Diagnosis of Probabilistic Models using Causality and Regression

Hichem Debbi
Department of Computer Science
University of M'sila
M'sila
Algeria
hichem.debbi@gmail.com

The counterexample in probabilistic model checking (PMC) is a set of paths in which a path formula holds, and their accumulated probability violates the probability bound. However, understanding the counterexample is not an easy task. In this paper we address the complementary task of counterexample generation, which is the counterexample analysis. We propose an aided-diagnostic method for probabilistic counterexamples based on the notions of causality and regression analysis. Given a counterexample for a Probabilistic CTL (PCTL)/Continuous Stochastic Logic (CSL) formula that does not hold over Discrete Time Markov Chain (DTMC)/Continuous Time Markov Logic (CTMC) model, this method generates the causes of the violation, and describes their contribution to the error in the form of a regression model.

Probabilistic Model Checking (PMC), Probabilistic Computation Tree Logic (PCTL), Probabilistic Counterexample, Causality, Structural Equation Modeling (SEM), Regression Analysis.

1. INTRODUCTION

Probabilistic Model Checking (PMC) has appeared as an extension of model checking for modelling and 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 Chains (DTMC), Continuous-Time Markov Chains (CTMC) or Markov Decision Processes (MDP), and verified against properties specified in Probabilistic Computation Tree Logic (PCTL) Hansson and Jonsson (1994) or Continuous Stochastic Logic (CSL) (Aziz et al (2000), Baier et al (2003)).

Unlike the previous methods proposed for conventional model checking that generate the counterexample as a single path ending with bad state 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 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 Aljazzar and Leue (2009, 2010, 2011); Han et al. (2009). In PMC, this task is more challenging.

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 understand the failure (Ball et al. (2003), Zeller et al. (2002), Groce et al. (2003), Groce et al. (2006), Wang et al. (2006), Kumazawa and Tamai (2011), Beer et al. (2011)). As it was done in conventional model checking, addressing the error explanation in the PMC is highly required, especially that probabilistic counterexample is a set of multiple paths instead of single path, and it is probabilistic.

In this paper, we address the diagnosis of probabilistic counterexamples. To this end, we adopt the definition of causality based on counterfactuals by Halpern and Pearl (Halpern and Pearl (2001)) to reason formally about the causes, and then transform the causality model into regression model using Structural Equation Modelling (SEM) in order to quantify the contribution of these causes to the error . SEM is a comprehensive analytical method used for testing and estimating causal relationships between variables embedded in theoretical causal model (Wright and Sewall (1921)). Regression analysis, path analysis and factor analysis are all special cases of SEM.

This paper contains two complementary contributions. The first defines a formal causality model for the counterexample, where we give the definition of an actual cause for the violation of PCTL/CSL formula $\phi = \mathbf{P}_{\leq p}(\varphi)$. The second complements the first by generating a regression model from the causality model, which quantifies the effect of each cause on the violation of the PCTL/CSL formula. In this paper, we will focus on upper bounded properties. Lower bounded properties can be transformed easily to upper bounded properties (Aljazzar and Leue (2010), Han et al. (2009)). Our approach does not ignore the previous approaches of generating probabilistic counterexamples, but instead it is based on them. Our approach for error explanation is based directly on the most indicative counterexamples(Aljazzar and Leue (2010), Han et al. (2009)).

The rest of this paper is organized as follows. In Section 2, we present some preliminaries and definitions. The Probabilistic Computation Tree Logic (PCTL) and probabilistic counterexamples are presented in this section. In section 3, we give the definition of Causes in probabilistic counterexamples with the formal causality model, and then we show how to generate the regression model from it. Following that, we introduce an algorithm for generating the causes of violation of PCTL/CSL properties and the related regression model. Section 4 describes an experimental study. Section 5 presents the related works. At the end, we present conclusion and future works.

2. PRELIMINARIES AND DEFINITIONS

A Discrete-Time Markov Chain (DTMC)is a tuple $D = (S, s_{init}, P, L)$, such that S is a finite set of states, $s_{init} \in S$ the initial state,$P : S \times S \to [0,1]$ represents the transition probability matrix, $L : S \to 2^{AP}$ is a labelling function that assigns to each state $s \in S$ the set $L(s)$ of atomic propositions. An infinite path σ is a sequence of states $s_0 s_1 s_2 ...$, where $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A finite path is finite prefix of an infinite path. We define a set of paths starting from a state s_0 by $Paths(s_0)$. The underlying σ-algebra is formed by the cylinder sets which are induced by finite paths in $Paths(s_0)$. The probability of this cylinder set is:

$$P(\sigma \in Paths(s_0) | s_0 s_1 ... s_n \text{is a prefix of } \sigma) = \prod_{i \leq 0 < n} P(s_i, s_{i+1})$$

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. We use the PCTL for defining quantitative properties of DTMCs. PCTL state formulas are formed 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, **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 $\phi_1 and \phi_2$ are state formulas and $n \in N$. As in CTL, the temporal operators (**U** for strong until, **W** for weak (unless) until and their bounded variants) are required to be immediately preceded by the operator **P**. The PCTL formula is a state formula, where path formulas only occur inside the operator **P**. The operator **P** can be seen as a quantification operator for both the operators $\forall$ (universal quantification) and $\exists$ (existential quantification), since the properties are representing quantitative requirements.

The semantics of a PCTL formula over a state s (or a path σ) in a DTMC model $D = (S, s_{init}, P, L)$ can be defined by a satisfaction relation denoted by $\models$. 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.

The semantics of PCTL state formulas for DTMC is defined as follows:

$s \models true \Leftrightarrow true$
$s \models a \Leftrightarrow a \in L(s)$
$s \models \neg\phi \Leftrightarrow s \not\models \phi$
$s \models \phi_1 \wedge \phi_2 \Leftrightarrow s \models \phi_1 \wedge s \models \phi_2$
$s \models \mathbf{P}_{\sim p}(\varphi) \Leftrightarrow P(s \models \varphi) \sim p$

Given a path $\sigma = s_0 s_1 ...$ in D and an integer $j \geq 0$, where $\sigma[j] = s_j$, The semantics of PCTL path formulas for DTMC is defined as for CTL as follows:

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

For specifying properties of CTMC, we use The Continuous Stochastic Logic (CSL). CSL has the same syntax and semantics as PCTL, except that in CSL, the time bound in bounded until formula can be presented of an interval of non-negative reals. Before verifying CSL properties over CTMC, the CTMC has to be transformed to its embedded DTMC. Therefore, further description of CTMC model checking is beyond the scope of this paper. We refer to [2, 3] for further details. In the rest of paper, we refer to probabilistic formula PCTL/CSL as $\phi = \mathbf{P}_{\leq p}(\varphi)$.

2.2. Probabilistic Counterexamples

The probabilistic counterexample is generated when a PCTL/CSL property is not satisfied. The probabilistic property $\phi = \mathbf{P}_{\leq p}(\varphi)$ is refuted when the probability mass of the paths satisfying φ exceeds the bound p. Therefore, a probabilistic counterexample for the property ϕ is formed by a set of paths starting at state s and satisfying the path formula φ. We denote these paths by $Paths(s_0 \models \phi)$. The counterexample can be formed of set of finite paths where each path $\sigma = s_0 s_1 ... s_n$ is a prefix of an infinite path from $Paths(s_0 \models \phi)$ satisfying the formula φ.We denote these paths by $FinitePaths(s_0 \models \phi)$. These finite paths are also called diagnostic paths (Aljazzar and Leue (2010), Aljazzar and Leue (2011)).

It is clear that we can get a set of probabilistic counterexamples, noted $PCX(s_0 \models \phi)$, which is a set of any combination from $FinitePaths(s_0 \models \phi)$ that their probability mass exceeds the bound p. Among all these probabilistic counterexamples, we are interested by the most indicative one. The most indicative counterexample is minimal counterexample (has the least number of paths from $FinitePaths(s_0 \models \phi)$) and its probability mass is the highest among all other minimal counterexamples. We denote the most indicative probabilistic counterexample by $MIPCX(s_0 \models \phi)$. We should note that the most indicative probabilistic counterexample may not be unique.

For the counterexample to have high probability, it should consist of paths that carry high probabilities from $FinitePaths(s_0 \models \phi)$. The path σ having the highest probability over all these paths is called strongest path and is defined as follows: for every path $\sigma' \in FinitePaths(s_0 \models \phi) : P(\sigma) \geq P(\sigma')$. The strongest path also may not be unique.

Lemma 2.1 *A most indicative probabilistic counterexample $MIPCX(s_0 \models \phi)$ contains at least one strongest path $\sigma \in FinitePaths(s_0 \models \phi)$.*

Lemma 2.2 *For every path $\sigma \in MIPCX(s_0 \models \phi)$ on which the property $\phi_1 \mathbf{U} \phi_2$ $(\phi_1 \mathbf{U}^{\leq n} \phi_2)$ is satisfied, the right state sub-formula (ϕ_2) is satisfied in the last state of σ.*

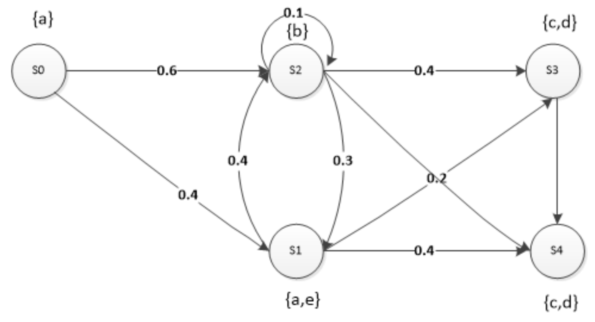


Figure 1: *DTMC*

Example 1 Let us consider the example of DTMC shown in Figure 1 and the property $P_{\leq 0.7}(\varphi)$, where $\varphi = (a \vee b)U(c \wedge d)$. The property above is violated in this model ($s_0| \not\models \mathbf{P}_{\leq 0.7}(\varphi)$), since there exists a set of paths satisfying φ whose probability mass is higher than the probability bound (0.7). Any combination from $FinitePaths(s_0 \models \phi)$ having probability mass higher than 0.7, is a valid counterexample including the whole set. For instance, we can find three counterexamples:

$FinitePaths(s_0 \models \phi) = \{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_1s_2s_3, s_0s_2s_1s_3, s_0s_1s_2s_4\}$

Any combination of finite paths from $FinitePaths(s_0 \models \phi)$ having probability mass higher than 0.7, is a valid probabilistic counterexample including the whole set. We can find three counterexamples:

$P(CX_1) = P(\{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_1s_2s_3, s_0s_2s_1s_3, s_0s_1s_2s_4\})$
$= 0.24 + 0.16 + 0.12 + 0.09 + 0.072 + 0.064 + 0.036 + 0.032 = 0.804$

$P(CX_2) = P(\{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_2s_1s_3\})$
$= 0.24 + 0.16 + 0.12 + 0.08 + 0.072 + 0.036 = 0.708$

$P(CX_3) = P(\{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_1s_2s_3\})$
$= 0.24 + 0.16 + 0.12 + 0.08 + 0.072 + 0.064 = 0.736$

The last probabilistic counterexample is the most indicative since it is minimal and its probability is higher than the other minimal counterexample CX_2,$P(CX_3) > P(CX_2)$.The strongest path is $P(s_0s_2s_3) = 0.24$, which is included in the most indicative probabilistic counterexample.

2.3. Causality Background

Halpern and Pearl (Halpern and Pearl (2001)) have extended the Lewis counterfactual model (Lewis (1973)) to what they refer to as structural equations for modelling the causal influence made by random variables. They introduce the causality model M as a tuple $M = (U, V, F)$, where random variables are partitioned into two sets, the exogenous variables U, whose values are determined by factors outside the model M, but they should be represented to encode the context, and the endogenous variables V whose values are determined by set of functions F, where each $f_{V_i} \in F$ is a mapping from $U \cup (V \setminus V_i)$ to V_i. Thus, each f_{V_i} tells us the value of V_i given the values of all other variables in $U \cup V$.

We call a Boolean combination of primitive events a basic causal formula. We call a Boolean combination of basic causal formulas a causal formula. A causal formula φ is true or false in causal model, given a context. We write $(M, \overrightarrow{u}) \models \varphi$ if φ is true in causality model M given a context $\overrightarrow{u}$, where $\overrightarrow{u}$ represents a setting for the variables in U. We write $(M, \overrightarrow{u}) \models [\overrightarrow{Y} \leftarrow \overrightarrow{y}](X = x)$, where $\overrightarrow{Y} \subset V$, if X has a value x in M given a context $\overrightarrow{u}$ and the assignment $\overrightarrow{y}$ to $\overrightarrow{Y}$. The formulas that are allowed to be causes for φ are ones of the form $X_1 = x_1 \wedge ... \wedge X_k = x_k$ which is abbreviated to the form $\overrightarrow{X} = \overrightarrow{x}$. With all these definitions in hand, we can now give the definition of an actual cause by Halpern and Pearl.

Definition 2.1 (Actual Cause) we say that $\overrightarrow{X} = \overrightarrow{x}$ is an actual cause of φ in $(M, \overrightarrow{u})$ if the following holds:

1. $(M, \overrightarrow{u}) \models (\overrightarrow{X} = \overrightarrow{x}) \wedge \varphi$
2. There exists a partition $(\overrightarrow{Z}, \overrightarrow{W})$ of V with $\overrightarrow{X} \subseteq \overrightarrow{Z}$ and some setting $(\overrightarrow{x}', \overrightarrow{w}')$ of the variables in $(\overrightarrow{X}, \overrightarrow{W})$ such that if $(M, \overrightarrow{u}) \models Z = z^*$ for $Z \in \overrightarrow{Z}$, then
   a-$(M, \overrightarrow{u}) \models \left[\overrightarrow{X} \leftarrow \overrightarrow{x}', \overrightarrow{W} \leftarrow \overrightarrow{w}'\right] \wedge \neg\varphi$
   b-
   $(M, \overrightarrow{u}) \models \left[\overrightarrow{X} \leftarrow \overrightarrow{x}, \overrightarrow{W} \leftarrow \overrightarrow{w}', \overrightarrow{Z}' \leftarrow \overrightarrow{z^*}\right] \wedge \varphi$
   for all subsets $\overrightarrow{Z}'$ of $\overrightarrow{Z}$.
3. The set of variables $\overrightarrow{X}$ is minimal (no subset of $\overrightarrow{X}$ satisfies the conditions 1 and 2).

The first condition states that both $\overrightarrow{X} = \overrightarrow{x}$ and φ are true in the current context, given the variables $\overrightarrow{X}$ and their values $\overrightarrow{x}$. The second condition states that any change on $(\overrightarrow{X}, \overrightarrow{W})$ will change φ from true to false, changing $\overrightarrow{W}$ will have no effect on φ as long as the values of $\overrightarrow{X}$ are kept at the current values, even if all subsets $\overrightarrow{Z}'$ of $\overrightarrow{Z}$ are set to their original values in the current context. Minimality condition ensures that only elements in the conjunction $\overrightarrow{X} = \overrightarrow{x}$ are essential for changing φ from true to false. As a consequence of this condition, we can refer to a cause as $X = x$, because the causes can always be taken to be single conjuncts (Chockler and Halpern (2004)).

2.4. Regression Analysis

Regression analysis is a statistical technique enables us to reason about the relationship between variables. It describes the change in value of a variable Y, namely dependent or endogenous in response to the variation of a variable X namely independent or exogenous. The relationship between these variables is either positive or negative. If the value of Y increases when the value of X rises, it is said that the relationship is positive. Conversely, the relationship is said to be negative. One of the relationships that we can infer and statistically reason about using regression analysis is causality. The causal model is build based on assumptions or hypothesis, and then is tested against statistical data to determine how the specified model fits the data. So when it comes to causal relationships, we can use regression analysis to estimate the causal effect instead of exploring the causal relationships. The causal relationship between dependent variable Y and independent variable X with the consideration of factors noise called disturbance term or error term and denoted ε is given by the linear equation

$$Y = \beta X + \varepsilon \tag{1}$$

Where X stands as a cause for Y,β stands as a coefficient or weight that quantifies the direct causal effect of X on Y. ε is an error term stands for all extraneous variables, which are assumed to be uncorrelated with X. Whereas the variables X and Y are perfectly known, the objective behind using regression analysis is to produce an estimate of the two parameters β and ε.

3. CAUSES IN PROBABILISTIC COUNTEREXAMPLES

For probabilistic properties of the form $\phi = \mathbf{P}_{\leqslant p}(\varphi)$, explaining the violation reduces to the explanation of exceeding the probability bound over the DTMC model. Therefore, the question:what labelling and/or probability values in the counterexample causes the system to falsify a specification, reduces to the question: what labelling and/or probability values in the counterexample causes the exceeding of probability bound over the model.

3.1. Causality Model

A causality model for the most indicative probabilistic counterexample $MIPCX(s_0 \models \phi)$ is a tuple $M = (U, V, F)$, where the set U is represented by a context variable; its value u represents a state $s \in MIPCX(s_0 \models \phi)$. 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.

For example, $f_{p\wedge r}(s, p = 1, r = 1) = 1$ where p and r are atomic propositions, and s is state representing a context. The causal influence is modelled by the transitions in $MIPCX(s_0 \models \phi)$.

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

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

That is, for the probabilistic counterexample $MIPCX(s_0 \models \phi)$, we say that the value of a variable X is critical for the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$ in a state s, if changing the value of X in this state renders the result not a counterexample.

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

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

Definition 3.4. A Diagnostic causality model for $MIPCX(s_0 \models \phi)$ is a tuple $\langle M, CC^\sigma \rangle$, where M is a causality model, and CC^σ is a contribution function that assigns to each cause its contribution to the satisfaction of φ with respect to a path $\sigma \in MIPCX(s_0 \models \phi)$ as follows

$$CC^\sigma(X = x) = \sum_{s\in\sigma | f_X(s)=x} P(s) \qquad (2)$$

That is, with respect to each path $\sigma \in MIPCX(s_0 \models \phi)$, we have associated to each cause a measure that represents the sum of probabilities of reaching the states in which $X = x$. We can say that $X = x$ is a cause for the satisfaction of φ with respect to a path σ with a contribution CC^σ. So, the cause having the highest contribution will be the one found the most along the path.

Example. Consider the most indicative counterexample

$$CX_3 =$$
$$\{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_1s_2s_3\}$$

generated from the DTMC presented in Fig1 against the property $P_{\leq 0.7}(\varphi)$, where $\varphi = (a \vee b)U(c \wedge d)$. It is possible to define a causality model for CX_3, where $u \in \{s_0, s_1, s_2, s_3, s_4, s_5\}$, and F can be defined over the variables in V as follows

$$f_a(s_1) = 1$$
$$f_{c\wedge d}(s_1, c = 0, d = 0) = 0$$
...

For instance, it is clear that in s_2, the actual cause for the satisfaction of $\varphi = (a \vee b)U(c \wedge d)$ is $b = 1$. The probability of the actual cause $b = 1$ in the path $\sigma_2 = s_0s_2s_3$ is $CC^{\sigma_2}(b = 1) = 0.6$.

3.2. Model Analysis

We aim now quantify the effect of a cause $X = x$ on the violation of ϕ. To do so, we use the regression analysis as a comprehensive technique for estimating this effect, where the path formula φ will stand as a dependent variable, whereas the formula of the form $X = x$ will stand as an independent variable. So, the regression model will describe the behaviour of the system by analysing the change of the probability of φ satisfaction with respect to variables that are considered to be causes.

While the variables are now well defined, the remaining task is to generate the data required for estimating the structural parameters. With respect to the definition of diagnostic causality model, we have seen that each cause $X = x$ has a contribution $CC^\sigma(X = x)$ with respect to each path $\sigma \in MIPCX(s_0 \models \phi)$. So in our setting, $P(\sigma)$ will stand for the value of φ and $CC^\sigma(X = x)$ will stand for the value of $X = x$. As a result, the number of data rows will be exactly the number of finite paths forming the $MIPCX(s_0 \models \phi)$. We present an algorithm for generating the causes and their contributions, hence constructing the data set.

This algorithm performs on counterexample generated by the tool DiPro (Aljazzar and Leue (2011)), since probabilistic model checkers do not have the ability to generate counterexamples. DiPro is a tool used for generating counterexamples from DTMC, CTMC and MDPs models, and can be jointly used with the model checkers PRISM and MRMC. The algorithm gets from DiPro tool the counterexample $MIPCX(s_0 \models \phi)$ and the probabilistic formula $\phi = \mathbf{P}_{\leqslant p}(\varphi)$ as input, and outputs the dataset. The formula φ is until formula of the form $\phi_1 \mathbf{U} \phi_2 (\phi_1 \mathbf{U}^{\leq n} \phi_2)$given in NNF (Negative Normal Form).

```
**Algorithm. ComputeDataset**
**Inputs:** The probabilistic formula $\phi = P_{\leq p}(\varphi)$
        The counterexample $MIPCX(s_0 \models \phi)$
**Outputs:** DataSet: Variables (causes) with values ($CC^{\sigma}$)
begin
    Causes := $\varnothing$
    for each path $\sigma \in MIPCX(s_0 \models \phi)$ do
        for each state $s \in \sigma$ do
        if $s$ is the last state in $\sigma$ then
          Causes := findCauses($s, \phi_2$)
          $CC^{\sigma}(Causes) = \sum_{s \in \sigma | f_X(s) = x} P(s)$
        end if
        else
          Causes := findCauses($s, \phi_1$)
          $CC^{\sigma}(Causes) = \sum_{s \in \sigma | f_X(s) = x} P(s)$
        end else
        end for
    end for
end ComputeDataset
function findCauses($s$, $\psi$)
begin
    if $\psi$ is of the form $\psi_1 \wedge \psi_2$ then
        return  findCauses($s$, $\psi_1$) $\cup$ findCauses($s$, $\psi_2$)
    end if
    If $\psi$ is of the form $\psi_1 \vee \psi_2$ Then
      If $s \models \psi_1$ and $s \models \psi_2$ Then
        return FindCauses($s$, $\psi_1$) $\cup$ FindCauses($s$, $\psi_2$)
      End If
      If $s \models \psi_1$ and $s \not\models \psi_2$
        return FindCauses($s$, $\psi_1$)
      End If
      If $s \not\models \psi_1$ and $s \models \psi_2$
        return FindCauses($s$, $\psi_2$)
      End If
    End If
    if $\psi$ is of the form $a$ where $a \in AP$ and $a \in L(s)$ then
        return $a$
    end if
    if $\psi$ is of the form $\neg a$ where $a \in AP$and $a \notin L(s)$ then
        return$\neg a$
    end if
end findCauses
```

The algorithm explores the counterexample path by path, and computes the causes that will act as data variables, and computes their probabilities that will act as data values. We notice that these two tasks are performed simultaneously. We compute $CC^{\sigma}(X = x)$ of each cause $X = x$ found in set of states. The condition put on last state follows Lemma 2.3. The main function of this algorithm is FindCauses. It takes a state and state formula as input and returns recursively a set of causes. We note that when the state formula ψ is of the form $\psi_1 \wedge \psi_2$, both sub-formulas are essentially true at state s. But When ψ is of the form $\psi_1 \vee \psi_2$, one of them could be true at s or both of them. This actually follows the causal intuition that in conjunctive scenario, both $\psi_1$and ψ_2 are required for ψ being satisfied, whereas in the disjunctive scenario, either ψ_1 or ψ_2 suffices to make ψ satisfied. In the two cases, we apply FindCauses to each sub-formula. Finally, at the propositional level, the cause would be $a(a = 1)$ or $\neg a(a = 0)$, and it is allowed to be a conjunction of primitive formulas.

The Algorithm over-approximates the set of causes , since computing the set of causes exactly in binary causal models is NP-complete (Eiter and Lukasiewicz (2002)). In (Beer et al. (2011)), the authors introduced a polynomial algorithm that approximates the set of causes for the violation of LTL formula. The reduction from binary causal models to Boolean circuits and from Boolean circuits to model checking as introduced in (Chockler et al. (2007)) proved that computing the causes and the degree of responsibility for branching time formula is also NP-complete, and they provided a polynomial algorithm for computing responsibility for read-once Boolean formulas.

It is evident that the complexity of this algorithm is polynomial in size of $MIPCX(s_0 \models \phi)$ and the size of φ. This follows from the fact that the left sub-formula ϕ_1 is evaluated in each state except the last ones, in which the right sub-formula $\phi 2$ is evaluated, this is much less hard than evaluating both sub-formulas at each state of the counterexample (Beer et al. (2011)). Moreover, if the sub-formula is AND formula, it is not necessary to be evaluated, since its satisfaction is ensured. Once the causes found in state s, computing the causes probability is performed each time at a state s with respect to each path.

3.3. Illustrative Example

We modelled the DTMC presented before in Fig.1 in PRISM. Then we generate most indicative counterexample using DiPro for the property

$\mathbf{P}_{\leq 0.7}(\varphi)$. The counterexample generated is $CX3$ where

$$P(CX_3) = P(\{s_0s_2s_3, s_0s_1s_4, s_0s_2s_4, s_0s_1s_3, s_0s_2s_1s_4, s_0s_1s_2s_3\}) = 0.24 + 0.16 + 0.12 + 0.08 + 0.072 + 0.064$$

We pass the counterexample to be analyzed to our algorithm, which is implemented in Java. It takes the counterexample generated by DiPro in XML format and returns the dataset. The set of causes as can be generated for this counterexample using our algorithm is as follows:$C_1 = a, C_2 = b, C_3 = \{c, d\}$

These causes denoted C_i stand as independent variables (X) for the regression model we are going to build, where the until formula φ denoted UF stands for the dependent variable (Y). The data values of these observed variables are given in the table bellow

UF	C1	C2	C3
0.24	1	0.6	0.24
0.16	1.04	0	0.16
0.12	1	0.6	0.12
0.08	1.04	0	0.08
0.072	1.03	0.6	0.072
0.064	1.04	0.4	0.064

Table 1: *CX_3 data set*

As we see in the table, the number of data rows represents the number of paths of $MIPCX(s_0 \models \phi)$, where the value of until formula UF refers to the probability of each path $P(\sigma)$ and the value of cause refers to $CC^{\sigma}(X = x)$.

4. EXPERIMENTAL RESULTS

We have implemented the above method in Java. To evaluate our method, we use two benchmark case studies, the embedded control systems taken from (Muppala et al. (1994)) and the cyclic polling model taken from (Ib and Trivedi (1990)). All the experiments were carried out on windows XP with Intel Pentium CPU 3.2 GHz speed And 512 mb of memory.

4.1. Embedded Control System

The system consists of input processor (I) that reads incoming data from three sensors (s1,s2,s3) and then passes it to main processor (M). The processor M processes the data and sends instructions to an output processor (O) that controls two actuators (A1 and A2) using these instructions.Any of the system's components M, I/O, the sensors and the actuators may fail; as a result the system is shut down. The types of failure are:

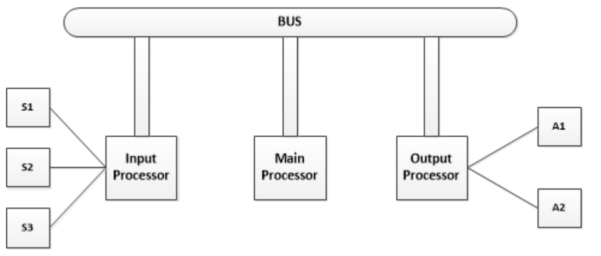


Figure 2: *Embedded controle system*

$$fail\_sensors = (i = 2 \wedge s < MIN\_SENSORS)$$
$$fail\_actuators = (o = 2 \wedge a < MIN\_ACTUATORS)$$
$$fail\_io = (count = MAX\_COUNT + 1)$$
$$fail\_main = (m = 0)$$

We use the variable Max_Count to refer to the maximum number of consecutive cycles skipped allowed. Thus, the I/O processor will fail if the count exceeds the limit Max_Count. The down status of the system is labelled as:

$$down = fail\_sensors | fail\_actuators | fail\_io | fail\_main$$

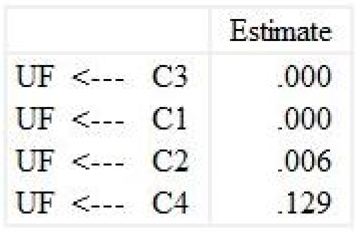
That is, the systems is down if any of this failures occurs. For this model, we choose the property that estimates the probability of I/O failure occurring first, which is given as follows:

$$P = ?[\neg(down) U fail\_io]$$

We test this property using PRISM for ($Max\_Count = 6$). For this value, PRISM renders a probability equal to 0.11. We chose the value 0.1 as a threshold for this property to generate the counterexample. Thus the property can be rewritten as follows:

$$P \leq 0.1[\neg(down) U fail\_io]$$

We use DiPro to generate the counterexample, which in turn uses PRISM. The counterexample can be rendered graphically and stored in XML format. The tool implements many algorithms. In our experiments, we used the heuristic search algorithm XBF that computes the counterexample as a diagnostic sub-graph. Our method takes the counterexample generated from DiPro in XML format and the property to be verified , and outputs the causes and their values as a dataset in Excel file. This excel file is passed to the tool AMOS in order to generate the regression model. AMOS is well-known software for SEM that enables user to specify, confirm and refine models, by incorporating many statistical methods.



			Estimate
UF	<---	C3	.000
UF	<---	C1	.000
UF	<---	C2	.006
UF	<---	C4	.129

Figure 3: *Regression weights*

The PRISM model consists of 6858 states and 28907 transitions. For generating the counterexample, DiPro Explored 480 traces in about 5 minutes resulting in 1685 vertices and 3291 edges. Finally, the counterexample rendered consists just of 33 diagnostic paths. It is evident that the number of explored vertices and explored edges while searching the counterexample is always less than the number of states and the transitions of the model. It is also evident that the number of diagnostic paths is less than the number of solution traces. While solution traces refer to all the paths of the diagnostic subgraph found through exploring the model, diagnostic paths refer just to the paths forming the counterexample.

We pass this counterexample to our algorithm for generating the dataset of causes. The causes will be the basic sub-formulas causing the satisfaction of $\neg(down)U fail\_io$. For the right sub-formula, the cause generated is $C0 = (count = MAX\_COUNT + 1)$. For the left sub-formula, the set of causes is

$C1 = \neg(i = 2)$, $C2 = \neg(s < MIN\_SENSORS)$
$C3 = \neg(o = 2)$, $C4 = \neg(a < MIN\_ACTUATORS)$
$C5 = \neg(count = MAX\_COUNT + 1)$
$C6 = \neg(m = 0)$

After generating these causes with their probabilities with respect to each path as a dataset, we found the following results: along all paths (data rows), C0 has the same probability. C2, C5 and C6 have exactly equivalent probabilities with respect to each path (data row). This means that they always occur together. As a result, before passing the data set to AMOS tool, C0 will be ignored since its value is constant along all paths.C2, C5 and C6 will be regrouped into one variable named (C2), since they share the same values. Thus, the final data set will consist of the causes C1, C2, C3, C4 and the 'until' formula φ denoted UF. For estimating the causal effect of each cause on UF, we generate the regression model based on this data set using AMOS. The results as rendered by AMOS tool are presented in Fig.3.

What concerns us more, is how to get the diagnostic information by interpreting these weights or coefficients. We should recall that C1(input processor is not in OK state),C2(all sensors are working) are the probable causes for not sensor failure occurring, whereas C3(output processor is not in OK state),C4 (all actuators are working) are the causes for not actuators failure occurring. Here we are facing a disjunctive scenario for both failures. In fact, the weights presented above have more importance when we face a disjunctive scenario $(\psi_1 \vee \psi_2)$, because the designer will need to know which sub-formula is more responsible for the satisfaction of φ along the paths.

The results as generated by AMOS tool shows that C4 has the highest effect where C2 has more effect than C1 and C4 has more effect than C3. Performing a fast check on the dataset generated, these explanations are confirmed, which means that along all the paths, C2 and C4 are usually the actual causes not C1 and C3, with great notable presence for C4 comparing to C2.

Taking just $MIPCX(s_0 \models \phi)$ to be the causality model instead of the whole model is due first to the nature of $MIPCX(s_0 \models \phi)$ itself; $MIPCX(s_0 \models \phi)$ by definition covers the most probable causes for the error, since it is consisting of paths with high probabilities. Second, lowest probability values will have no effect on the regression model generated since they tend to be zero. We tested this issue, by generating a counterexample for the property $P = ?[\neg(down)U fail\_io]$.

That is, the counterexample will consist of all paths satisfying $\neg(down)U fail\_io$, not just the paths with high probabilities. The counterexample generated consists of 132 paths, DipRo takes more than 30 minutes for computing this counterexample. Adding the new paths (99 paths) to the old data set and analysing it using AMOS did not bring considerable change to the previous regression model generated depicted in Fig.3.

4.2. Polling Server System

The system is modelled in PRISM as a CTMC, where the number of stations handled by the polling server is denoted by N. Each station has a single-message buffer and is cyclically attended by the server. We choose the property that measures the probability of station 1 being served (s=1) before station 2 (s=2) where (a=1) denotes serving. This property is given as follows:

$$P =?[!(s = 2 \wedge a = 1)U(s = 1 \wedge a = 1)]$$

We test this property using PRISM for (N=3, N=5, N=7 and N=9). For all of these values, PRISM renders a probability higher than 0.5. As a result, we chose the value 0.5 as a threshold. The property can be rewritten as follows:

$$P \leq 0.5[(!(s=2) \vee !(a=1)) U (s=1 \wedge a=1)]$$

We use DiPro to generate the counterexample, which in turn uses PRISM. We have to specify the model and the property to be verified, and then DiPro computes the counterexample for violating the probability threshold. We used the heuristic search algorithm XBF that computes the counterexample as a diagnostic sub-graph. Our method takes the counterexample generated from DiPro and the property to be verified as parameters, and outputs the dataset.

The PRISM model consists of 1344 states and 5824 transitions. For generating the counterexample, DiPro Explored 184 traces in about 5 minutes resulting in 1094 vertices and 3310 edges. Finally, the counterexample rendered consists just of 18 diagnostic paths. It is evident that the number of explored vertices and explored edges while searching the counterexample is always less than the number of states and the transitions of the model.

We pass this counterexample to our algorithm for generating the dataset of causes. The causes will be the basic sub-formulas causing the satisfaction of $(!(s=2) \vee !(a=1)) U (s=1 \wedge a=1)$. For the right sub-formula, the cause generated is $C0 = (s = 1 \wedge a = 1)$. For the left sub-formula, the set of causes is $C1 = !(a=1)$ and $C2 = !(s=2)$.

After generating these causes with their contribution with respect to each path as a dataset, we found that $C0$ has the same probability along all paths (data rows). thus before passing the data set to AMOS tool, C0 will be ignored since its value is constant along all paths. The effect of $C1$ and $C2$ on UF is as generated by AMOS is given by the following equation:

$$UF = -(0.304)C1 + (0.464)C2 \quad (3)$$

The results as rendered by AMOS shows that C2 has higher effect than C1, where the effect of C1 is negative. This means that the effect of C1 on UF increases once the paths probabilities get lower, which means that the presence of C1 increases just with in the paths with low probabilities.

5. RELATED WORKS

Various approaches for probabilistic counterexample generation have been proposed. The authors (Aljazzar et al (2005), Aljazzar and Leue (2006) introduced an approach for counterexample generation for DTMC and CTMC against timed reachability properties using heuristics guided and directed explicit state space search. In complementary work (Aljazzar and Leue (2009)), with the intuition that single scheduler makes an MDP as DTMC, they proposed an approach for counterexample generation for MDPs using existing methods for DTMC. They introduced more complete work in (Aljazzar and Leue (2010)) for generating counterexample as what they refer to as diagnostic sub-graphs. Based on all the previous works, they built a tool, DiPro (Aljazzar and Leue (2011)), for generating counterexamples for DTMC, CTMC and MDPs. Similar to the previous works, (Han et al. (2009)) 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 (Wimmer et al. (2012)) have proposed a novel approach based on critical subsystems. Following this work, the authors in (Jansen et al. (2012)) 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, (Wimmer et al. (2013)) suggests to rely directly on the guarded command language used by the model checker, which is more likely and helpful for debugging purpose.

In conventional model checking, many works have proposed techniques and algorithms for discovering error causes from counterexamples, hence presenting them to the user in a comprehensive way. Most of these works (Ball et al. (2003), Zeller et al. (2002), Groce et al. (2003), Groce et al. (2006)) consider the existence of successful runs or executions to be compared with the error trace, with the assumption that the more successful execution closed to the error trace indicates the causes of the error. Based on Lewis counterfactual theory of causality (Lewis (1973)) and distance metrics, the authors in (Groce et al. (2006)) have proposed semi-automated approach for isolating errors in ANSI C programs by considering the alternative worlds as programs executions and the events as propositions about those executions. Unlike the previous works that require multiple executions, the work (Wang et al. (2006)) introduced a technique performed on a single concrete execution path using a weakest pre condition algorithm. While all of these works addressed safety properties, some of works attended to explain errors for liveness properties that involve more computational complexity (Kumazawa and Tamai (2011)).

In recent work, (Debbi and Bourahla (2013))used the definition of causality and its quantitative measure responsibility for analysing probabilistic counterexamples of DTMCs and CTMCs. The causes are given as pairs (state, variable) and ordered with respect to such weights. Another closely related to our work that of (Beer et al. (2011), Chockler et al. (2007)). They used the definition of causality for explaining LTL counterexamples in (Beer et al. (2011)). Unlike addressing the question in (Beer et al. (2011)), what causes a system to falsify a specification? In the context of coverage (Chockler et al. (2007)), the question addressed was: what causes a system to satisfy specification? In this aim, they adapt the definition of causality and its quantitative measure, responsibility (Chockler and Halpern (2004)). The definition of causality has also been used by (Kuntz et al. (2011); Leitner-Fischer and Leue (2013)). They adapted the definition of causality to event orders for generating fault trees from probabilistic counterexamples. In (Leitner-Fischer and Leue (2013)) they proposed the notion of causality checking by integrating the causality computation in the model checking algorithm itself .

6. CONCLUSION AND FUTURE WORKS

In this paper, we have shown how the notion of causality can be interpreted in the context of probabilistic counterexamples. Due to the quantitative nature of the causal model, we had to define for each cause its contribution to the error. Following that, we generate the regression model corresponding to the causality model. For doing so, we have proposed an algorithm for generating the causes as a data set. We have seen that delivering the causes for the violation of PCTL formula as a regression model stands as a good technique for describing the effect of variables with their values on the error.

Evidently, our approach does not ignore the previous works of counterexample generation. But instead, it acts as a complementary task.

As future works, We aim to show how our method can perform on counterexamples generated from MDPs models. Furthermore, we plan to investigate the problem of incomplete data, which is well known problem in regression, as well as the problem of linear dependence between variables in the context of probabilistic counterexamples.

REFERENCES

Aljazzar, H., Hermanns, H., Leue, S.(2005) Counterexamples for timed probabilistic reachability, In: FORMATS 05. LNCS vol. 3829, pp. 177-195. Springer.

Aljazzar, H., Leue, S.(2006) Extended directed search for probabilistic timed reachability. in FORMATS 06, LNCS vol. 4202, pp. 33-51.

Aljazzar, H. and Leue, S.(2009) Generation of counterexamples for model checking of markov decision processes, In: Proc. of International Conference on Quantitative Evaluation of Systems, pp. 197-206.

Aljazzar, H. and Leue, S (2010) Directed explicit state-space search in the generation of counterexamples for stochastic model checking. IEEE Trans. on Software Engineering vol. 36 , no. 1, pp. 37-60.

Aljazzar, H. and Leue, S. (2011) DiPro - A Tool f or Probabilistic Counterexample Generation, LNCS 6823, pp. 183-187. Aviable at `http://www.inf.uni-konstanz.de/soft/dipro/`

Aziz, A., Sanwal, K., Singhal, V., Brayton R.(2000) Model-checking continuous-time Markov chains. ACM Trans on Computational Logic. vol. 1, no. 1, pp. 162-170.

Baier, C., Haverkort, B., Hermanns, H., Katoen J.-P.(2003) Model checking algorithms for continuous-time Markov chains. IEEE Trans on Software Engineering, vol. 29, no. 7.

Ball, T., Naik, M., Rajamani, S.K.(2003) From symptom to cause: localizing errors in counterexample traces, In: POPL 2003, pp. 97-105.

Beer, I., BenDavid, S., et al. (2011) Explaining counterexamples using causality. Formal Methods Systems Design vol. 40, pp. 20–40.

Chockler, H., Halpern, J.(2004) Responsibility and blame: a structural-model approach, Journal of Artificial Intelligence Research (JAIR) vol. 22, pp. 93-115.

Chockler, H., Halpern, J. Y., Kupferman, O.(2007) What causes a system to satisfy a specification?. ACM Trans. on Computational Logic Vol.5, no. 5, pp. 1-24.

Debbi, H. and Bourahla, M.(2013) Causal Analysis of Probabilistic Counterexamples, in Proceedings of Eleventh ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE),pp . 77–86.

Debbi, H. and Bourahla, M.(2013) Generating diagnoses for probabilistic model checking using causality, Journal of Computing and Information Technology vol. 21, no 1 pp . 13–23.

Eiter, T., Lukasiewicz, T.(2002) Complexity results for structure-based causality. Artificial Intelligence, vol. 142, pp. 53-89, Elsevier.

Groce, A., Visser, W.: What went wrong(2003) explaining counterexamples. In: Ball, T., Rajamani, S.K. (eds.) SPIN, LNCS, vol. 2648, pp. 121-135.

Groce, A., Chaki, S., Kroening, D., Strichman, O.(2006) Error explanation with distance metrics. STTT vol.8, no 3, pp. 229-247.

Halpern, J., Pearl, J.: Causes and explanations (2001) A structural-model approach part I: Causes. In: Proc. of 17th UAI, pp. 194–202. Morgan Kaufman Publishers.

Han, T., Katoen, J.P.(2009) Counterexamples Generation in probabilistic model checking. IEEE Transactions on Software Engineering 35(2).pp. 72–86.

Hansson, H., Jonsson B.(1994) logic for reasoning about time and reliability. Formal aspects of Computing, vol. 6, no. 5, pp. 512-535.

Hinton A., Kwiatkowska M., Norman G., Parker D.: PRISM (2006) A tool for automatic verifi-cation of probabilistic systems. In Proc. 12th International Conference on Tool s and Algo-rithms for the Construction and Analysis of Systems (TACAS 06). LNCS, vol. 3920, pp. 441-444.

Ib, O. C. and Trivedi, K.(1990) Stochastic petri net models of polling systems, IEEE Journal on Selected Areas in Communications, vol. 8 , no. 9, pp.1649-1657.

Jansen N., Abraham E., Volk, M., Wilmer, R., Katoen J.P and Becker B. (2012) The COMICS Tool - Computing Minimal Counterexamples for DTMCs, in ATVA, LNCS vol.7561, pp. 249-353.

Katoen J.-P., Khattri M., Zapreev I. S.(2005) A Markov Reward Model Checker. in Second International Conference on the Quantitative Evaluation of systems (QEST 2005). pp.243-244. IEEE Computer Society.

Kumazawa, T., Tamai, T.2011) Counterexample-Based Error Localization of Behavior Models. NASA Formal Methods, pp. 222–236.

Kuntz, M., Leitner-Fischer, F., Leue, S. (2011) From probabilistic counterexamples via causality to fault trees. LNCS, vol 6894, pp . 71-84.

Leitner-Fischer, F. and Leue, S.: Causality Checking for Complex System Models, in Proceedings of VMCAI ,LNCS, vol 7737, 2013, pp 248–276.

Leitner-Fischer, F. and Leue, S.: Probabilistic fault tree synthesis using causality computation. International Journal of Critical Computer-Based Systems, vol 4, no 2, 2013, pp 119–143.

Lewis, D.: Causation (1973). Journal of Philosophy. Vol. 70, pp. 556-567.

Muppala, J., Ciardo, G., and Trivedi, K. (1994) Stochastic reward nets for reliability prediction, Communications in Reliability, Maintainability and Serviceability,vol.1, no.5.

Wang, C., Yang, Z., Ivancic, F., Gupta A.(2006) Whodunit? Causal Analysis for Counterexamples. In:Proc.of the International Symposium on Automated Technology for Verification and Analysis (ATVA).

Wimmer, R., Jansen N., Abraham E., Becker B., Katoen J.P (2012) Minimal Critical Subsystems for Discrete-Time Markov Models, in TACAS, LNCS vol.7214 , pp . 299-314.

Wimmer, R., Jansen N., Vorpahl A. (2012) High-Level Counterexamples for Probabilistic Automata, in Quantitative Evaluation of Systems (QEST), LNCS vol.8054 , 2013, pp . 39-54.

Wright, Sewall S.: Correlation and causation. Journal of Agricultural Research, vol. 20, 1921, pp. 557-585.

Zeller, A.(2002) Isolating cause-effect chains from computer programs. In: SIGSOFT 2002/FSE 10, pp. 1-10.

AMOS, http://amosdevelopment.com/download/.

Cyclic Server Polling System: available at http://www.prismmodelchecker.org/casestudies/polling.php. /embedded.php.

Embedded Control System: available at http://www.prismmodelchecker.org/casestudies/embedded.ph