Journal of Computing and Information Technology - CIT 21, 2013, 1, 13–22
doi:10.2498/cit.1002115

Generating Diagnoses for Probabilistic Model Checking Using Causality

Hichem Debbi and Mustapha Bourahla

Department of Computer Science, University of M'sila, Algeria

One of the major advantages of model checking over other formal methods of verification is its ability to generate an error trace when the specification is falsified in the model. We call this trace a counterexample. In probabilistic model checking (PMC), counterexample generation has a quantitative aspect. The counterexample is a set of paths in which a path formula holds, and their accumulated probability mass violates the probability bound. In this paper, we address the complementary task of counterexample generation, which is the counterexample diagnosis. We propose an aided-diagnostic method for probabilistic counterexamples based on the notion of causality. Given a counterexample for a probabilistic CTL (PCTL) formula that does not hold over Discrete Time Markov Chain (DTMC) model, this method guides the user to the most responsible causes in the counterexample.

Keywords: Discrete-Time Markov Chain (DTMC), Probabilistic Model Checking (PMC), Probabilistic Computation Tree Logic (PCTL), Probabilistic Counterexamples, Causality

1. Introduction

Probabilistic model checking has appeared as an extension of model checking for modeling and analyzing systems that exhibit stochastic behavior. 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) [24] or Continuous Stochastic Logic (CSL) [7, 8].

For counterexample generation in probabilistic model checking (PMC), many approaches have been proposed. But unlike the previous method proposed for conventional model checking that generates the counterexample as a single path ending with a bad state representing the failure [14], 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 indicative to be easy for analyzing [4, 22].

However, generating small and indicative counterexamples only is not enough for understanding the error. Therefore, many works in conventional model checking have addressed the analysis of counterexamples to better understand the error [9, 19, 20, 28, 38 and 40]. As it was done in conventional model checking, addressing the error explanation in the probabilistic model checking is highly required, especially that probabilistic counterexample consists of multiple paths instead of single path, and that it is probabilistic.

In this paper, we address the diagnosis of probabilistic counterexamples. To this end, we adopt the definition of causality introduced by Halpern and Pearl [21]. In this paper, we will focus on upper bounded properties. The other cases can be transformed easily to upper bounded properties. 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 on the smallest most indicative counterexamples [4, 22]. To our knowledge, no work has been done yet for error explanation in probabilistic model checking.

The rest of this paper is organized as follows. In Section 2, we present some preliminaries and definitions. The probabilistic 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. Following that, we introduce an algorithm for generating the causes and their responsibilities for the violation of PCTL properties. Experimental results are given in Section 5. Section 6 presents the related works. At the end, we present conclusion and future works.

2. Preliminaries and Definitions

We call a discrete-time stochastic process with discrete state space a Discrete-Time Markov Chain (DTMC) if it satisfies the Markov property:

$$P[x_n = i_n|_x 0 = i_0, x_1 = i_1, \ldots, x_{n-1} = i_{n-1}] = P[x_n = i_n|x_{n-1} = i_{n-1}]$$

This means, the probability to pass to next state depends only on the previous state and not on the state's history.

More formally, a 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, $P : S \times S \rightarrow [0,1]$ represents the transition probability matrix, $L : S \rightarrow 2^{AP}$ 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 $s_0 s_1 s_2 \ldots$, 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:

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

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, $\mathbf{P}$ 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 $n \in N$. As in CTL, the temporal operators ($\mathbf{U}$ for strong until, $\mathbf{W}$ for weak (unless) until and their bounded variants) are required to be immediately preceded by the threshold operator $\mathbf{P}$.The PCTL formula is a state formula, where path formulas occur only inside the operator $\mathbf{P}$. The operator $\mathbf{P}$ 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 state formula ϕ 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 $\models$. A path $\sigma = s_0 s_1 \ldots$ satisfies a PCTL formula ϕ if the relation $s_0 \models \phi$ is satisfied. We define the set of paths satisfying the relation $s \models \phi$ by $Paths(s \models \phi)$ and the probability of the satisfaction of $s \models \phi$ by

$$P(s \models \phi) = \sum\nolimits_{\sigma \in Paths(s \models \phi)} P(\sigma)$$

The PCTL semantics is defined as follows:

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

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

In the rest of the paper, we will focus on properties of upper probability bound of the form $\phi_1 \mathbf{U} \phi_2$ or its variant $(\phi_1 \mathbf{U}^{\leq n} \phi_2)$. Probabilistic lower bounded properties can be easily transformed to upper bounded properties [4, 22].

2.2. Probabilistic Counterexamples

The probabilistic counterexamples are generated when a PCTL 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 can be formed of a set of finite paths starting at state s and satisfying the path formula φ. We refer to these paths as $FinitePaths(s \models \phi)$. Thus, each path $\sigma = s_0 s_1 \ldots s_n$ from $FinitePaths(s \models \phi)$ is a prefix of an infinite path from $Paths(s \models \phi)$ satisfying the formula ϕ. $FinitePaths(s \models \phi)$ are also called diagnostic paths [4, 5].

It is clear that we can get a set of probabilistic counterexamples, noted $PCX(s \models \phi)$, which is a set of any combination from $FinitePaths(s \models \phi)$, 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 $FinitePaths(s \models \phi)$) 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.

Lemma 2.1 Let $MIPCX(s_0 \models \phi)$ be the most indicative probabilistic counterexample. Every finite path $\sigma \in MIPCX(s_0 \models \phi)$ is critical. Which means $\forall \sigma : MIPCX(s_0 \models \phi) - \sigma$ (removing any path σ from $MIPCX(s_0 \models \phi)$) will render the result not a counterexample.

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

Lemma 2.2 The most indicative probabilistic counterexample contains at least one strongest path $\sigma \in FinitePaths(s \models \phi)$.

Corollary 2.1 If a path property $\phi_1 \mathbf{U} \phi_2 (\phi_1 \mathbf{U}^{\leq n} \phi_2)$ is satisfied in a finite path σ, the right state sub-formula (ϕ_2) is satisfied in the last state of σ.

Lemma 2.3 According to the semantics presented above, any PCTL path formula, if it is satisfied on a finite path, it is also satisfied on any suffix of this path.

Example 2.1 Let us consider the example of DTMC shown in Figure 1 and the property $P_{\leq 0.5}(\varphi)$, where $\varphi = (a \vee b) U (c \wedge d)$.

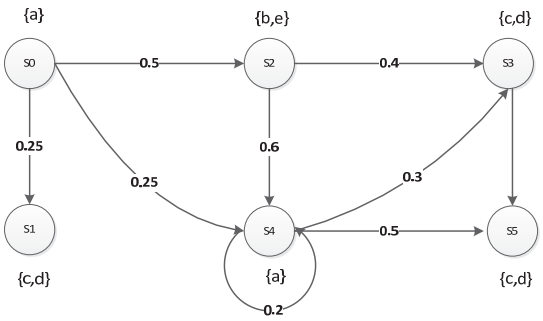

Figure 1. A DTMC.

The property above is violated in this model $(s_0 \not\models \mathbf{P}_{\leq 0.5}(\varphi))$, since the probability mass of all paths satisfying φ is higher than the probability bound (0.5). We have

$$FinitePaths(s_0 \models \phi) = \{s_0 s_1, s_0 s_2 s_3, s_0 s_2 s_4 s_3, s_0 s_2 s_4 s_5, s_0 s_4 s_5\}$$

Any combination of paths from $FinitePaths(s_0 \models \phi)$ having probability mass higher than 0.5,

is a valid probabilistic counterexample including the whole set. For instance, we can find three counterexamples:

$$\begin{aligned}P(C_1)&=P(\{s_0s_1, s_0s_2s_3, s_0s_2s_4s_3, s_0s_2s_4s_5, s_0s_4s_5)\\&=0.25+0.2+0.09+0.15+0.12=0.81\\P(C_2)&=P(\{s_0s_1, s_0s_2s_4s_5, s_0s_4s_5\})\\&=0.25+0.15+0.12=0.52\\P(C_3)&=P(\{s_0s_1, s_0s_2s_3, s_0s_2s_4s_5\})\\&=0.25+0.2+0.15=0.60\end{aligned}$$

The last probabilistic counterexample is the most indicative one, since it is minimal and its probability is higher than the other minimal counterexample C_2, $P(C_3) = 0.6 > P(C_2)$. The strongest path is s_0s_1, which is included in the most indicative probabilistic counterexample.

3. Causes in Probabilistic Counterexamples

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

3.1. Causality Model

The counterfactual notion of causality [30] states that: event A is a cause of event B if, had A not happened, then B would not have happened. Unfortunately, this statement does not cover all real cases. Let's take the major known example of Suzy and Billy who both pick up rocks and throw them at a bottle [21]. Suzy's rock gets there first, shattering the bottle. Since both throws are perfectly accurate, Billy's would have shattered the bottle had it not been preempted by Suzy's throw. Thus, according to the counterfactual condition, Suzy's throw is not a cause for shattering the bottle. Halpern and Pearl [21] have addressed this issue by taking A to be a cause of B if B counterfactually depends on A under some contingency. For example, Suzy's throw is a cause of the bottle shattering because the bottle shattering counterfactually depends on Suzy's throw, under the contingency that Billy does not throw.

With respect to the definition of causality by Halpern and Pearl [21], the causality model M is defined by a set of exogenous variables U, whose values $\overrightarrow{u}$ are determined by factors outside the model M, but they should be represented to encode the context, and by a set of endogenous variables V, whose values are determined by structural equations, and set of functions F, where each $f_{v_i} \in F$ is a mapping from $U \times (V \setminus V_i)$ to V_i. Thus, each f_{v_i} tells us the value of V_i given the values of all other variables in $U \cup V$. The causality model M has no control on context changes.

We can define a causality model for the most indicative counterexample $MIPCX(s_0 \models \phi)$ as a tuple $M = (U, V, F)$, where U is presented by a single variable; we call it a context variable, its value u encodes a state in $MIPCX(s_0 \models \phi)$. V is a set of variables representing atomic propositions and Boolean formulas. F is a set of truth functions associating to every variable in V a value (0 or 1). So, each $f_{v_i} \in F$ tells us the value of a variable in V given the values of all the other variables. 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 influences in are modelled by the transitions in $MIPCX(s_0 \models \phi)$.

3.2. Generating Causes

We call the path until formula φ a causal formula. We write $(M, u) \models \varphi$ if φ is true in causality model M given a context u. We write $(M, u) \models [\overrightarrow{Y} \leftarrow \overrightarrow{y}](X = x)$, if X has a value $(0 or 1)$ in M given a context u and the assignment $\overrightarrow{y}$ to $\overrightarrow{Y}$, where $\overrightarrow{Y} \subset V$.

We write $(M, u) \models \varphi$ if φ is true in causality model given a context u. For probabilistic causality, we add a probability to each context, which is a state s belonging to finite paths from $MIPCX(s_0 \models \phi)$. We should note that a state can belong to other paths that are not in the most indicative probabilistic counterexample.

Therefore, a context probability is defined by this equation:

$$P(u=s) = \sum_{s\in\sigma|\sigma\in MIPCX(s_0\models\phi)} P(\sigma)$$

The statement for causality is "A formula C is a cause of φ in context u of M". The types of formulas that are allowed to be causes for φ are ones of the form $X_1 = x_1 \wedge \ldots \wedge X_k = x_k$ which is abbreviated to the form $\overrightarrow{X} = \overrightarrow{x}$. Since F defines a mapping from U to V, We can associate to each cause a probability that represents exactly the context probability as $P(\overrightarrow{X} = \overrightarrow{x}) = P(s)$.

With all these definitions in hand, we can now give the definition of an actual cause for the violation of PCTL property $\phi = \mathbf{P}_{\leq p}(\varphi)$.

Definition 3.1 we say that $\overrightarrow{X} = \overrightarrow{x}$ is an actual cause for the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$ in (M,s) with a probability equal to $P(\overrightarrow{X} = \overrightarrow{x}) = P(s)$ if the following holds:

1. $(M,s) \models (\overrightarrow{X} = \overrightarrow{x}) \wedge \varphi$
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 $(M,s) \models Z = z$ for $Z \in \overrightarrow{Z}$ then $(M,s) \models \left[\overrightarrow{X} \leftarrow \overrightarrow{x}', \overrightarrow{W} \leftarrow \overrightarrow{w}'\right] \wedge \neg\varphi$ and $(M,s) \models \left[\overrightarrow{X} \leftarrow \overrightarrow{x}, \overrightarrow{W} \leftarrow \overrightarrow{w}', \overrightarrow{Z}' \leftarrow \overrightarrow{z}\right] \wedge \varphi$ 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.

Each cause $X = x$ has a probability $P(X = x)$ that measures its contribution to the error, where the cause involved in the satisfaction of φ in more paths is usually the most probable cause. So that, given the probability of $MIPCX(s_0 \models \phi)$, we associate to each cause a degree of contribution for the error, and we call it responsibility. The degree of responsibility of each cause is given by:

$$R(X = x) = P(X = x)/P(MIPCX(s_0 \models \phi))$$

Definition 3.2 A cause C_1 has more responsibility over another cause C_2 for the violation of $\phi = \mathbf{P}_{\leq p}(\varphi)$, if $R(C_1) > R(C_2)$.

Example 3.1 Consider the most indicative counterexample $C_3 = \{s_0s_1, s_0s_2s_3, s_0s_2s_4s_5\}$ generated from the DTMC presented in Figure 1 against the property $P_{\leq 0.5}[(a \vee b)U(c \wedge d)]$.

It is possible to define a causality model for C_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

$$f_b(s_2) = 1$$
$$f_{c\wedge d}(s_2, c = 0, d = 0) = 0$$
$$\vdots$$

For instance, it is clear that in $s_2, \overrightarrow{X} = \{b\}$, $\overrightarrow{Z} = \{b, e\}$ and $\overrightarrow{W} = \{a, c, d\}$. So, the cause in s_2 is $b = 1$ with a responsibility $R(b = 1) = (0.2 + 0.15)/0.6 = 0.58$.

4. Algorithm for Generating Causes

This algorithm performs on counterexample generated from the tool DiPro [5], since probabilistic model checkers do not offer the possibility 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 [26] and MRMC [27], and can render the counterexamples in text formats as well as in graphical mode.

The algorithm gets from DiPro tool the counterexample $MIPCX(s_0 \models \phi)$ and the probabilistic formula $\phi = \mathbf{P}_{\leq p}(\varphi)$ as input, and outputs the causes with their responsibilities. The

formula φ is until formula written in NNF (Negative Normal Form), which means that negation appears just at the front of atomic propositions.

Algorithm. GenerateCauses
Inputs: The most indicative counterexample $MIPCX(s_0 \models \phi)$
The probabilistic formula $\phi = P_{\leq p}(\varphi)$ where φ is of
the form $\phi_1 \mathbf{U} \phi_2$ or $(\phi_1 \mathbf{U}^{\leq n} \phi_2)$
Outputs: Set of causes with their responsibilities
Begin
Contexts := ComputeContexts($MIPCX(s_0 \models \phi)$)
Causes := $\emptyset$
For each context s from the contexts set
 If s is the last state in a path σ **then**
  Causes := Causes $\cup$ **FindCauses(**s, ϕ_2**)**
  R(Causes)=$\sum_{s \in \sigma | \sigma \in MIPCX(s_0 \models \phi)} P(\sigma) /$
  $P(MIPCX(s_0 \models \phi))$
 End If
 Else
  Causes := Causes $\cup$ **FindCauses(**s, ϕ_1**)**
  R(Causes)=$\sum_{s \in \sigma | \sigma \in MIPCX(s_0 \models \phi)} P(\sigma) /$
  $P(MIPCX(s_0 \models \phi))$
 End Else
End For
Sort (Causes)
End GenerateCauses
Function FindCauses(s, ψ)
Begin
If ψ is of the form a where $a \in AP$ and $a \in L(s)$
 Then return $\langle s, a \rangle$
End If
If ψ is of the form $\neg a$ where $a \in AP$ and $a \notin L(s)$
 Then return $\langle s, \neg a \rangle$
End If
If ψ is of the form $\psi_1 \wedge \psi_2$ **Then**
 return {FindCauses(s, ψ_1) $\cup$
  FindCauses(s, ψ_2)}
End If
If ψ is of the form $\psi_1 \vee \psi_2$ **Then**
 return {FindCauses(s, ψ_1)} $\cup$
  {FindCauses(s, ψ_2)}
End If
Otherwise return $\emptyset$
End FindCauses

The algorithm explores the counterexample and computes the causes and their responsibilities with respect to each state s. The causes then will be sorted in order according to their responsibilities R. The main function of this algorithm is FindCauses, which is based on the formula structure. It takes a state and state formula as input and returns recursively the set of causes. The condition put on the last state follows Corollary 2.1. 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 the 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 subformula. Finally, at the propositional level, the cause will be a pair $\langle s, a \rangle$ if $a \in L(s)$ or $\langle s, \neg a \rangle$ otherwise.

This algorithm computes an approximate set of causes, since computing the set of causes exactly in binary causal models is NP-complete [16]. The reduction from binary causal models to Boolean circuits and from Boolean circuits to model checking as introduced in [12] proved that computing a set of causes for branching time formula can be done in linear time. Therefore, further analysis of the complexity of causes computing is beyond the scope of this paper.

5. Experimental Results

We have implemented the above method in Java. To evaluate our method, we use a benchmark case study of the embedded control system taken from [31]. The system is modelled in prism as a CTMC [41]. We should mention that before performing the verification, the CTMC has to be transformed to its embedded DTMC. The system consists of input processor (I) that reads incoming data from three sensors (s1, s2 and 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 failures are:

$fail\_sensors = (i = 2 \wedge s < MIN\_SENSORS)$
$fail\_actuators = (o = 2 \wedge a < MIN\_ACTUATORS)$

$fail\_io = (count = MAX\_COUNT + 1)$
$fail\_main = (m = 0)$

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:

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

For this model, we choose the PCTL property that estimates the probability of I/O failure occurring first, which is given as follows:

$$P =?[!(down)Ufail\_io]$$

We test this property using prism for ($Max\_Count = 1$). For this value, prism renders a probability equal to 0.43. We chose the value 0.4 as a threshold for this property to generate the counterexample. Thus the property can be rewritten as follows:

$$P \leq 0.4[!(down)Ufail\_io]$$

We use DiPro to generate the counterexample, which in turn uses prism. The counterexample rendered by DiPro can be saved in text format as well as in XML format. The tool implements many algorithms. In our experiments, we used its heuristic search algorithm XBF that generates the sub-graph inducing the counterexample. Our method takes the counterexample generated from DiPro in XML format and the property to be verified as arguments, and outputs the causes with their responsibilities. All the experiments were carried out on windows XP with Intel Pentium CPU 3.2 GHz speed and 512 mb of memory.

The prism model consists of 2633 states and 11072 transitions. For generating the counterexample, DiPro Explored 480 traces, 558 vertices and 1080 edges in more than 1 minute. Finally, the counterexample rendered consists of just 35 diagnostic paths. It is evident that the number of explored vertices and explored edges while searching the counterexample is less than the number of states and the transitions of the model. It is evident also 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 sub-graph found through exploring the model, diagnostic paths refer just to the paths forming the counterexample $MIPCX(s_0 \models \phi)$. We pass this counterexample to our algorithm for generating the causes and their responsibilities. Our algorithm takes less than 1 second for generating the causes with their responsibilities. We notice that this time is negligible comparing to the size of the model and the time taken for computing the counterexample.

The causes generated are the basic sub-formulas satisfying the path formula $\neg(down)Ufail\_io$. For the right sub-formula ($fail\_io$), the cause generated is $C0 = (count = MAX\_COUNT + 1)$. For the left sub-formula, the set of causes for the system not to be in down state is:

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

$C1$and $C2$ refer to the probable causes for the failure in the level of sensors, whereas $C3$ and $C4$ refer to the probable causes for the failure in the level of actuators. The failure causes for I/O and M are singletons, $C5$ and $C6$ respectively.

The number of states from 35 diagnostic paths, in which we found these causes are estimated to be 145 states, which is much less than the states of the model (2633). The responsibility classes found are 50, ranging between 0 and 0.3, except the initial state that has intuitively 1 as a degree of responsibility, because it is included in all paths. It is evident that the number of responsibility classes (50) is lower than the number of states (145), because the states included in the same set of paths share the same responsibility.

Due to the size of the counterexample, it is not possible to cite here all the pairs (state, variable), but we can give results description concerning $C1$ to $C6$. $C5$ and $C6$ are guaranteed to have the highest responsibility (0.3) in such states, since they are found in all states, because they are singletons. For sensor and actuators failures, we are facing disjunctive scenario, which means that either $C1$ or $C2$ can be a cause of the absence of sensors failure. It is the same

case for actuators failure with $C3$ and $C4$. In all states, $C2$ (all sensors are working) and $C4$ (all actuators are working) are found to be the actual causes of the absence of both, sensors and actuators failures. Thus, they have absolutely more responsibility than the two other causes in such states: $C1$ (input processor not Ok) and $C3$ (output processor not OK), respectively.

6. Related Works

The original algorithm for counterexample generation was proposed by Clarke et al. [14] and was implemented in most symbolic model checkers. This algorithm of generating linear counterexamples for a fragment of ACTL (ACTL∩LTL) was later extended to handle arbitrary ACTL properties [15] using the notion of tree-like counterexamples. Since then, many works have addressed this issue in conventional model checking [10, 18, 32, 36 and 37].

However, the counterexample generated does not indicate where the failure really exists. Therefore, counterexamples analysis is inevitable. In conventional model checking, many works have proposed techniques for discovering error causes from counterexamples, hence presenting them to the user in a comprehensive way. Most of these works range in the software model checking and programs debugging, especially C programs, in the aim to find bugs in the source code [9, 19, 20 and 40]. Based on Lewis counterfactual theory of causality [30] and distance metrics, Groce et al. in [20] 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 work that requires multiple executions, the work [38] introduced a technique performed on a single concrete execution path using the weakest pre condition algorithm. While all of these works addressed safety properties, some of them attempted to explain errors for liveness properties, which involves more computational complexity [28].

For counterexample generation in probabilistic model checking (PMC), many approaches have been proposed. In [1,2], Aljazzar et al. 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 [3], 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 [4] for generating counterexample for DTMC and CTMS as what they refer to as diagnostic sub-graphs. Based on all the previous works, they built an open source tool, DiPro [5], for generating counterexamples for DTMC, CTMC and MDPs. This tool can be used with the probabilistic model checkers PRISM and MRMC and renders the counterexamples graphically. Similar to the previous works, Han et al. have proposed the notion of the smallest most indicative counterexample that reduces to the problem of finding K shortest paths [22, 23]. Instead of generating path-based counterexamples, Wimmer et al. have proposed a novel approach based on critical subsystems [39]. Following this work, the authors in [34] proposed the COMICS tool for generating the critical subsystems that induce the counterexamples. In [6], the authors proposed an approach for finding sets of evidences for bounded probabilistic LTL properties on Markov Decision Processes (MDP) that behave differently from each other giving significant diagnostic information. More special cases are treated in [25, 33 and 35].

Several works have used the definition of causality in the context of model checking. We found that the one most closely related to our work is that of [11, 12]. They used the definition of causality for explaining LTL counterexample [11]. Unlike addressing the question in [11], what causes a system to falsify a specification? In the context of coverage [12], the question addressed was: what causes a system to satisfy a specification? In this aim, they adapt the definition of causality and its quantitative measure, responsibility [13]. The definition of causality has also been used by [29]. They adapt the definition of causality to event orders for generating fault trees from probabilistic counterexamples. They extended their approach by introducing the notion of causality checking, through integrating causality in the model checking algorithm itself [17].

7. 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 probabilistic nature of the causal model, we had to define for each context its probability. Accordingly, we defined for each cause its responsibility that measures its contribution to the error inherited from the context it is located in. Following that, we introduced an algorithm for diagnoses generation that acts as a guided-method to the most responsible causes in the counterexample. The most responsible cause is considered to be the most relevant to the user. Evidently, our approach does not ignore the previous works of counterexample generation, but instead, it acts as a complementary task. To our knowledge, we are the first who introduce diagnosis approach that acts on counterexamples generated in PMC.

As future works, we aim to show how our method can also perform on counterexamples generated from CTMCs against CSL properties, as well as MDPs models. In this paper, we did not introduce a graphical way for representing the causes and their responsibilities, so as a future work, we aim to build a tool for generating the diagnoses graphically.

References

[1] H. ALJAZZAR, H. HERMANNS, S. LEUE, Counterexamples for timed probabilistic reachability. In *Formal Modeling and Analysis of Times systems (FORMATS)*, (2005), pp. 177–195.

[2] H. ALJAZZAR, S. LEUE, Extended directed search for probabilistic timed reachability. In *FORMATS*, (2006), pp. 33–51.

[3] H. ALJAZZAR, S. LEUE, Generation of Counterexamples for Model Checking of Markov Decision Processes. In *Proceedings of the International Conference on Quantitative Evaluation of Systems (QEST)*, (2009), pp. 197–206.

[4] H. ALJAZZAR, S. LEUE, Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. on Software Engineering*, vol. 36 no. 1 (2010), pp. 37–60.

[5] H. ALJAZZAR, S. LEUE, DiPro – A Tool f or Probabilistic Counterexample Generation. *LNCS*, (2011), pp. 183–187. Available at `http://www.inf.uni-konstanz.de/soft/dipro/`

[6] M. E. ANDRÉS, P. R. D'ARGENIO, P. VANROSSUM, Significant Diagnostic Counterexamples in Probabilistic Model Checking. In *Haifa Verification Conference*, (2008), pp. 129–148.

[7] A. AZIZ, K. SANWAL, V. SINGHAL, R. BRAYTON, Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, vol. 1, no. 1 (2000), pp. 162–170.

[8] C. BAIER, B. HAVERKORT, H. HERMANNS, J.-P. KATOEN, Model checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, vol. 29, no. 7 (2003).

[9] T. BALL, M. NAIK, S. K. RAJAMANI, From symptom to cause: localizing errors in counterexample traces. In *Symposium on Principles of Programing Languages (POPL)*, (2003), pp. 97–105.

[10] M. CHECHIK, A. GURFINKEL, A framework for counterexample generation and exploration. In *Fundamentals Approaches to Software Engineering (FASE)*, (2005), pp. 217–233.

[11] H. CHOCKLER, I. BEER, S. BEN-DAVID, A. ORNI, R. TREFLER, Explaining counterexamples using causality. In *Bouajjani, A., Maler, O. (eds.), LNCS*, vol. 5643 (2009), pp. 94–108.

[12] H. CHOCKLER, J. Y. HALPERN, O. KUPFERMAN, What causes a system to satisfy a specification? *ACM Transactions on Computational Logic* vol. 9, no. 3 (2007), pp. 1–24.

[13] H. CHOCKLER, J. Y. HALPERN, Responsibility and blame: a structural-model approach. *Journal of Artificial Intelligence Research (JAIR)*, vol. 22 (2004), pp. 93–115.

[14] E. CLARKE, O. GRUMBERG, M. C. MILLAN, X. ZHAO, Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the Design Automation Conference*, (1995), pp. 427–432.

[15] E. CLARKE, Y. LU, S. JHA, H. VEITH, Tree-like counterexamples in model checking. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, (2002), pp. 19–29.

[16] T. EITER, T. LUKASIEWICZ, Complexity results for structure-based causality. *Artificial Intelligence*, vol. 142 (2002), pp. 53–89, Elsevier.

[17] F. FISCHER S. LEUE, Causality Checking for Complex System Models. In *Proceedings of the VMCAI, LNCS*, vol. 7737 (2013), pp. 248–276.

[18] P. GASTIN, P. MORO, Minimal counterexample generation for SPIN. *LNCS*, Vol. 4595 (2007), Springer.

[19] A. GROCE, W. VISSER, What went wrong: explaining counterexamples. In *Ball, T., Rajamani, S.K. (eds.) SPIN, LNCS*, vol. 2648 (2003), pp. 121–135.

[20] A. GROCE, S. CHAKI, D. KROENING, O. STRICHMAN, Error explanation with distance metrics. *International Journal on Software Tools for Technology (STTT)*, vol. 8, no. 3 (2006), pp. 229–247.

[21] J. HALPERN, J. PEARL, Causes and explanations: A structural-model approach – part I: Causes. In *Proceedings of the 17th UAI*, (2001), pp. 194–202.

[22] T. HAN, J. P. KATOEN, Counterexamples in probabilistic model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, (2007).

[23] T. HAN, Diagnosis, synthesis and analysis of probabilistic models. Ph.D. Thesis, RWTH Aachen University, University of Twenty, 2009.

[24] H. HANSSON, B. JONSSON, Logic for reasoning about time and reliability. *Formal aspects of Computing*, vol. 6, no. 5 (1994), pp. 512–535.

[25] H. HERMANNS, B. WACHTER, L. ZHANG, Probabilistic CEGAR. In *Proceedings of the Computer Aided Verification (CAV), LNCS*, vol. 5123 (2008), pp. 162–175.

[26] A. HINTON, M. KWIATKOWSKA, G. NORMAN, D. PARKER, PRISM: A tool for automatic verification of probabilistic systems. In *Proceedings of the TACAS'06*, vol. 3920 (2006), pp. 441–444.

[27] J.-P. KATOEN, M. KHATTRI, I. S. ZAPREEV, A Markov Reward Model Checker. In *QEST*, (2005), pp. 243–244.

[28] T. KUMAZAWA, T. TAMAI, Counterexample-Based Error Localization of Behavior Models. *NASA Formal Methods*, (2011), pp. 222–236.

[29] M. KUNTZ, F. LEITNER-FISCHER, S. LEUE, From probabilistic counterexamples via causality to fault trees. *LNCS*, vol. 6894 (2011), pp. 71–84, Springer.

[30] D. LEWIS, Causation. *Journal of Philosophy*, vol. 70 (1973), pp. 556žč567.

[31] J. MUPPALA, G. CIARDO, K. TRIVEDI, Stochastic Reward Nets for Reliability Prediction. *Communications in Reliability Maintainability and Serviceability*, vol. 1, no. 5 (1994).

[32] T. NOPPER, C. SCHOLL, B. BECKER, Computation of Minimal Counterexamples by Using Black Box Techniques and Symbolic Methods. In *Proceedings of the International Conference on Computer Aided Design (CAD)*, (2007), pp. 273–280.

[33] N. JANSEN, E. ABRAHAM, J. KATELAAN, R. WIMMER, J. P. KATOEN, B. BECKER, Hierarchical counterexamples for discrete-time Markov chains. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis (ATVA)*, vol. 699 (2011).

[34] N. JANSEN, E. ABRAHÁM, M. VOLK, R. WILMER, J. P. KATOEN, B. BECKER The COMICS Tool – Computing Minimal Counterexamples for DTMCs. In *Proceedings of the ATVA, LNCS* vol. 7561 (2012), pp. 249–353.

[35] M. SCHMALZ, D. VARACCA, H. VOLZER, Counterexamples in probabilistic LTL model checking for Markov chains. In *Proceedings of the International Conference on Concurrency Theory (CONCUR)*, vol. 5710 (2009).

[36] V. SCHUPPAN, A. BIERE, Shortest Counterexamples for Symbolic Model Checking of LTL with Past. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, (2005), pp. 493–509.

[37] J. STAUNTON, J. CLARK, Finding short counterexamples in promela models using estimation of distribution algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, (2011), pp. 1923–1930.

[38] C. WANG, Z. YANG, F. IVANCIC, A. GUPTA, Whodunit? Causal Analysis for Counterexamples. In *Proceedings of the ATVA*, (2006).

[39] R. WIMMER, N. JANSEN, E. ABRAHÁM, B. BECKER, J. P. KATOEN, Minimal Critical Subsystems for Discrete-Time Markov Models. In *Proceedings of the TACAS, LNCS*, vol. 7214 (2012), pp. 299–314.

[40] A. ZELLER, Isolating cause-effect chains from computer programs. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering* , (2002), pp. 1–10.

[41] EMBEDDED CONTROL SYSTEM: CASE STUDY, `http://www.prismmodelchecker.org/casestudies/embedded.php`

Received: November, 2012
Revised: May, 2013
Accepted: May, 2013

Contact addresses:

Hichem Debbi
Department of Computer Science
University of M'sila
Algeria
e-mail: `hichem.debbi@gmail.com`

Mustapha Bourahla
Department of Computer Science
University of M'sila
Algeria
e-mail: `mbourahla@hotmail.com`

HICHEM DEBBI received the B.S. and M.S. degrees in Computer Science from the University of M'sila, Algeria, in 2009 and 2011, respectively. Currently, he is a PhD student at University of M'sila. His research interests are formal methods and model checking.

MUSTAPHA BOURAHLA has Habilitation from the University of Annaba, Algeria (2010), a PhD degree in computer science from the University of Biskra, Algeria (2007) and a Master degree in computer science from the University of Montreal, Canada (1989). He was a member of the VHDL group at Bell-Northern Research, Ottawa, Canada (1989–1993). He worked for Bell Canada for one year. He was teacher-researcher at the University of Biskra (Algeria), from 1994 until 2009. Now, he is teacher-researcher at the University of M'sila (Algeria). He has publications in the domains of VLSI and formal methods. His current research interests are formal methods, especially model checking critical systems, semantics web and business intelligence. Dr. Bourahla is a member of a research group working in the domains of semantics web, decision support systems and formal methods at the University of M'sila (Algeria).