n.\sigma[j] \models \phi_2 \wedge (\forall 0 \leq k < j.\sigma[k] \models \phi_1)$
$\sigma \models \phi_1 \mathbf{W}^{\leq n} \phi_2 \Leftrightarrow \sigma \models \phi_1 \mathbf{U}^{\leq n} \phi_2 \vee (\forall 0 \leq k \leq n.\sigma[k] \models \phi_1)$

We should mention that for model checking of MDPs we have to consider either maximizing or minimizing scheduler. Let $P_{max}(\varphi)$ be the maximal probability of φ where $P_{max}(\varphi) = max\{P_d(\varphi)|d \in D\}$, and dually the minimal probability $P_{min}(\varphi)$ be the minimal probability of φ where $P_{min}(\varphi) = min\{P_d(\varphi)|d \in D\}$. For instance for properties of upper threshold , it is evident that $(M \not\models P_{\leq p}(\varphi)) \Leftrightarrow P_{max}(\varphi) > p$.

2.2 Probabilistic Counterexamples

The PCTL property $\phi = \mathbf{P}_{\leq p}(\varphi)$ is violated on an MDP, if there exists at least one scheduler d such that the probability mass of the paths satisfying φ under d exceeds the bound p. A probabilistic counterexample for the property $\phi = \mathbf{P}_{\leq p}(\varphi)$ can be formed of a set of paths from $FinitePaths_d(s_0)$ starting at state s_0 and satisfying the path formula φ such that $P_{max}(\varphi) > p$ for some scheduler d. We denote this set by $FinitePaths_d(s_0 \models \phi)$. These finite paths are also called diagnostic paths [6],[5].

It is clear that given a scheduler d, it is possible to find a set of probabilistic counterexamples under d denoted $PCX_d(s_0 \models \phi)$, which is a set of any combination from $FinitePaths_d(s_0 \models \phi)$, their probability mass exceeds the bound p. Among all these probabilistic counterexamples, we are interested by the most indicative one. A most indicative counterexample is a minimal counterexample (has the least number of paths from $FinitePaths_d(s_0 \models \phi)$) and its probability mass is the highest among all other minimal counterexamples. We denote a most indicative probabilistic counterexample by $MIPCX_d(s_0 \models \phi)$. We should mention that a most indicative probabilistic counterexample may not be unique.

Lemma 2.1 *For every path $\sigma \in MIPCX_d(s_0 \models \phi)$ on which the property $\phi_1 \boldsymbol{U} \phi_2 (\phi_1 \boldsymbol{U}^{\leq n} \phi_2)$ is satisfied, the right state sub-formula (ϕ_2) is satisfied in the last state of σ.*

Example 1 Let us consider the example of MDP shown in Figure 1 and the property $P_{\leq 0.5}(\varphi)$, where $\varphi = (a \vee b) U (c \wedge d)$.

This property is violated in this model $(s_0 \not\models P_{\leq 0.5}(\varphi))$, since there exists a scheduler d (of α-actions) that induces a set of finite paths satisfying φ and their probability mass is greater than the probability bound (0.5). Any combination from $FinitePaths_d(s_0 \models \phi)$ having probability mass greater than 0.5, is a valid probabilistic counterexample including the whole set. Let us take the following counterexamples:

$P(CX_1) = P(\{s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_7, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_3, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_4 \xrightarrow{\alpha_4} s_3, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_4 \xrightarrow{\alpha_4} s_5, s_0 \xrightarrow{\alpha_0} s_4 \xrightarrow{\alpha_4} s_5$
$\})$
$= 0.25 + 0.2 + 0.09 + 0.15 + 0.12 = 0.81$

$P(CX_2) = P(\{s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_7, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_4 \xrightarrow{\alpha_4} s_5, s_0 \xrightarrow{\alpha_0} s_4 \xrightarrow{\alpha_4} s_5\})$
$= 0.25 + 0.15 + 0.12 = 0.52$

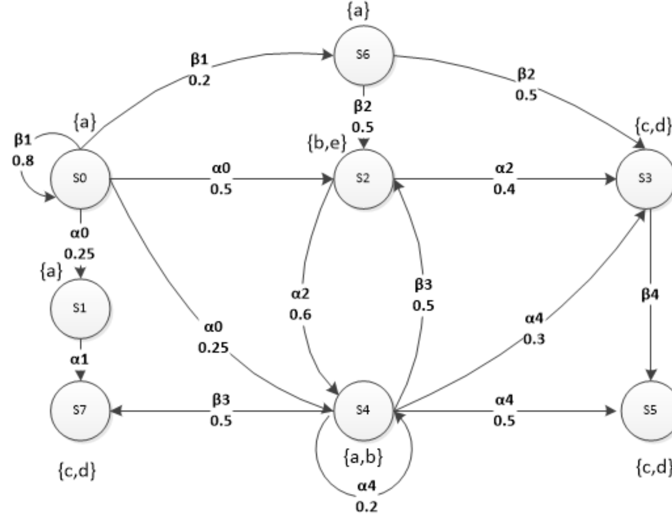

Figure 1: An Example for an MDP

$$P(CX_3) = P(\{s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_7, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_3, s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_4 \xrightarrow{\alpha_4} s_5\})$$
$$= 0.25 + 0.2 + 0.15 = 0.60$$

The last probabilistic counterexample CX_3 is most indicative since it is minimal and its probability is greater than the other minimal counterexample CX_2.

3 Generating Diagnoses for MDPs

3.1 Causality and Responsibility for $MIPCX_d(s_0 \models \phi)$

As the same as for DTMCs, explaining the violation of PCTL properties of the form $\phi = \mathbf{P}_{\leq p}(\varphi)$ for MDPs reduces to the explanation of exceeding the probability bound over the MDP model. MDP is a discrete time probabilistic model similar to DTMC. Therefore, the definitions of causality and responsibility can be adapted for $MIPCX_d(s_0 \models \phi)$ in the same way they have been adapted for counterexamples of DTMCs [14].

Definition 3.1 ***(Causality Model)*** *A causality model for a most indicative probabilistic counterexample $MIPCX_d(s_0 \models \phi)$ is a tuple $M = (U,V,F)$, where the set U is represented by a context variable; its value u represents a state s. V is a set of atomic propositions and Boolean formulas. F associates with every variable $V_i \in V$ a truth function f_{V_i} that determines the value of V_i (0 or 1), given a state s and the values of other variables in V.*

Let us denote by $MIPCX_d(s, \widehat{X \leftarrow x'})(s_0 \models \phi)$ the set of finite paths resulted from $MIPCX_d(s_0 \models \phi)$ by switching the value of a variable $X \in V$ in a state s.

Definition 3.2 ***(Criticality)*** *Consider a counterexample $MIPCX_d(s_0 \models \phi)$ for a probabilistic formula $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, a state s in $MIPCX_d(s_0 \models \phi)$ and a variable $X \in V$ that has a value $x \in \{0,1\}$ in s. We say that a pair $(s, X = x)$ is critical for the violation of $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, if $MIPCX_d(s, \widehat{X \leftarrow x'})(s_0 \models \phi)$ is not a valid counterexample for $\phi = \boldsymbol{P}_{\leq p}(\varphi)$.*

Definition 3.3 ***(Actual Cause)*** *Consider a counterexample $MIPCX_d(s_0 \models \phi)$ for a probabilistic formula $\phi = \boldsymbol{P}_{\leq p}(\varphi)$ and a variable $X \in V$. We say that $(s, X = x)$ is a cause for the violation of $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, if $(s, X = x)$ is critical , or there exists a subset of variables W of V such that switching their values in s makes $(s, X = x)$ critical.*

Thus, in our setting, we can refer to a cause as a pair $(s, X = x)$ where X is an atomic proposition has the value 1 in s if $X \in L(s)$, and it has the value 0 otherwise. $(s, X = x)$ is said to be cause, if it is critical, or it can be made critical by switching the values of a set of variables W in s.

Definition 3.4 ***(Responsibility)*** *Consider a counterexample $MIPCX_d(s_0 \models \phi)$ for a probabilistic formula $\phi = \boldsymbol{P}_{\leq p}(\varphi)$ and a variable $X \in V$. The degree of responsibility of a cause $(s, X = x)$ for the violation of $\phi = \boldsymbol{P}_{\leq p}(\varphi)$ denoted $dR(s, X = x, \phi)$is 1 if $(s, X = x)$ is critical, and otherwise is* $1/(|W| + 1)$.

That is, we can think of responsibility as a quantitative measure that gives us diagnostic information on $(s, X = x)$ being a cause for the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$, where the cause having the highest responsibility for the violation is the critical one.

Definition 3.5 ***(Probabilistic Causality Model)*** *A probabilistic causality model for $MIPCX_d(s_0 \models \phi)$ is a tuple (M, Pr), where M is the causality model and Pr is a probability function defined over the states of $MIPCX_d(s_0 \models \phi)$. We define for each state s from $MIPCX_d(s_0 \models \phi)$ its probability as follows:*

$$Pr(s) = \sum_{\sigma \in MIPCX_d(s_0 \models \phi) | s \in \sigma} P(\sigma) \tag{2}$$

Since every variable in V is a function of U, we can define the probability of each cause $(s, X = x)$ in the same way:

$$Pr(s, X = x) = Pr(s) \tag{3}$$

Definition 3.6 ***(Most Responsible Cause)*** *A cause C is a most responsible cause for the violation of $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, if $dR(C)Pr(C) \geq dR(C')Pr(C')$ for any cause C'.*

3.2 Blame for $MIPCX_d(s_0 \models \phi)$

MDPs are non-deterministic Markov models through the possible actions that can be taken at each state, where each action enables a set of transitions. In probabilistic guarded command language of probabilistic model checker like PRISM, transitions represent updates on the variables with probabilities assigned to them. Therefore, for complete analysis of probabilistic counterexamples for MDPs, we have also to deal with actions. As a result, we should ask: how does an action α contribute to the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$? which leads to the question, which action should be more blamed for exceeding the probability threshold p. Hence, the designer would not need just to be aware of probable false assignments, but he also needs to know how such actions are involved. Given this information, the designer could fix the guard commands in way he gets the acceptable outcome. We should mention that blame considers mainly whether an action α is to blame for an outcome φ under uncertainty [12].

Consider a state s in $MIPCX_d(s_0 \models \phi)$. For each action $\alpha \in A(s)$, we denote by $Suc(s, \alpha)$ the set of α-successors of s, where α-successor is a state $s' \in S$ such that $P(s, \alpha, s') > 0$. We associate for each transition from s to s', where $s' \in Suc(s, \alpha)$ a probability as follow

$$P_{\alpha}^{MIPCX_d(s_0 \models \phi)}(s, s') = \sum_{\sigma \in MIPCX_d(s_0 \models \phi) | (s, \alpha, s') \in \sigma} P(\sigma) \tag{4}$$

It is evident that not every transition enabled by an action α has the same presence in the paths forming the counterexample. So this probability measures simply the contribution of a transition to the probability of the counterexample by summing the probabilities of the paths in which it is included.

Proposition 3.1 *Consider a transition (s,α,s') in $MIPCX_d(s_0 \models \phi)$, $Max(P_{\alpha}^{MIPCX_d(s_0\models\phi)}(s,s')) = Pr(s)$ iff s' is the unique successor of s.*

Proof Pr(s) represents the sum of probabilities of paths in which s is included, $P_{\alpha}^{MIPCX_d(s_0\models\phi)}(s,s'))$ represents the probabilities of the paths in which the transition is included, so it is sufficient to prove that both of them are included in the same set of paths, if only if s' is the unique successor of s. Let s' and s'' two successors of s, if s is included in N paths, the transition (s,s'') will be included at least in one path from this set, and thus s' will be included at most in $N-1$ of paths. Hence, $Max(P_{\alpha}^{MIPCX_d(s_0\models\phi)}(s,s'))$ will not equal to $Pr(s)$ if there is another successor s'' of s.

We should mention that every transition in a probabilistic program describes how the values of the variables evolve over time, and thus considering the transitions and their contribution to the error is very important as debugging information, which makes it a required measure for the definition of blame.

Definition 3.7 ***(Blame)*** *Consider a counterexample $MIPCX_d(s_0 \models \phi)$ for a probabilistic formula $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, a state s, an action $\alpha \in A(s)$ and a set of successors $Suc(s,\alpha)$. The degree of blame for an action α for the violation of $\phi = \boldsymbol{P}_{\leq p}(\varphi)$ in a state s denoted $dB(s,\alpha,\phi)$ is*

$$\sum_{s' \in Suc(s,\alpha)} dR(s',X=x,\phi)P_{\alpha}^{MIPCX_d(s_0\models\phi)}(s,s') \tag{5}$$

That is, the degree of blame dB informs us about the contribution of an action α in a state s to the violation of the probabilistic formula $\phi = \mathbf{P}_{\leq p}(\varphi)$. While responsibility stands as a criticality measure for actual causes given well-defined states for the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$, dB describes how an action should be blamed for this violation through considering the probabilities assigned to the transitions leading to these states. So that, the action more blamed for the violation will be the one more contributing to the probability of $MIPCX_d(s_0 \models \phi)$ and leading to more critical states.

Definition 3.8 ***(Most Blame)*** *An action $\alpha \in A(s)$ has most blame for the violation of PCTL property $\phi = \boldsymbol{P}_{\leq p}(\varphi)$, if $dB(s,\alpha,\phi) \geq dB(s',\alpha',\phi)$ for any s' and any $\alpha' \in A(s')$.*

Proposition 3.2 *Let $\alpha \in A(s)$, $Max(dB(s,\alpha,\phi)) = P(MIPCX_d(s_0 \models \phi))$ iff for every $\sigma \in MIPCX_d(s_0 \models \phi)$ there exists $(s,\alpha,s') \in \sigma$, such that s' is critical with respect to such variable X.*

That is, the maximum degree of blame of an action α in a state s is equal to the probability of the counterexample, if only if for every path σ of the counterexample, there exists a transition from s to a state s' under this action, and s' is critical with respect to such variable X.

Theorem 3.1 *Let $\alpha \in A(s)$ and $s' \in Suc(s,\alpha)$, $(s',X=x)$ is most responsible cause $\nRightarrow$ α has the most blame.*

Proof. Let s_1 and s_2 be two states, α_1 and α_2 two actions enabled at these states respectively. Let α_1 leads to a most responsible cause C and we assume that $dB(s_1,\alpha_1,\phi) \geq dB(s_2,\alpha_2,\phi)$. Then, for every cause C', α_2 leads to, $dR(C')Pr(C') < dR(C)Pr(C)$. Let C be the unique cause α_1 it leads to, let $Pr(s_2) > Pr(s_1)$ and let all the causes that α_1 and α_2 lead to are critical. With respect to Proposition 3.1, the probability of the transition leading to C' is $Pr(s_1)$, and thus with respect to the definition of blame, $dB(s_1,\alpha_1,\phi)$ will be at most $Pr(s_1)$, which is less than $Pr(s_2)$, and since the causes that α_2 leads to are critical, it contradicts α_1 being the action with the most blame.

3.3 Algorithm for Generating Diagnoses

This algorithm performs on counterexamples generated by DiPro [4]. DiPro is a tool used for generating counterexamples from DTMCs, CTMCs and MDPs models, and can be jointly used with the probabilistic model checkers PRISM [22] and MRMC [25].

Algorithm 1 . Generate Diagnoses

1: **Inputs:** The counterexample $MIPCX_d(s_0 \models \phi)$, The probabilistic formula $\phi = P_{\leq p}(\varphi)$ where φ is of the form $\phi_1 \mathbf{U} \phi_2$ or $(\phi_1 \mathbf{U}^{\leq n} \phi_2)$
2: **Outputs:** Causes with responsibilities and probabilities
3: Actions with blame
4:
5: Causes **:=** $\emptyset$
6: Actions **:=** $\emptyset$
7: **for each** state s in $MIPCX_d(s_0 \models \phi)$ **do**
8: W:= 0
9: **if** s is the last state in a path σ **then**
10: Causes with dR**:=** Causes $\cup$ FINDCAUSES(s, ϕ_2, W)
11: Pr(Causes)=$\sum_{\sigma \in MIPCX_d(s_0 \models \phi) | s \in \sigma} P(\sigma)$
12: **else**
13: Causes with dR**:=** Causes $\cup$ FINDCAUSES(s, ϕ_1, W)
14: Pr(Causes)=$\sum_{\sigma \in MIPCX_d(s_0 \models \phi) | s \in \sigma} P(\sigma)$
15: $P_\alpha^{MIPCX_d(s_0 \models \phi)}(s, s') = \sum_{\sigma \in MIPCX_d(s_0 \models \phi) | (s, \alpha, s') \in \sigma} P(\sigma)$
16: **end if**
17: **end for**
18: **for each** s in $MIPCX_d(s_0 \models \phi)$ **do**
19: Actions with dB**:=** Order(Actions $\cup$ $\alpha \in A(s)$, $dB(s, \alpha, \phi_1) = \sum_{s' \in Suc(s,\alpha)} dR(s', X =$
20: $x, \phi_1) P_\alpha^{MIPCX_d(s_0 \models \phi)}(s, s')$)
21: **end for**
22: **O**UTPUTDIAGNOSES(Causes with dR and Pr, Actions with dB)

The algorithm 1 (Generate Diagnoses) explores the counterexample and computes the causes with their responsibilities and probabilities with respect to each state s, and computes the degree of blame for each action enabled at this state. The condition put on last state follows Lemma 2.1. Algorithm 1 uses the function FindCauses that takes a state and a state formula as input as well as the variable W, and returns recursively the set of causes and their responsibilities, where the variable W is used to compute responsibility [14].

Computing the set of causes exactly in binary causal models is NP-complete in general[15]. However, [16] shows that causes can be computed in polynomial time if the causal graph forms a directed causal tree. Our algorithm is linear in the size of $MIPCX_d(s_0 \models \phi)$ and the size of φ as it has been presented in [14], because reconfiguring the algorithm to compute the degree of blame by adding the loop in line 18 does not bring additional complexity since it is directly based on measures already computed, which are the degree of responsibility of each cause and the probability of each transition $P_\alpha^{MIPCX_d(s_0 \models \phi)}(s, s')$. Computing $P_\alpha^{MIPCX_d(s_0 \models \phi)}(s, s')$ (line 15) is performed under the same loop for computing the probabilities of causes (line 14).

```
function FindCauses(s, ψ, W)
2:  if ψ is of the form a where a ∈ AP and a ∈ L(s) then
        return (⟨s,a⟩, dR(⟨s,a⟩) = 1/(W+1))
4:  end if
    if ψ is of the form ¬a where a ∈ APand a ∉ L(s) then
6:      return (⟨s,¬a⟩, dR(⟨s,¬a⟩) = 1/(W+1))
    end if
8:  if ψ is of the form ψ1 ∧ ψ2 then
        return FindCauses(s, ψ1, W) ∪
10:     FindCauses(s, ψ2, W)
    end if
12: if ψ is of the form ψ1 ∨ ψ2 then
        if s ⊨ ψ1 and s ⊨ ψ2 then
14:         return FindCauses(s, ψ1, W++) ∪
            FindCauses(s, ψ2, W++)
16:         if s ⊨ ψ1 ∧ s ⊭ ψ2 then
                return FindCauses(s, ψ1, W)
18:         end if
            if s ⊭ ψ1 ∧ s ⊨ ψ2 then
20:             return FindCauses(s, ψ2, W)
            end if
22:     end if
    end if
24: end function
```

```
function OutputDiagnoses(Causes with dR and Pr, Actions with dB)
    for each actionα ∈ A(s) from Actions do
3:      Output (α)
        for each a successor s′ ∈ Suc(s, α) do
            Output Causes Ordered with respect to dR × Pr
6:          Output (s, s′)
        end for
    end for
9: end function
```

At the end we present the function OUTPUTDIAGNOSES that shows the way of presenting the diagnoses to te user. It gets the actions ordered with respect to dB, and the causes with dR and Pr and outputs the diagnoses in order. We see that the output of the diagnoses starts with the action with the most blame, and among all the causes it leads to, we begin by the most responsible cause, by indicating also the transition leading to this cause. Presenting the transition enabled under this action is very important as a diagnostic information, because transitions in typical probabilistic program describe how the values of the variables evolve over time.

Example 2
Let us apply this algorithm on the counterexample $CX3$ from the previous example. The user gets the action α_0 first, since $dB(s_0,\alpha_0,\phi) = 0.6$ is the highest, it is equal to the probability of the counterexample. From $Suc(s_0,\alpha)$, the user gets first the cause (s_2,b) because it is the most responsible by computing the measure $Pr(s_2,b) \times dR(s_2,b) = 0.58 > Pr(s_1,a) \times dR(s_1,a,\phi) = 0.41$. The following action the user gets is α_2 with degree of blame $dB(s_2,\alpha_2,\phi) = 0.5 \times (0.15) + 1 \times (0.2) = 0.275$ with the causes it led to $\{(s_3,c),(s_3,d)\}$ and $\{(s_4,a)\}$,$\{(s_4,b)\}$ respectively, then α_1 with $dB(s_1,\alpha_1) = 0.25$ with the cause it led to $\{(s_7,c),(s_7,d)\}$, and finally the action α_4 with $dB(s_4,\alpha_4,\phi) = 0.15$ with the cause it led to $\{(s_5,c),(s_5,d)\}$.

4 Experiments

We have implemented the above method in Java. We used two benchmark case studies to evaluate our method, the Zeroconf protocol [2] and CSMA/CD protocol [1]. All the experiments were carried out on windows XP with Intel Pentium CPU 3.2 GHz speed And 2 GB of memory. We sue DiPro for generating the counterexamples. DiPro employs many algorithms for generating counterexamples, among these algorithms we use the K* algorithm [7]. Our method takes the counterexample generated from DiPro and the property to be verified as input, and outputs the diagnoses.

4.1 Zeroconf

The protocol is modeled in PRISM as an MDP, where the number of abstract hosts is denoted by N, the number of probes to send is denoted by K, and the probability of message loss is denoted by $loss$. Each station has a single-message buffer and is cyclically attended by the server. The buffer could store the messages that it wants to send, in such cases, messages are not relevant after reconfiguring, and thus keeping these messages can slow down the network and making hosts reconfigure when they do not need to. We therefore considered two different versions of the network: one where the host does not do anything about these messages (No_Reset) and the another where the host deletes these messages when it decides to choose a new IP address (Reset).

We chose the property that measures the probability of not choosing a fresh address by time T. This property is given in PRISM as follows:

$$Pmax = ?[!(l = 4 \wedge ip = 2)U(t > T)]$$

We test this property using PRISM for both types of network (Reset) and (No_Reset) for the following values (T = 10; N = 1000;loss = 0.1 and K could vary from 4 to 8). For these values, PRISM renders a probability greater than 0.5. As a result, we chose the value 0.5 as a threshold. The property can be rewritten as follows:

$$P \leq 0.5[!(l = 4 \wedge ip = 2)U(t > T)]$$

Table 1: PRISM results for Zeroconf

Reset	K	states	transitions
true	4	9683	15727
	6	7743	11401
	8	7743	11401
false	4	59076	121265
	6	58937	120525
	8	58937	120525

The PRISM results are shown in Table 1. We notice that the size of the models is very huge with (No_Reset), comparing to (Reset). Despite that, DiPro renders the same counterexample for all these different configurations with the same execution time. For generating the counterexample, DiPro Explored 24 traces resulting in 121 vertices and 150 edges in 5 seconds for all the configurations. Finally, the counterexample rendered by DiPro consists just of 8 diagnostic paths.

We pass this counterexample to our algorithm for generating the diagnoses. Our algorithm takes less than 1 second as execution time. The causes generated for this property are as follows: for the right sub-formula, the cause generated is singleton $C0 = (t > T)$. For the left sub-formula, the set of causes for not choosing a fresh address: $C1 = !(l = 4)$ and $C2 = !(ip = 2)$. Notice that we are facing disjunctive scenario here, which means that either $C1$ (address not in use) is the actual cause or $C2$ (not fresh address) or both of them. Our results show that except the initial state where $ip = 1$ (IP address of an abstract host which the concrete host is currently trying to configure), the actual cause for not choosing fresh address within 10 time units is $C1$, which means that we reach states in which there is fresh ip which the concrete host is currently trying to configure but without being used. The number of states from 8 diagnostic paths in which we found theses causes are estimated to be 59 states, this is much less than the states of the model (9683).

Concerning the actions and their blame, in the model we have two main actions causing the non-determinism in such states, which are: 'Reconfigure' denoted *rec* and 'defend' which is performed by sending an ARP packet and thus this action is denoted in the model by *send*. For the counterexample generated given the previous property, our results show that there is no state in which the host reconfigures, which means that the only action we are dealing with is *send*. As a result, we compute the dB of *send* action in such states. We found that it has the same degree of blame ($dB = 0.25$) in each state it is enabled.

4.2 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection)

CSMA/CD is a protocol for carrier transmission access in Ethernet networks that avoids collision (minimizing simultaneous use of the channel) when Network Interface Card (NIC) tries to send its packet. The protocol is modeled as a probabilistic timed automata (PTA) [1] and can be reduced to an MDP in order to be analyzed against probabilistic properties by PRISM. The model in PRISM consists of three main components or modules, the two senders namely station 1 and station 2 respectively and the third component is the bus or the medium. The protocol functionality is as follows: if a station has a data to send, it listens first to the medium, in case it is free, the stations send the data, otherwise (bus is busy), it repeats the process after random amount of time. If there is a collision the station attempts to retransmit the packet where the scheduling of the retransmission is determined by a truncated binary exponential backoff process. The complete model is available in the PRISM benchmark suite under MDPs section [3].

We chose the property that estimates the maximum probability of all stations sending successfully

Table 2: PRISM Results

K	States	Transitions
2	1083	1282
4	7958	10594

Table 3: DiPro Results

K	States	Transitions	Time Construction (Sec)	Diagnostic Paths
2	1037	1276	6 sec	134
4	4222	5201	16 sec	324

before a collision with max backoff. This property is given as follows:

$$Pmax = ?[!\text{"}collision_max_backoff\text{"}U\text{"}all_delivered\text{"}]$$

We tested this property using PRISM for the following values: N=2 and K=2, N=2 and K=4 respectively, where N represents the number of stations and K represents the exponential backoff limit. For these values, PRISM renders a probability greater than 0.7. As a result, we chose the value 0.7 as a threshold. The property can be rewritten as follows:

$$P \leq 0.7[!\text{"}collision_max_backoff\text{"}U\text{"}all_delivered\text{"}]$$

Where $collision_max_backoff$ and $all_delivered$ are defined as follows: $collision_max_backoff = (cd1 = K\&s1 = 1\&b = 2)|(cd2 = K\&s2 = 1\&b = 2)$, $all_delivered = (s1 = 4\&s2 = 4)$. The variables $cd1$ and $cd2$ refer to collision counters for both stations where K represents the backoff limit , $s1$ and $s2$ refer to the state of the stations where $s1 = 1, s2 = 1$ indicate that the stations are transmitting data, and finally b refers to the state of the bus where $b = 2$ indicates that there is a collision.

Table 2 shows the size of the model by PRISM. Table 3 shows the states and transition explored while searching for the counterexample and the time required for its construction by DiPro. We notice that DiPro nearly explored all the model to generate the counterexample for K=2, whereas for K=4, DiPro explores nearly half of the model. For k=2, the counterexample generated consists of 134 diagnostic paths, and for k=4 the counterexample generated consists of 324 diagnostic paths. We pass both counterexamples to our algorithm for generating the diagnoses.

Table 4 shows the execution results of our algorithm. The second column represents the number of causes generated from the counterexample with respect to the states, while the third column shows the number of classes of causes probabilities. The results show that the time taken for computing the causes is less than the time taken for generating the counterexample.

The causes generated for this property are as follows: For the right sub-formula, the cause generated is a conjunct $C0 = (s1 = 4\&s2 = 4)$. For the left sub-formula, the set of causes for not facing a collision with max backoff are: $C1 = \neg(cd1 = K)$, $C2 = \neg(s1 = 1)$, $C3 = \neg(b = 2)$,$C4 = \neg(cd2 = K)$ $C5 = \neg(s2 = 1)$. For both values of K, our results show that there are causes that share the same probability, as we mentioned before that most responsible cause may not be unique. In addition, for both values of

Table 4: Execution results of our algorithm

K	Causes	Causes Probabilities	Execution Time (Sec)
2	616	98	3 sec
4	923	47	8 sec

K, the most responsible causes are the same. Among all the causes, we found that the most responsible causes for not facing collision are $C1$ and $C4$, where the states in which these causes are found have the highest probability.

Concerning the actions and their blame, we found that $send1$ and $send2$ have the most blame, where $send1$ and $send2$ lead to the most responsible causes $C1$ and $C4$. For $send1$, the first transition that it enables is represented by the valuations $b = 1$ (bus active) and $s1 = 1$ (station 1 is transmitting). Given these information we will be able to identify the commands that contributed the most to the violation, which are $[send1](b = 0) -> (b' = 1)$ (line 35) in the bus module from a side, and from the side of station, the command is $[send1](s1 = 0) -> (s1' = 1) \& (x1' = 0)$ (line 82). For $send2$, the first transition that it enables is represented by the valuations $b = 2$ (bus busy - collision) and $s2 = 1$ (station 2 is transmitting). Given these information we will be able to identify the commands, which are $[send2](b = 1|b = 2) \& (y2 < sigma) -> (b' = 2)$ (line 40) in the bus module from a side, and from the side of station, the command is $[send2](s2 = 0) -> (s2' = 1) \& (x2' = 0)$ (line 82). Then, based on the second transition enabled by $send2$, which is represented by the valuations $b = 1$ (bus active) and $s2 = 1$ (station 2 is transmitting) and leads to a critical state, the commands to check are $[send2](b = 0) -> (b' = 1)$ (line 36) in the bus module, and $[send2](s2 = 3) \& (x2 = slot) \& (bc2 = 0) -> (s2' = 1) \& (x2' = 0)$ (line 99) in the station module respectively. Finally, based on the second transition enabled by $send1$, which is represented by the valuations $b = 2$ (bus busy - collision) and $s1 = 1$ (station 1 is transmitting), the commands to check are: $[send1](b = 1|b = 2) \& (y1 < sigma) -> (b' = 2)$ (line 39)in the bus module, and $[send1](s1 = 3) \& (x1 = slot) \& (bc1 = 0) -> (s1' = 1) \& (x1' = 0)$ (line 99) in the station module respectively. The other states in which $C1$ and $C4$ are not the causes, we find that $C3$ is the most responsible cause. So by defining the transition leading to it, and under which action is enabled, we could also map to the commands related and analyze the rest of the model driven by the diagnoses. So as we see, given the diagnoses generated by our algorithm, it would be easy to go-back to the model, which is given in PRISM guarded command language, and identify the commands that contributed the most to the violation of the probability threshold. By performing some changes on these commands in order, the probability as estimated by PRISM goes below the probability threshold, and thus the probabilistic property will be satisfied.

In general, the results presented here report that our method has a good execution time and thus it can be adapted in practice. Comparing to the execution time taken for generating the counterexample, the execution time of our method is remarkably lower. Concerning the number of diagnoses generated, our method outputs low number of causes comparing to the size of the model, in addition our method outputs the most responsible causes first to the user, which could help the user to find the source of the error rapidly without reading all the causes generated. Along the results, we found that many causes could share the same probability, which means that many causes are found in the same set of paths of the counterexample, this could mean that there is such dependence between variables where the values are changing together. This information is very important for debugging, because it could help user even to locate the line responsible for error in the model itself given the action leading to it. Concerning actions we notice that such actions from the model might be ignored since they do not exist under the scheduler resolving the non-determinism, for instance for Zerconf protocol, we found the only action to be analyzed is *send*, the action *reconfigure* is not concerned, and that means that the search space is limited. It is evident that relaying on small number of actions also facilitates the debugging. Concerning the structure of the probabilistic property to be analyzed, our method has more importance when we face a disjunctive scenario, because in disjunctive scenario we can not be sure about the variables causing the violation, and here where responsibility measure plays the major role. Finally, the commands as identified in the PRISM modules with respect to the diagnoses generated do not necessarily follow the same order in the

module, for instance the command $[send2](b=1|b=2)\&(y2<sigma)->(b'=2)$ (line 40) has been reported before the command $[send2](b=0)->(b'=1)$ (line 36). We also notice that we were able to deal with the parallel composition of the modules, for instance $[send1](b=0)->(b'=1)$ in the bus module and $[send1](s1=0)->(s1'=1)\&(x1'=0)$ in the station module.

5 Conclusion and Future Works

In this paper we have shown how the notions of causality, responsibility and blame can be useful in the context of probabilistic counterexamples of MDPs. Due to the probabilistic nature of the causality model, we introduced the definition of most responsible cause and the action with most blame. We showed that delivering the causes/actions with respect to their responsibilities/blame stands as a good debugging method that guides the user through large counterexamples, and thus it could help us to identify the commands responsible for the violation of the probabilistic formula.

As future works, we plan to deliver a debugging tool that generates the diagnoses graphically for all kinds of Markov models. Furthermore, we aim to integrate our method in the model checking process itself, in order to locate the commands in PRISM code directly without depending on the counterexample generated.

References

[1] : *CSMA/CD protocol.* `http://www.prismmodelchecker.org/casestudies/csma.php`.

[2] : *IPv4 Zeroconf protocol.*`http://www.prismmodelchecker.org/casestudies/zeroconf.php`.

[3] : *PRISM benchmark suite - Models.* `http://www.prismmodelchecker.org/benchmarks/models.php#mdps`.

[4] H. Aljazzar, F. Leitner-Fischer, S. Leue & D. Simeonov (2011): *DiPro - A Tool for Probabilistic Counterexample Generation.* In: Proceedings of the 18th International SPIN Workshop, LNCS 6823, Springer, Berlin, Heidelberg, pp. 183–187, doi:10.1007/978-3-642-22306-8_13.

[5] H. Aljazzar & S. Leue (2009): *Generation of counterexamples for model checking of Markov decision processes.* In: Proceedings of the International Conference on Quantitative Evaluation of Systems (QEST), pp. 197–206, doi:10.1109/QEST.2009.10.

[6] H. Aljazzar & S. Leue (2010): *Directed explicit state-space search in the generation of counterexamples for stochastic model checking.* IEEE Trans. on Software Engineering 36(1), pp. 37–60, doi:10.1109/TSE.2009.57.

[7] Husain Aljazzar & Stefan Leue (2011): *K*: A heuristic search algorithm for finding the k shortest paths.* Artificial Intelligence 175(18), pp. 2129 – 2154, doi:10.1016/j.artint.2011.07.003.

[8] A. Aziz, K. Sanwal, V. Singhal & R. Brayton (2000): *Model-checking continuous-time Markov chains.* ACM Transactions on Computational Logic 1(1), pp. 162–170, doi:10.1145/343369.343402.

[9] C. Baier, B. Haverkort, H. Hermanns & J.-P Katoen (2003): *Model checking algorithms for continuous-time Markov chains.* IEEE Transactions on Software Engineering 29(7), pp. 524–541, doi:10.1109/TSE.2003.1205180.

[10] T. Ball, M. Naik & S.K. Rajamani (2003): *From symptom to cause: Localizing errors in counterexample traces.* In: Proceedings of ACM Symposium on the Principles of Programming Languages, pp. 97–105, doi:10.1145/640128.604140.

[11] A. Beer, S. Heidinger, U. Kuhne, F. Leitner-Fischer & S. Leue (2015): *Symbolic Causality Checking Using Bounded Model Checking.* In: Proceedings of SPIN 2015, LNCS 9232, Springer-Verlag, Berlin, Heidelberg, pp. 203–221, doi:10.1007/978-3-319-23404-5_14.

[12] H. Chockler & J. Y. Halpern (2004): *Responsibility and blame: a structural model approach*. Journal of Artificial Intelligence Research (JAIR) 22(1), pp. 93–115, doi:10.1613/jair.1391.

[13] H. Debbi (2014): *Diagnosis of Probabilistic Models using Causality and Regression*. In: Proceedings of the 8th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2014), pp. 33–44.

[14] H. Debbi & M. Bourahla (2013): *Causal Analysis of Probabilistic Counterexamples*. In: Proceedings of the Eleventh ACM-IEEE International Conference on Formal Methods and Models for Codesign (Memocode), pp. 77–86.

[15] T. Eiter & T. Lukasiewicz (2002): *Complexity results for structure-based causality*. Artificial Intelligence 142(1), p. 53, doi:10.1016/S0004-3702(02)00271-0.

[16] T. Eiter & T. Lukasiewicz (2006): *Causes and explanations in the structural-model approach: Tractable cases*. Artificial Intelligence 170(6–7), pp. 542–580, doi:10.1016/j.artint.2005.12.003.

[17] F. Fischer & S. Leue (2013): *Causality Checking for Complex System Models*. In: Proceedings of Verification, Model Checking, and Abstract Interpretation (VMCAI), LNCS 7737, Springer, Berlin, Heidelberg, pp. 248–276, doi:10.1007/978-3-642-35873-9_16.

[18] A. Groce (2004): *Error explanation with distance metrics*. In: Proceedings of Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS), pp. 108–122, doi:10.1007/s10009-005-0202-0.

[19] J. Halpern & J. Pearl (2001): *Causes and explanations: A structural-model approach part I: Causes*. In: Proceedings of the 17th UAI, pp. 194–202, doi:10.1093/bjps/axi147.

[20] T. Han & J.P. Katoen (2009): *Counterexamples Generation in probabilistic model checking*. IEEE Trans. on Software Engineering 35(2), pp. 72–86, doi:10.1109/TSE.2009.5.

[21] H. Hansson & B. Jonsson (1994): *Logic for reasoning about time and reliability*. Formal aspects of Computing 6(5), pp. 512–535, doi:10.1007/BF01211866.

[22] A. Hinton, M. Kwiatkowska, G. Norman & D. Parker (2006): *PRISM: A tool for automatic verification of probabilistic systems*. In: Proceedings of TACAS, LNCS 3920, Springer, Berlin, Heidelberg, pp. 441–444, doi:10.1007/11691372_29.

[23] I.Beer, S. Ben-David, H. Chockler, A. Orni & R. Treer (2012): *Explaining counterexamples using causality*. Formal Methods Systems Design 40(1), pp. 20–40, doi:10.1007/s10703-011-0132-2.

[24] N. Jansen, E. Abraham, M. Volk, R. Wilmer, J.P Katoen & B. Becker (2012): *Proceedings of The COMICS Tool - Computing Minimal Counterexamples for DTMCs*. In: Proceedings of ATVA, LNCS 7561, Springer, Berlin, Heidelberg, pp. 249–253, doi:10.1007/978-3-642-33386-6_27.

[25] J.-P. Katoen, M. Khattri & I. S. Zapreev (2005): *A Markov Reward Model Checker*. In: Proceedings of QEST, pp. 243–244, doi:10.1109/QEST.2005.2.

[26] F. Leitner-Fischer & S. Leue (2013): *On the Synergy of Probabilistic Causality Computation and Causality Checking*. In: Proceedings of SPIN 2013, LNCS 7976, Springer-Verlag, Berlin, Heidelberg, pp. 246–263, doi:10.1007/978-3-642-39176-7_16.

[27] F. Leitner-Fischer & S. Leue (2013): *Probabilistic fault tree synthesis using causality computation*. International Journal of Critical Computer-Based Systems 4(2), pp. 119–143, doi:10.1504/IJCCBS.2013.056492.

[28] R. Wimmer, N. Jansen, E. Abraham, B. Becker & J.P. Katoen (2012): *Minimal Critical Subsystems for Discrete-Time Markov Models*. In: Proceedings of TACAS, LNCS 7214, Springer, Berlin, Heidelberg, pp. 299–314, doi:10.1007/978-3-642-28756-5_21.

[29] R. Wimmer, N. Jansen & A. Vorpahl (2013): *High-Level Counterexamples for Probabilistic Automata*. In: Proceedings of Quantitative Evaluation of Systems (QEST), LNCS 8054, Springer, Berlin, Heidelberg, pp. 39–54, doi:10.1007/978-3-642-40196-1_4.