University of Pennsylvania
Dept. of Computer and Information Science
Philadelphia, Pennsylvania, USA

In Partial Fulfillment of the WPE-II Requirement

Advances in Theory and Applications of Token Causality in Trace Analysis

Author: Shaohui Wang

Committee Chair: Professor Boon Thau Loo
Committee Member: Professor Oleg Sokolsky
Committee Member: Professor Sampath Kannan

September 17, 2013

Abstract

Token causality refers to the causal relationship between system events on an individual system execution, as opposed to general causality, where relationships between types of events are studied. When a system fails at run-time, represented as the violations of certain system safety properties on an observed system execution trace, the investigation of token causality is helpful in understanding the root cause why such system safety properties are violated.

In this report, three different techniques in defining token causality are surveyed. Though all based on counterfactual reasoning, the detailed formulations of the causality definitions, and hence the algorithms for deciding/approximating causalities, vary depending on the system definitions and on the intended use of the causalities. For each of the three surveyed approaches, the causality definitions are summarized and illustrated with examples, based on which the strength and limitations of each approach are discussed. The survey is concluded with comparisons across the three approaches and related work, as well as the challenges and potential extensions in the study of token causality.

Contents

List of Tables

List of Figures

1 Introduction

Designing and implementing a complex software system has never been an easy task. Over the years many theories/principles in software engineering have been proposed, yet the problems with faulty software systems have never decreased. Failures in software systems could lead to substantial loss in time, energy, money, or even human life. When a software system fails, it is an important task to establish a causal chain leading from a component fault to the system failure.

The causal chain contains valuable information that can be exploited for further use in system design and analysis. (1) The system designers can learn about potential system design errors from the causal chains and avoid such design errors in the next generation of the system design. (2) The causal chain contains information on why the component errors propagated to the system level so proper control measures can be added to the system so that such error propagation are mitigated. (3) The chance that the identified causes for a system failure can happen is an important source of information in the fault tree analysis.

In this report, I aim to study the advances in the identification of root causes and the exploitation of information from the identification process, based on trace analysis. System traces are either recordings from system execution where key events in the system are kept, or counterexamples generated during the system verification process, such as model checking.

One thing to note is that in this report, token causality is studied as opposed of general causality. Token causality represents the relationships between two events in a given system execution, while general causality refers to the general relationship between categories of events. For example, "calling methods on uninitialized class variables may cause `NullPointerException` in Java" is a statement of general causality, while token causality studies, for example, whether an occurrence of a `NullPointerException` is caused by not initializing certain variables, given a particular program execution.

The three papers that are selected are as follows.

[8] G. Gössler, D. L. Métayer, and J.-B. Raclet. Causality analysis in contract violation, in Runtime Verification, 2010.

[2] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. In Computer Aided Verification, pages 94–108. Springer, 2009.

[12] M. Kuntz, F. Leitner-Fischer, and S. Leue. From probabilistic counterexamples via causality to fault trees. In Computer Safety, Reliability, and Security, volume 6894 of Lecture Notes in Computer Science, pages 71–84. Springer Berlin Heidelberg, 2011.

The three papers focus on different aspects of causality analysis: definitions of causality given a particular system model, usage of causality for explaining failures, and exploitation of causality information from a given set of traces to

obtain new knowledge in system analysis. The first paper discusses the ingredients that are necessary to define a cause in the engineering domain. The second paper discusses one possible use of the token causality relationship: providing an explanation to the violation of system properties. The third paper discusses another potential use of the token causality relationship: serving as basis for the identification of fault tree analysis.

The problems proposed in the papers are both challenging and useful. They introduce a relatively new research field in system analysis in that, in addition to the fault diagnosis which aims at finding the faulty components responsible for the system failure, the work in this set of papers formally defines the notion of causes and demonstrates the uses of the causality analysis.

1.1 An Informal Overview of Causality Analysis

The informal reasoning about causes is performed by considering observations otherwise than they have been observed. In the literature, the phrases *factual world* and *counterfactual world* are used to describe the observations/facts and the imagined situations during the thought process of causality analysis.

Common requirements that an event A is the cause of another event B include the following.

1. A happens before B in the factual world.
2. In any counterfactual world where A does not happen, B does not happen, *i.e.,* the occurrence of A is necessary for the occurrence of B.
3. In any counterfactual world where A happens, no matter how the system execution changes after the occurrence of A, B would happen, *i.e.,* the occurrence A is sufficient for the occurrence of B.

As will be seen in the reviewed work, not all of the requirements need to hold for the establishment of causal relationship.

The informal notion of causality is inexact in two ways. First, the notion of a counterfactual world is often under-specified. On one hand, it is overwhelming to consider everything involved in a causality analysis session, while on the other hand, key factors that should indeed be considered is sometimes not known to the analysis in the first place. Second, for a causality analysis to be convincing, the counterfactual world should share certain level of similarity with the factual world. The ideal situation would be that the only events of concern are changed while other factors are kept the same. However, this situation is hardly achieved when reason about system behaviors under changed events. The surveyed papers in this report are among the first attempts to the formal definitions of causality. Although not agreeing on every aspect in defining causalities, the papers show several ways to providing a more precise characterization of the notion of token causality.

1.2 Organization

Sections 2, 3, and 4 reviews the papers [8], [2], and [12], respectively. In each of the sections, the basic theory of causality definition and their applications are introduced, and the assumptions, strength, and limitations of each approach are then discussed. In order to gain further insights of the causality analysis problem in general and the differences between the approaches, a horizontal comparison between the approaches is given in Section 5.1. Several general topics with related to the causality analysis in general is discussed in Section 5.2, and the reported is concluded in Section 5.3.

2 Logical Causality in Contract Violation

Given a system execution on which some system property violation happened, the investigation of the cause for this violation depends on the causality definition. In this section, the work by [8] is reviewed as one representative approach to the formal definition of causality.

2.1 Traces, Contracts, and Violations

In [8], a system is modeled as a block diagram where system components are basic blocks sending and receiving events via wire connections. Figure 1 shows an example of an automatic cruise control system. The **SLD** (System Limit Detector), **OR** (Object Recognition), and **ACC** (Adaptive Cruise Control) components are synchronized via a rendezvous *tck* events from the **Clock** component. In addition, events on the same wire are considered synchronous. An interaction is either an event of a single component or a set of events involved in a synchronization between components.

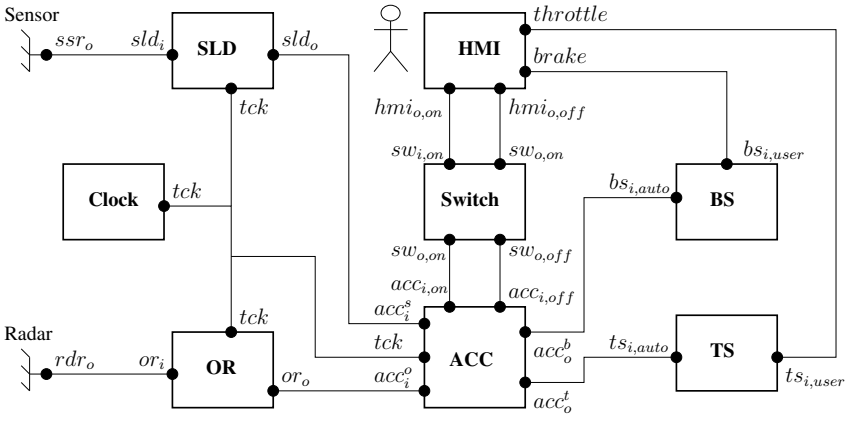


Figure 1: An Example Cruise Control System ([8])

A *trace* for this system is represented as a sequence of interactions between

components with the two aforementioned means of synchronization for events. For example:

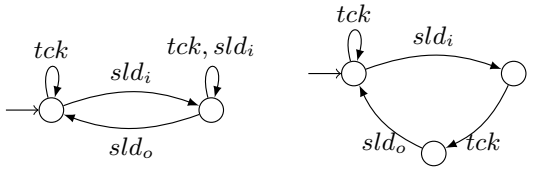
$$\begin{array}{lccccccc} \textbf{SLD}: & sld_i, & tck, & tck, & sld_o, & tck, & tck, & \ldots \\ \textbf{ACC}: & & tck, & tck, & acc_i^s, & tck, & tck, & acc_o^b, \quad \ldots \\ \textbf{Clock}: & & tck, & tck, & & tck, & tck, & \ldots \\ \text{index}: & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \tag{1}$$

is a (partial) trace where the traces from **OR** (Object Recognition), **HMI** (Human-Machine Interface), **Switch**, **BS** (Break System), and **TS** (Throttle System) components are not shown.

The interactions are ordered globally as shown in the index line of Equation (1). For convenience, when a single component is in consideration, its local index of events are used, *e.g.,* for **ACC**, the first local event tck is part of the second interaction globally. For a global trace $\mathbf{tr}$, let π_k be the projection function from the global trace $\mathbf{tr}$ to the local trace tr_k of component B_k, and let $\mathbf{tr}[j]$ be the j^{th} interaction. For a trace tr, let $tr[1..i]$ represent the prefix of tr upto the i^{th} event and let $tr[i]$ represent the i^{th} event on tr. The notation for global traces is analogously defined.

The four tck events represent the elapse of four time units. The sld_o and acc_i^s events in the fourth global interaction represent an interaction between the **SLD** and the **ACC** components with the output sld_o from **SLD** consumed by the **ACC** as input acc_i^s.

System and component properties can be formally expressed as predicates on traces, using a *contract* language based on labeled transition systems (LTSs). An LTS (over an alphabet Σ of events) is a tuple $B = (Q, q^0, \Sigma, \rightarrow)$ with Q a finite set of states, $q^0 \in Q$ an initial state, and $\rightarrow: Q \times \Sigma \times Q$ the transition relation. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ is a pair of LTSs $\mathcal{A}$ and $\mathcal{G}$, known as the assumption and guarantee of the component, respectively. The assumption of a component determines when the component is responsible for its designated responsibilities, specified in the guarantee part of a contract. For example, Figure 2 shows the contract for the **SLD** component. The assumption LTS $\mathcal{A}_{SLD}$ (Figure 2(a)) specifies that, the **SLD** component expects new input (sld_i) from the environment only after an existing one has been processed (sld_o). Under this assumption, the **SLD** component guarantees ($\mathcal{G}_{SLD}$ in Figure 2(b)) that the input is forwarded after exactly one clock tick.



(a) Assumption $\mathcal{A}_{SLD}$ (b) Guarantee $\mathcal{G}_{SLD}$

Figure 2: A Contract for the SLD Component ([8])

A trace tr is *accepted* by an LTS $B = (Q, q^0, \Sigma, \rightarrow)$, denoted $tr \vDash B$, if there exists a sequence of states $q_0, \ldots, q_i, \ldots$ of Q such that $q_0 = q^0$ and for all $i \geq 0$, $(q^i, tr[i+1], q^{i+1}) \in \rightarrow$. A trace tr *satisfies* a contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, denoted $tr \vDash \mathcal{C}$, if $tr[1..i] \vDash \mathcal{G}$ for the maximal position i such that $tr[1..i] \vDash \mathcal{A}$. A trace tr *violates* a contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ if and only if it does not satisfies $\mathcal{C}$, *i.e.,*, there exists an index i such that $tr[1..i] \vDash \mathcal{A}$ but $tr[1..i] \nvDash \mathcal{G}$. In particular, a minimal index can be determined for a violation.

On the example trace for **SLD** in (1), the assumption $\mathcal{A}_{SLD}$ is satisfied, while the guarantee $\mathcal{G}_{SLD}$ is not (detected on the third event of the trace (second occurrence of tck)). Therefore it is said that the contract $\mathcal{C}_{SLD} = (\mathcal{A}_{SLD}, \mathcal{G}_{SLD})$ is violated on the trace.

2.2 Weak, Necessary, and Sufficient Causality

Causality definitions are based on a given trace with a global violation. In the following definitions, it is assumed that the following are given:

- A system with n components B_i, $1 \leq i \leq n$.
- For each B_i, a component contract $\mathcal{C}_i = (\mathcal{A}_i, \mathcal{G}_i)$.
- The system contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, defined over the union of event alphabets of each component.
- A system trace, given in the form of individual component traces tr_i as in Equation (1).

Three notions of causality are defined in [8], namely the weak, necessary, and sufficient causality. The causalities are defined as relationships between prefixes of local traces and the violation of the guarantee of the global contract. (The discussion on this choice is in Subsection 2.5.1.)

2.2.1 Weak Causality

Weak causality formalizes the Lamport causality [13] in the distributed systems setting, *i.e.,* a local violation must precede the global violation to be considered as the cause.

A prefix $tr_k[1..i]$ of the trace tr_k for component B_k is a weak cause of the violation of the global contract $\mathcal{G}$, if

1. $tr_k[1..i]$ violates its local guarantee, *i.e.,* $tr_k[1..i] \nvDash \mathcal{G}_k$, and
2. the last event on the prefix $tr_k[1..i]$ occurs before the global violation, *i.e.,*

$$\exists \mathbf{tr} : \Big(\big(\forall l : \pi_l(\mathbf{tr}) = tr_l \big) \wedge \big(\exists j : |\pi_k(\mathbf{tr}[1..j])| = i \big) \\ \wedge \mathbf{tr}[j](k) \neq \varnothing \wedge j \leq \min\{m \mid \mathbf{tr}[1..m] \nvDash \mathcal{G}\} \Big). \tag{2}$$

2.2.2 Necessary Causality

A prefix $tr_k[1..i]$ of the trace tr_k for component B_k is a necessary cause of the violation of the global contract $\mathcal{G}$, if

1. $tr_k[1..i]$ is a weak cause for the violation of $\mathcal{G}$, and
2. replacing $tr_k[1..i]$ with another prefix that satisfies $\mathcal{G}_k$ while keeping all other traces would not have violated the global guarantee, *i.e.,*

$$\begin{aligned} &\forall \mathbf{tr}' s.t. \left(|\pi_k(\mathbf{tr}')| \geq i \wedge \forall j \in \{1,\ldots,n\} \setminus \{k\} : \pi_j(\mathbf{tr}') = tr_j\right) : \\ &\quad \left(\pi_k(\mathbf{tr}'[1..i] \vDash \mathcal{G}_k \Longrightarrow \mathbf{tr}' \vDash \mathcal{G}\right). \end{aligned} \tag{3}$$

2.2.3 Sufficient Causality

A prefix $tr_k[1..i]$ of the trace tr_k for component B_k is a sufficient cause of the violation of the global contract $\mathcal{G}$, if

1. $tr_k[1..i]$ is a weak cause for the violation of $\mathcal{G}$, and
2. replacing any traces except tr_k with traces satisfying their respective component contracts would still lead to a violation of the global guarantee, *i.e.,*

$$\left(\forall p \in \{1,\ldots,n\} \setminus \{k\} : \pi_p(\mathbf{tr}') \vDash \mathcal{C}_p \wedge \pi_k(\mathbf{tr}') = tr_k\right) \Longrightarrow \mathbf{tr}' \nvDash \mathcal{G}. \tag{4}$$

One thing to note in the sufficient causality definition is that the reconstructed traces should be maximal to ensure that $\mathbf{tr}'$ is long enough to contain the potential violation of $\mathcal{G}$.

2.2.4 An Example of Using the Causality Definitions

Take the trace given in Equation (1) as an example. Assume that the global contract is that the latency Δ between an input either **SLD** or **OR** to an output from **ACC** is less than three, *i.e.,* $\Delta \leq 3$, then the trace shows an violation of the global contract. Also assume that the contract for the **ACC** component is analogous to the one for **SLD**, shown in Figure 2, *i.e.,* the latency is one. According to the causality definitions, both the violations in **SLD** and the **ACC** are necessary for the global violation, but not sufficient.

2.3 Deciding Causality

The causality definitions shown in Subsection 2.2 are all decidable. Despite the use of "forall" and "exists" quantifications over the set of possibly infinite global traces in the causality definitions, the decidability is seen from the facts that

(a) the potential traces satisfying local contracts are exactly characterized by the component contracts,

(b) the potential traces satisfying the condition (in sufficient causality definition) that "the projection of a global trace $\mathbf{tr}'$ onto a component B_k is the same as observed on trace $\mathbf{tr}$" can be encoded as a "linear" LTS accepting exactly the sequence of events on tr_k, and

(c) the causality checking can be transformed into checking language containment relations between the composition of component LTSs that symbolically encodes the set of potential traces and the LTS for the global contract.

A detailed construction for the sufficient causality case can be found in [8].

2.4 Extension: Horizontal Causality

The causality definitions introduced in Subsection 2.2 are for the relationship between a prefix of a local trace and the violation of the *global* guarantee. This is referred to as *vertical* causality by the authors of [8]. By adopting similar formalization, it is possible to define the causality between a prefix of a local trace and the violation of a downstream component. This is referred to as *horizontal* causality.

The main idea to adapt the definitions is to consider the potential changes to the satisfaction of the guarantee $\mathcal{G}_l$ of component B_l (instead of the global guarantee $\mathcal{G}$), when a prefix of the suspected trace tr_k is replaced (for necessary causality) or persisted (for sufficient causality).

2.5 Critiques

2.5.1 Discussion

Assumption on Contract Consistency. One assumption involved in the causality definitions in [8] is that, the contracts $\mathcal{C}_i$ are consistent with the global guarantee, *i.e.,* for all traces $\mathbf{tr}$,

$$\big(\forall i : \pi_i(\mathbf{tr}) \vDash \mathcal{C}_i\big) \Longrightarrow \big(\mathbf{tr} \vDash \mathcal{G}\big). \tag{5}$$

One interpretation for this assumption is that the system is *well-defined*, in that, if there is a violation of the system contract, then there must be at least one violation in the component contract. If this assumption is dropped, it may happen that the system contract is violated, while there are no components to start the investigation of the the causes. These are symptoms that the system design should be reconsidered that the

Assumption on Local Trace Recording. Another assumption used in this work is that, the local traces recorded at each component during run-time does not define a global trace uniquely, in general. This assumption leads to the formulation of the causality in this work to filter from all possible traces that satisfy certain filtering criteria. The straightforward algorithm suggested by the definitions, *i.e.,* generating the set of traces and filtering the ones that satisfy the criteria, is not effective, if applicable at all. This calls for alternative ways to deciding the causality definitions, which the paper does not discuss.

Relata of Causality. Also notice that the causality definitions in this paper establish relationships between prefixes of local traces and violations of system/component contracts. This seems counter-intuitive at first since the common understanding of causality is that, causality is a relation between two events. However, as is also evidenced by the third reviewed paper ([12]), this treatment provides a better viewpoint towards the property violation. Indeed, it is the prefix of a trace leading to a particular event that causes the effect to happen, rather than a single occurrence of an event.

2.5.2 Strength

Separate Treatment of Causality Definitions. One advantage of the approach in this paper is that, the definitions for weak, necessary, and sufficient causalities are separately given. This enables flexible choices when different scenarios are of concern, as it is not always possible or necessary to identify a cause which is both necessary and sufficient.
Component-based Systems. Another advantage of this approach is to take the viewpoint of systems as component-based. This makes it intuitive when assigning blame to the traces produced by the faulty components. Also, it enables the definitions of horizontal causalities discussed in Subsection 2.4, where the traces from individual components are manipulated.

2.5.3 Limitations or Possible Extensions

Causes Being A Set of Local Traces. One possible next work for this paper is to define the notions of causality between a set of local traces violating their contracts and the violation of the global guarantee. For instance, the example in Subsection 2.2.4 has neither **SLD** nor **ACC** as the sufficient cause, because the individual local violations alone does not sufficiently lead to the global violation. However, viewing the two components together as a set, the violation of the global contract is inevitable. In this case, the set of local traces from both **SLD** and **ACC** is a sufficient cause for the global violation of the guarantee of the system contract.
Lack of Efficient Algorithms. Another limitation of this work is the lack of an efficient algorithm to compute the causality definition. According to discussions with one of the authors, the problem of checking causality can be translated to an equivalent problem of model checking [7]. It is known that model checking has an algorithmic complexity of EXPTIME in general [4]. Whether algorithms leveraging existing study in model checking or approximation algorithms for efficient estimation of causality definitions can be developed is worth further examination.

3 Explaining Counterexamples Using Causality

The causality definitions given in [8], summarized and discussed in Section 2, is by no means the only way to define causality. Different trace models, assump-

tions on system execution, means of deciding violations, etc. are all potential factors for a different set of causality definitions. In this section, an alternative approach in [2] will be discussed and contrasted with the one in [8] in detail.

In model checking LTL formulas for Kripke structures [4], when a property φ fails to hold, a counterexample trace is often given by the model checker. Such a counterexample trace is usually minimal, which contains limited information for the understanding of why the property φ is violated. Causality analysis provides a natural explanation of violations of properties on a given counterexample trace, by identifying the set of variable values that are causes for the violation.

3.1 Preliminaries

A list of notations used in the definitions are summarized in Table 1. The introduction of the notions $\langle s, v\rangle$ and thereof is a step towards the formal analysis of traces, *i.e.,* taking states on traces as mathematical objects used in the causality definition.

Notation	Meaning
V	a finite set of (boolean) variables
$\mathcal{K}$	$\mathcal{K} = (S, I, R, L)$; a Kripke structure
S	a finite set of states of $\mathcal{K}$
I	$I \subseteq S$; the set of initial states
R	$R \subseteq S \times S$; the (total) transitivity relation
L	$L : S \to 2^V$; the labeling function; $v \in L(s)$ iff v is true in s
π	$\pi = s_0, s_1, \ldots$ a path in $\mathcal{K}$, if $s_0 \in I$ and $\forall i.(s_i, s_{i+1}) \in R$
$\pi[j..k]$	the subpath of π starting with s_j and ending with s_k
$\pi \cdot \rho$	the concatenation of a finite path π and a path ρ
$\langle s, v\rangle$	the same as state s, but with variable v in consideration
$\langle \hat{s}, v\rangle$	the state derived from s, where v's (boolean) value is flipped
$\pi^{\langle \hat{s}, v\rangle}$	the path derived from π by flipping v's value in state s
A	a set of pairs of the form $\langle s, v\rangle$
$\hat{A}$	the set of states derived from A; $\hat{A} = \{\langle \hat{s}, v\rangle \mid \langle s, v\rangle \in A\}$
$\pi^{\hat{A}}$	the path derived from π by flipping v's value in s, $\forall \langle s, v\rangle \in A$

Table 1: Notations Used for Causality Definition in [2]

LTL formulas are defined with the syntax

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi, \tag{6}$$

where a ranges over the set V of all variables. An occurrence of a sub-formula ψ of φ has positive (negative) polarity if it occurs under an even (odd) number of negations.

Given a Kripke structure $\mathcal{K} = (S, I, R, L)$, a path π in $\mathcal{K}$, a formula φ, a pair $\langle s, v\rangle$ has a *bottom value* for φ in π, if for at least one occurrence of v in φ,

either (a) v has a positive polarity in φ and $v \notin L(s)$; or (b) v has a negative polarity in φ and $v \in L(s)$.

A pairs $\langle s, v\rangle$ with a bottom value is a candidate which would *potentially* make a formula become true if the value of v is flipped in the state s. For example, on a path $\pi = s_0, s_1, s_2, \ldots$ where $L(s_i) = \varnothing$ $(0 \leq i \leq 2)$, a formula $\mathbf{G}p$ has bottom pairs $\langle s_i, p\rangle$ $(0 \leq i \leq 2)$. Flipping the values of p on any of the states s_i $(0 \leq i \leq 2)$ would make the formula $\mathbf{G}p$ a step further towards being true. Similarly, if the formula is $\mathbf{F}p$, then flipping the values of p on any of the states s_i $(0 \leq i \leq 2)$ would make the formula $\mathbf{F}p$ true, if it were not true already. Pairs with bottom values are candidates for causes for property violations, as will be defined in the next subsection. In the worst case, flipping the value v in a pair $\langle s, v\rangle$ has no effect on changing the formula truth value. This happens when the variable has both positive and negative polarities, *e.g.,* $\mathbf{G}(p \wedge \neg p)$.

3.2 Defining Causality

For safety properties expressed in LTL, a counterexample given by model checkers is usually a finite path, expressed as a prefix $\pi[0..k]$ of an infinite path π. For such finite prefixes, the satisfaction relation of an LTL formula φ cannot be always determined. Three-valued semantics are used to describe the satisfaction on finite traces, as defined below [5]. (The subscript $_f$ represents "finitely".)

- The value of φ is **true** (denoted $\pi[0..k] \models_f \varphi$) on $\pi[0..k]$ iff for all infinite computation paths ρ, $\pi[0..k] \cdot \rho \models \varphi$.
- The value of φ is **false** (denoted $\pi[0..k] \not\models_f \varphi$) on $\pi[0..k]$ iff for all infinite computation paths ρ, $\pi[0..k] \cdot \rho \not\models \varphi$.
- Otherwise, the value of φ is **unknown** (denoted $\pi[0..k] \,?\, \varphi$).

The causality definition in [2] takes a different approach than the one in [8]. In the terminology of [8], the causality used in [2] is neither necessary nor sufficient, but rather, a notion of *potential cause*. This is seen from the definitions of *criticality* and *causality* in this paper.

A pair $\langle s, v\rangle$ is *critical* for the failure of φ on $\pi[0..k]$ if $\pi[0..k] \not\models_f \varphi$, but either $\pi^{\langle \hat{s}, v\rangle}[0..k] \models_f \varphi$ or $\pi^{\langle \hat{s}, v\rangle}[0..k] \,?\, \varphi$. In other words, a pair $\langle s, v\rangle$ is critical if changing v's value in s plays a role in changing the formula φ's satisfaction, *i.e.,* changing from **false** to either **true** or **unknown**.

A pair $\langle s, v\rangle$ is a *cause* of the first failure of φ on $\pi[0..k]$ if k is the smallest index such that $\pi[0..k] \not\models_f \varphi$, and there exists a set A of bottom-valued pairs, such that $\langle s, v\rangle \notin A$, and the following hold:

1. $\pi^{\hat{A}}[0..k] \not\models_f \varphi$, and
2. $\langle s, v\rangle$ is critical for the failure of φ on $\pi^{\hat{A}}[0..k]$.

In the definition for a cause, the set A can be chosen to be empty. In this case, it directly follows that if a pair $\langle s, v\rangle$ that is critical for the violation of

φ on $\pi[0..k]$ then it is a cause (provided that the k is the smallest index that $\pi[0..k] \not\models_f \varphi$).

In cases where A is not empty, the definition of cause expresses that, if some values of variables (those in A) on the path $\pi[0..k]$ could be flipped so that additionally flipping the value of the variable v in state s changes the truth value of the formula φ from **false** to either **true** or **unknown**, then the pair $\langle s, v\rangle$ is the cause. Since the pairs in A must be chosen to be bottom-valued pairs, this definition of cause represents a *potential* cause: as long as a variable could potentially play a critical role in changing the truth value of a formula φ, provided some other variables being flipped, it is considered to be a cause.

3.2.1 Examples

A small example is to consider the violation of $\mathbf{G}(a \wedge b \wedge c)$ on the path $\pi = s_0, s_1, s_2, \ldots$ labeled as $(\varnothing)^\omega$ (*i.e.,* every variable is false on every state). The first failure of the formula $\mathbf{G}(a \wedge b \wedge c)$ occurs on state s_0, thus $\pi[0..0]$ is considered as the prefix. The set $A = \{\langle s_0, b\rangle, \langle s_0, c\rangle\}$ can be chosen such that (1.) $\pi^{\hat{A}}[0..0] \not\models_f \mathbf{G}(a \wedge b \wedge c)$ and (2.) $\langle s_0, a\rangle$ is critical for the failure of $\mathbf{G}(a \wedge b \wedge c)$ on $\pi^{\hat{A}}[0..0]$. Therefore $\langle s_0, a\rangle$ is a cause for the first failure of φ on $\pi[0..0]$. Similarly $\langle s_0, b\rangle$ and $\langle s_0, c\rangle$ are causes too.

The authors incorporated the proposed approach into the product IBM Rule-Base PE,[1] where all causes for a violation of a property are displayed with a red dot, along with the counterexample trace generated by the model checker, as shown in Figure 3. For this example, the property under consideration is

$$\begin{aligned}\varphi := \mathbf{G}((\neg\texttt{START} \wedge \neg\texttt{STATUS\_VALID} \wedge \texttt{END}) \rightarrow \\ \mathbf{X}(\neg\texttt{START}\ \mathbf{U}\ (\texttt{STATUS\_VALID} \wedge \texttt{READY}))).\end{aligned} \tag{7}$$

This property expresses that, a new transaction should not begin before the previous transaction has been finalized (indicated by `STATUS_VALID` and `READY` both being true).

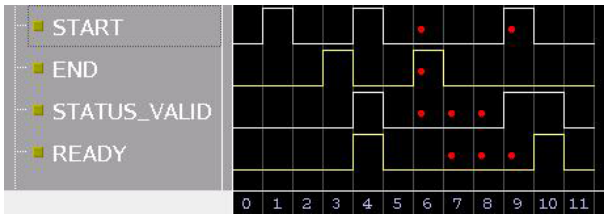


Figure 3: A Counterexample with Explanations

Figure 3 shows a (finite) path $\pi = s_0, \ldots, s_{11}$. The "high" value represents boolean true, and "low" for boolean false, for the corresponding variables. The path prefix $\pi[0..10]$ contains the first violation of property φ. This is because,

[1] `https://www.research.ibm.com/haifa/projects/verification/RB_Homepage/`

for the transaction that has finished at state s_6, a `START` occurs (in state s_9) before the transaction is finalized (in state s_{10}).

The upper-right red dot in Figure 3, overlaid on the pair $\langle s_9, \texttt{START}\rangle$, represents one potential cause for the property violation. Informally, this is interpreted as that, a `START` inadvertently occurs before the finalization of the previous transaction. Should it not occur, the violation to property φ in Equation 7 would not have occurred. This can be formally checked according to the definition by choosing the set A to be the empty set.

The fact that the two red dots in state s_7 (or s_8) are potential causes requires choosing the set A to be non-empty. For $\langle s_7, \texttt{STATUS\_VALID}\rangle$ to be a cause, $\langle s_7, \texttt{READY}\rangle$ has to be included in A, in the definition for cause. Flipping the values of either `STATUS_VALID` or `READY` alone in state s_7 cannot change the truth value of the property, while flipping them both can.

3.3 An Approximation Algorithm

It is proved by the authors of [2] that the exact computation of the set of causes is NP-complete. At the heart of the complexity is the unknown set A when deciding the defined causal relationship. The NP-hardness is proved by polynomially reducing a known NP-complete problem (checking causality in binary causal models [6]) to checking causality defined in this paper. The proof of membership of NP is by showing that the problem can be solved using a non-deterministic Turing machine. Specifically, the algorithm is to non-deterministically choose a candidate for the set A of bottom-valued pairs and check against the definition, where the latter is linear time to the length of the path $|\pi|$ and the depth of the formula $|\varphi|$.

To speed up the computation of the causes, an approximation algorithm is proposed in [2]. The approximation works by ignoring the check whether the set A of pairs makes $\langle s_i, v\rangle$ critical, but rather, computes all the bottom-valued pairs. At the atomic level, $\langle s_i, p\rangle$ ($\langle s_i, \neg p\rangle$, resp.) is considered as a cause for the formula $\neg p$ (p, resp.) on trace $\pi[i..k]$, and the approximation algorithm recursively aggregates causes for any sub-formula that has a value **false** at the current state. The algorithm, detailed in [2], has complexity that is linear in the trace length k and formula depth $|\varphi|$. Cases where the approximation is not exact is discussed in Subsection 3.4.1.

3.4 Critiques

3.4.1 Discussion

Different Viewpoint in Defining Causality. As opposed to the approach in [8], this paper defines a notion of potential cause. As long as there exists a set of changes on the counterexample trace that makes a variable critical on a state, it is considered to be a potential cause. This choice of definition is made by need. The authors intended to design complementary visualization techniques that assist system analyzers understand all potential causes of the property

violation, rather than to identify a culprit that has to be blamed without any excuse. For the latter, the notion of necessary cause in [8] would suit better.

Bottom-value Pairs. An assumption used in this paper is that only bottom-value pairs could be considered as candidates for causes. This assumption suits the particular need of this paper that all potential causes for violations of the property have to be found. If the purpose is to establish the causal relations between occurrences of events and the *satisfaction* of safety properties, then an alternative set of pairs of the form $\langle s, v\rangle$ should be considered.

On Approximation Algorithms. The approximation algorithm in this paper works by computing bottom-valued pairs. This is based on an observation that, discussed at the end of Subsection 3.1, bottom-valued pairs are potential candidates for causes. The definition requires that, upon flipping the value of a set A of pairs, flipping an additional value v in the state s_i is critical, *i.e.,* changes the formula value from **false** to **unknown** or **true**. The approximation algorithm would yield inaccurate result when this does not happen. For example, consider $\varphi = a\,\mathbf{U}\,(b\,\mathbf{U}\,c)$ and a trace $\pi = s_0, s_1, \ldots$ with $L(s_0) = \{a\}$ and $L(s_i) = \varnothing$ for $i \geq 1$. The formula has value **unknown** on $\pi[0..0]$ but value **false** on $\pi[0..1]$, so the analysis of causes is on the prefix $\pi[0..1]$. The pair $\langle s_0, b\rangle$ is a bottom-valued pair and is thus selected by the algorithm. However, it is not a cause since there does not exists a set A of bottom-valued pairs for the given definition.

3.4.2 Strength

Explicit Treatment of Modification of Paths. One advantage of this paper is the explicit representation of manipulations of traces as mathematical objects. Since the analysis of causality is sensitive to the capability to generate counterfactual worlds, the explicit representations, shown in the notations of $\pi^{\langle \hat{s}, v\rangle}$ and $\pi^{\hat{A}}$, make it easy for stating properties on the altered traces.

Incorporation with Model Checkers. Different from the case in [8] where the local traces are obtained from system execution, the traces in this paper is obtained from counterexamples by model checkers. Incorporating the approach of providing causal explanations of property violations back to the utility of trace visualization could improve the system analysis experience.

3.4.3 Limitations or Possible Extensions

More Meaningful Visualization. The visual representation of the causes overlaid on the counterexample trace, shown in Figure 3, shows limited information compared to the definition. According to the definition, when a pair $\langle s, v\rangle$ is identified as a cause for a property φ on a prefix of a path $\pi[0..k]$, a corresponding set A of pairs is also identified. The intended interpretation of the cause $\langle s, v\rangle$ is that, when the variables in pairs in A have flipped their values on the finite trace $\pi[0..k]$, the property's truth value remains **false**, until the value for variable v occurs in state s is changed. The visualization technique shown in this paper simply plots all the pairs $\langle s, v\rangle$ that are causes and ignores

the companion set A for each cause. This treatment is less helpful, compared to an alternative one that actually displays the companion set A for each cause pair $\langle s, v \rangle$. Currently, showing this additional information is impossible for the approximation algorithm in this paper, since the information of the companion set A is simply lost during the process of the approximation computation.

4 From Probabilistic Counterexamples via Causality to Fault Trees

In addition to providing explanations to violations of system property on a given counterexample trace, information presented in a set of counterexample traces can be extracted in order to obtain additional insight to the system analysis. The work presented in [12] is one such example, where the authors base their analysis on a set of given counterexamples, identify the common patterns on the traces, and extract such information in to a fault tree [17]. For each of the identified pattern, the probability that such pattern can occur can be estimated based on the set of given counterexample traces. The approach of obtaining fault trees presented in this work is fully automatic, which complements most of the existing methodologies which require human expertise.

4.1 Defining Causality

4.1.1 Traces, Feature Extraction, and Modeling of Faults

In this work, a trace is represented as a linear Kripke structure over a set AP of event variables, where the labeling relation is a function. $L(s_{i+1}) = a_i$ indicates that the event that the variable a_i represents just happened and lead the system to transition from s_i to s_{i+1}. To formally reason about patterns of traces, the authors defined an *event order logic* (EOL). The syntax for a formula φ in EOL is as follows.

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \mathbin{\mathbb{A}} \varphi \tag{8}$$

The $\mathbb{A}$ symbol is called the sequential conjunction. A formula $\varphi \mathbin{\mathbb{A}} \psi$ is satisfied when φ is satisfied on a trace and ψ is then satisfied later on the trace. The semantic interpretation of EOL on a trace is slightly different than that of normal temporal logics. Given $\pi = s_0, s_1, \ldots, s_n$ a finite trace, the semantics of an EOL formula on a state s_i of π is defined inductively as follows.

$$\begin{aligned}
&s_i \vDash a \text{ iff } a \in L(s_i).\\
&s_i \vDash \neg\varphi \text{ iff not } s_i \vDash \varphi.\\
&s_i \vDash \varphi \wedge \psi \text{ iff } \exists j,k : i \le j,k \le n.(s_j \vDash \varphi \text{ and } s_k \vDash \psi).\\
&s_i \vDash \varphi \vee \psi \text{ iff } \exists j,k : i \le j,k \le n.(s_j \vDash \varphi \text{ or } s_k \vDash \psi).\\
&s_i \vDash \varphi \mathbin{\mathbb{A}} \psi \text{ iff } \exists j,k : i \le j \le k \le n.(s_j \vDash \varphi \text{ and } s_k \vDash \psi).
\end{aligned} \tag{9}$$

The path π satisfies an EOL formula φ iff $\exists i.s_i \vDash \varphi$. Simply put, an EOL formula expresses features on traces, while satisfaction of the features does not have to be at the beginning of the trace.

Traces can be compared according to the events that occur on a trace. Given two traces π_1 and π_2, π_2 is said to contain π_1, denoted $\pi_q \subseteq \pi_2$ if for all variables a, $\pi_1 \vDash a \Rightarrow \pi_2 \vDash a$; π_2 is said to sequentially contain π_1, denoted $\pi_q \dot{\subseteq} \pi_2$ if for all variables a_1 and a_2, $\pi_1 \vDash a_1 \wedge a_2 \Rightarrow \pi_2 \vDash a_1 \wedge a_2$. The $\subseteq$ and $\dot{\subseteq}$ relations establishes two partial orders. The set of minimal traces can be determined given a set Π of traces, *i.e.,* $\min_{\subseteq}(\Pi) := \{\pi \in \Pi \mid \forall \pi' \in \Pi.\pi' \subseteq \pi \Rightarrow \pi' = \pi\}$, or $\min_{\dot{\subseteq}}(\Pi) := \{\pi \in \Pi \mid \forall \pi' \in \Pi.\pi' \dot{\subseteq} \pi \Rightarrow \pi' \doteq \pi\}$.

Each finite minimal trace is viewed as a feature in this work. The trace corresponds to a sequence of events happening in order, and is equivalent to a formula in EOL. The correspondence is straightforward: for the trace $\pi = s_0, \ldots, s_n$, where $L(s_{i+1}) = a_{i+1}$ $(0 \leq i < n)$, the EOL is simply $a_1 \wedge \cdots \wedge a_n$.

Faults are modeled as state formula in a usual temporal logic. In this work, the PRISM model checker [11] is used since the underlying system models are continuous-time Markov chains. For a given fault modeled as a state formula t, the authors use the PRISM model checker to check the formula $\mathcal{P}_{=?}(\mathbf{true}\,\mathbf{U}\,t)$. This leads the model checker to yield a set Π_C of counterexample traces together with the probability for each trace. The features that are shown in the set Π_C of counterexample traces is $\min_{\subseteq}(\Pi_C)$.

The goal of the causality analysis in this work is to identify which features in the set $\min_{\subseteq}(\Pi_C)$ is the cause for the fault t, according to the causality definition below.

4.1.2 Actual Cause

The theory of *actual cause* (synonym with token causality) used in this paper is an adaptation of the one in [9]. The main idea is to (a) distinguish endogenous and exogenous variables, *i.e.,* the ones that are and are not considered to be suspects of causes; and (b) identify three criteria to deciding causality based on the manipulations on endogenous variable values. For Part (b), the three criteria are *actual observation*, *necessity and sufficiency*, and *minimality*. The details of the theory is thoroughly discussed in [9], while the adaption used in this paper is discussed below.

Given a trace feature expressed as an EOL formula φ and a fault condition expressed as a state formula t, three conditions are needed for the determination of the *actual cause* between φ and t.

AC1: There exists a trace π_1 in Π_C such that $\pi_1 \vDash \varphi$ and $\exists i.\pi_1[i] \vDash t$. Let val_1 be the variable valuation on trace π_1.

AC2: Partition the set AP of variables into disjoint sets Z and W, where the variables in Z are the ones occurring in either φ or t. Let $X \subseteq Z$ be a subset. There exists a valuation val_2 such that

1. Changing the values of the variables in X and W from val_1 to val_2 changes t from true to false.
2. Setting the values of the variables in S from val_1 to val_2 has no effect on t as long as the values of the variables in X are kept at the values defined by val_1.

AC3: The set X is minimal, *i.e.,* no subset of X satisfies the conditions in AC1 and AC2.

The three conditions defined above reflect the criteria defined in [9], respectively. First, for the feature expressed in φ to be a cause for t, they must appear on at least one possible traces. Second, condition AC2.1 expresses the necessity condition, where the values of variables in X must be as in val_1 for the property t to be true; while condition AC2.2 expresses the sufficient condition, where no matter what values the variables other than X change to, the property t is still satisfied. The minimality condition in AC3 is essentially satisfied by construction, when the candidate for the fault condition t is chosen from the set $\min_{\subseteq}(\Pi_C)$. This is because,

Notice that a salient aspect of difference compared to the causality definitions in [8] is that, the choice of val_1 and val_2 is existentially quantified, as opposed to being universally quantified in [8]. The distinction between the two choices is discussed in Subsection 4.3.1.

4.1.3 A Traingate Example

Consider a traingate example where events in the system are modeled with variables observation, *i.e.,* Gc: gate closed; Go: gate open; Ta: train approaching the road; Tc: train crossing the road; Tl: train leaving the road; Ca: car approaching the gate; Cc: car crossing the tracks; and Cl: car has left the gate. A trace is then represented as a sequence of the occurrences of the events. The sequence $\langle Ta, Ca, Cc, Gc, Tc\rangle$ is a potential hazardous situation, since the car enters the tracks while a train is also crossing the road.

To formally reason about the causality on this trace, the fault condition is defined as $Cc \wedge (\neg Cl\, \mathbf{U} Gc)$. When this fault condition is satisfied, a hazard occurs. A feature extracted from the sequence $\langle Ta, Ca, Cc, Gc, Tc\rangle$ can be represented as an EOL formula $Cc \curlywedge Gc$. The definition of causality is satisfied when the set X is chosen to be $\{Tc, Cc\}$, Z to be $\{Cl\}$, and W to be $\{Ta, Ca\}$. As long as the events Tc and Cc occur as observed, there is a situation (when Cl does not happen) that no matter what values of other variables are, the fault condition still persists. Note that the subset $X = \{Cc, Cl, Tc\}$ could be an actual cause too, but it is not minimal.

4.2 Fault Tree Construction

4.2.1 Definitions

A fault tree is an effective method to represent the relationship between system hazards and their causes. Each node of a fault tree represents an event, indi-

cating a potential hazard situation. The children of the node are understood as the causes for the hazard event at the node. The relationships between a node's children are specified by the gate type of the node. Usual gate types are or-gate, and-gate, and priority and-gate. An or-gate node indicates that the node would fail if any of its children fails. An and-gate node indicates that the node would fail if all of its children fail. A priority and-gate node indicates that the node would fail if all of its children fail at a specific order (visually represented left-to-right).

4.2.2 Fault Tree Construction Methodology

Of particular interest to this paper, the or-gate and priority and-gate are used. The construction of fault trees from counterexamples is based on the decomposition of the set Π_C of all counterexample traces according to the set of minimal traces induced by Π_C. That is, for any minimal trace π in the set $\min_{\subseteq}(\Pi_C)$, it defines a *causality class*: $\Pi_\pi = \{\pi' \in \Pi \mid \pi \subseteq \pi'\}$. Each causality class represents a possible reason for the fault condition to occur. The union of all causality classes are the set Π_C of all counterexample traces. However, the causality classes are not necessarily disjoint, since on a single trace in Π_C, it may present multiple features that causes the failure.

In fault tree construction, the causality classes are the children of the root or-gate of the fault condition. Each child, child is then represented by a priority and-gate to represent the order according to which the events must appear for the fault condition to occur. A sample fault tree is presented in Figure 4. This example is from the analysis of an airbag system in [1].

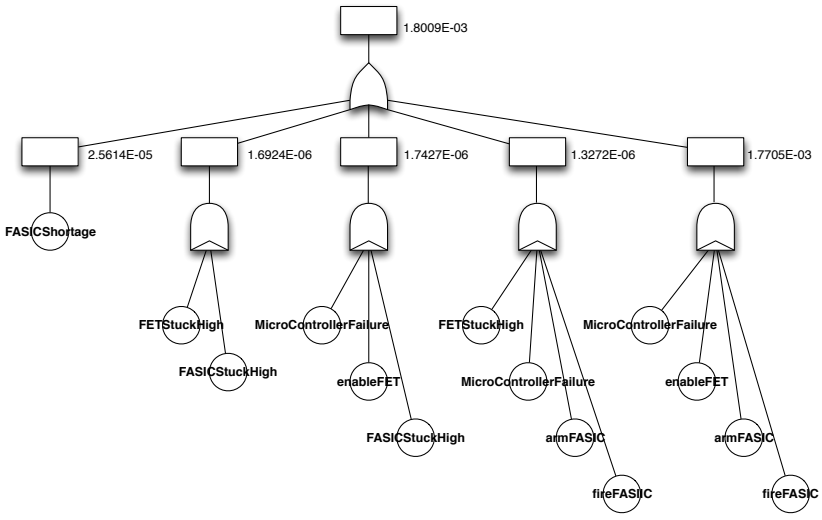


Figure 4: Fault Tree of the Airbag System (Reproduced from [12])

In this fault tree, the system safety property is that the airbag system should

not be deployed when there is no crash situation detected. Explanations to each hazard are not the scope of this report, and can be found in [1]. The structure of this automatically computed fault is that, each branch under the or-gate is an event sequence (indicated by the priority and-gate) that can lead to an inadvertent deployment of the airbag system.

The computation of the probabilities of each causality class is by summing up the individual probabilities of all the counterexample traces belonging to the causality class. The computation of the probability of the or-gate is by the equation $Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$. Since in fault trees, the probability values are relatively small, the term $Pr(A \cap B)$ is often omitted and an estimate $Pr(A \cup B) \approx Pr(A) + Pr(B)$ is used.

4.2.3 Approximation Algorithm for Deciding Causality and Fault Tree Construction

As has been proved in [6], causality definitions that are variants of [9] has algorithmic complexity of NP-complete. The approximation algorithms shown in this paper works by relaxing the search for a subset X of Z, but only deciding whether the set Z satisfies the causality definitions. The rationale for choosing this approximation is due to the preprocessing step where the minimal counterexample traces are determined. The algorithm takes the set Π_C of counterexample traces as choices of valuation in condition AC1, whereas the set of all other maximal simple traces Π_G are choices of the valuation in condition AC2. The algorithm for deciding one cause is then quadratic to the size of the set Π_C of counterexample traces. The process for assigning probabilities is by scanning through the set of counterexample traces, which has a complexity linear to $|\Pi_C|$. Therefore, the overall complexity for the computation of fault trees from counterexample traces is cubic to $|\Pi_C|$.

4.3 Critiques

4.3.1 Discussion

Assumptions There are several implicit assumptions used in this paper. First, it is assumed that all counterexamples are finite traces. While this assumption simplifies the definitions and algorithm of the approach, it is not discussed in the paper whether counterexamples for liveness properties are handled the same way. The computation of minimal traces would essentially lose the looping information that is usually associated with the counterexamples.

A second assumption is that, the set Π_G of "good" traces are "obtained at no additional cost". This assumption is not true, since the set Π_G is not the simple complement of the set $\Pi \setminus \Pi_C$ (where Π is the set of all finite traces built on the same set of atomic variables). Rather, Π_G is the set of the "maximal simple traces" out of $\Pi \setminus \Pi_C$. This set indeed requires additional computation. In addition, the set Π_G may be infinite, under which case the algorithm provided in the paper would not finish.

Using Different Logics for Causes and Effects. In this paper, the logic for expressing features of the traces is EOL, while the fault conditions are expressed in usual temporal logics. This choice may seem counter-intuitive, while in fact, the causality analysis is not affected by the use of different logics. The only requirement is that, the formula of the causes and effects could be checked for truth values on a common observation, in this case, a counterexample trace. A similar treatment where different logics are used can be found in [18].

On the Choice of Quantifiers in Counterfactual Trace Construction. In the definition of causality in this work (as well as in [9] from which the definition is adapted), the necessary and sufficient checks require the *existence* of an alternative valuation of the variables which satisfies the conditions defined in AC2.1 and AC2.2. In stark contrast, the necessary and sufficient definitions given in [8] require that *for all* counterfactual traces, certain conditions should hold (see Section 2.2).

None of the surveyed work in this report provided discussion on the rationale on the choice of the quantifiers. An understanding of this choice might be obtained from the perspectives of the two sets of papers. In [8], the authors want to bring the notion of *blame* used in the court setting into the engineering domain, so that once a component is identified as a (necessary or sufficient) cause, there should be no excuse for the component to escape the blame. Therefore, the quantifiers are chosen to be universal, where all possible system executions are considered. In the work of [9], the initial goal is to formally characterize the informal human reasoning of causes [10]. In such cases, a human rarely reasons about causality by enumerating all possible scenarios, but rather would provide an alternative *explanation* where the effect would be as observed if certain causal events did not happen. Thus, the quantifiers is chosen to be existential in the work of [9, 12]. It is not easy or necessary, if possible at all, to determine whether one definition is advantageous over another. One should choose the proper definition of causality based on the application scenario under analysis.

4.3.2 Strength

Full Automation in Computation and Fault Tree Construction. Not discussed in detail in this report, the causality analysis and fault tree construction presented in this work is fully automated by the authors. This enables the authors to apply the analysis to relative big case studies such as an embedded control system [16], a train odometer controller [3], and an airbag system [1], the biggest of which generates 738 counterexample traces but with a computation time within minutes.

4.3.3 Limitations or Possible Extensions

Modeling Absence of Events. As the example in Section 4.1.3 shows, the event that the car has left the tracks being missing can potentially also be a cause. However, the EOL formula $Cc \wedge \neg Cl \wedge Tc$ is not a proper way to express the facts that the car leaving event Cl is missing in between a car crossing and

train crossing event. This formula expresses that, on *one* of the states in between the Cc and Tc events, the Cl event does not appear, not *all*. A general problem associated with the EOL shown in this paper is the inability of expressing the absence of events in an interval on a trace. Therefore, the causality classes identified in this paper are all sequences of occurrences of events, rather than the non-occurrence of an event as part of a sequence.

A natural idea is to extend the EOL for such purposes. In a follow-up work by the same authors [14], the EOL is extended with a ternary operator $\wedge_{<}\wedge_{>}$, where $\varphi_1 \wedge_{<} \psi \wedge_{>} \varphi_2$ means that the event ψ occurs in between φ_1 and φ_2 on a trace. With this operator, the feature that the the car leaving event does not occur in between the car crossing and train crossing events is expressed as $Cc \wedge_{<} \neg Cl \wedge_{>} Gc$.

Need for More Suitable Toolchains. Though the authors managed to glue several existing tools together to analyze causality and build the fault trees, many smaller adjustment are made at each step. For example, the choice of a continuous-time Markov chain as the system model is affected by the choice available in the PRISM model checker, which is in turn chosen due to the availability of counterexample generator called DiPro. Affected by the choice of modeling language, the logic for describing system properties are limited too. In this paper, the temporal logic used is the continuous stochastic logic, but with only a fragment where the $\mathcal{P}_{\leq p}$ operator is used. A potential next step of the approach is to limit the modeling languages as well as specification languages, and design and implement an easy-to-use environment for causality analysis and fault tree generation, similar to the integration of the analysis into an existing product as done in [2].

5 Discussion and Conclusion

5.1 Comparisons Across the Approaches

As a brief summary, the causality analysis problems that are investigated in this paper follow a common pattern: defining a formal notion of causality, investigate the computation cost and approximation algorithm, and apply the causality definitions into applications, either directly or indirectly. Table 2 lists comparisons of the three approaches surveyed in the paper.

	[8]	[2]	[12]
Relata: Cause	Prefix of local trace (§2.2)	Pair $\langle s, v\rangle$ (§3.1)	Feature (§4.1.1)
Relata: Effect	Global property violation	Property violation	Property violation
Trace Model	Local recordings (§2.1)	Counterexample trace	Counterexample trace
Property Spec.	Not fixed	LTL	EOL/CSL
Causality	Weak, necessary, and sufficient	Potential cause	Actual cause
Complexity: Exact	EXPTIME	NP-complete	NP-complete
Complexity: Approx.	–	$O(k+\|\varphi\|)$	$O(\|\Pi_C\|^3)$
Application	Cruise control	Model checker integration	Airbag control system

Table 2: Comparison Across the Approaches

5.2 Discussion

5.2.1 Counterfactuals as a Tool for Causality Reasoning

It is assumed in all the work surveyed that causality reasoning is based on a formalization of counterfactual reasoning [15]. There are philosophical debates in the literature on whether counterfactuals are fundamental concepts than causes, or counterfactuals are only an apparatus for testing causality. The surveyed work circumvented the discussions by simply adopting the latter assumption for granted.

5.2.2 Using Three-valued Logics

In [2], a three-valued semantics for the LTL is used for the interpretation of an LTL formula on a finite trace. The rationale is that, when a trace has not finished, the truth value of an LTL formula cannot be decided. However, in [12], a similar temporal logic is used to describe system properties, but the approach has not mention of three-valued semantics. The reason for the latter is that, the fault conditions are expressed only as state formula and their truth values can indeed be decided on a single state. The use of a temporal logic in [12], is simply because the limitation to the PRISM tool so that a simple state formula is transformed to a temporal logic formula. In general, three-valued semantics are necessary only when path formulas are involved in the decision of a formula value on a finite trace.

5.2.3 Subjectivity of Causality Definitions

One reason that the causality definitions in the surveyed papers are different is that the underlying system models and property specification languages vary. However, as discussed in Section 4.3.1, when considering counterfactual worlds, either existential or universal quantifications can be used. There are essentially more available choices when the quantitative measures of the sets are considered. This observation in fact shows a lack of consensus in the definition of causality, regardless of the underlying system model of specification language.

The lack of consensus essentially makes the causality definitions subjective. It is up to the analyzer to define and use a notion of causality that is suitable for the system under analysis. The choice of causality defnitions may be subject to the expertise of the analyzer. Moreover, except for comparing with human intuition for causality, there does not exist a better way of deciding the appropriateness of the chosen causality definition. This is an inevitable outcome of the subjectivity of causality definitions. In algorithmic causality analysis, it is of importance that the involved parties agree on a commonly accepted definition causality.

5.2.4 First Violations

Both in [8] and [2], only the causality between the traces/events and the *first* violation of a property on a trace is considered. In [2], the authors' explanation

is that "the first failure is the most interesting one for the user." This may be the case for counterexamples from model checking, while in an monitoring setting, it is not always the case that the system halts after the first violation. Exploration of later failures would be helpful to gain insights to the system as well.

A challenge in dealing with later failures is that, it is not clear how to segment an observed trace. Some options include but not limited to: (a) always analysis from the start; (b) restart analysis after previous failure; and (c) keeping a fixed window of history events. Study on the validation of the choices would be useful to enhance the application of causality analysis.

5.3 Conclusion

The causality analysis has seen its applications in system analysis in the papers surveyed in this report. When a system property violation occurs, causality analysis helps establish the causal relationship between the local components, traces, and/or events with the violation. The formal definitions of causality make it possible for the approximation algorithms to estimate the causality, which greatly improves the applicability of causality analysis. Some challenges in causality analysis, including incorporating tool support, enriching the modeling languages, developing efficient algorithms, as well as the use of causality analysis in more case studies, lie ahead as interesting research problems.

References

[1] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue. Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples. In *Quantitative Evaluation of Systems, 2009. QEST '09. Sixth International Conference on the*, pages 299 –308, sept. 2009.

[2] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. In *Computer Aided Verification*, pages 94–108. Springer, 2009.

[3] E. Böde, T. Peikenkamp, J. Rakow, and S. Wischmeyer. Model based importance analysis for minimal cut sets. In *Automated Technology for Verification and Analysis*, pages 303–317. Springer, 2008.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[5] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification*, pages 27–39. Springer, 2003.

[6] T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142(1):53–89, 2002.

[7] G. Gossler. Personal communication.

[8] G. Gössler, D. L. Métayer, and J.-B. Raclet. Causality analysis in contract violation. In *Runtime Verification*, volume 6418 of *Lecture Notes in Computer Science*, pages 270–284. 2010.

[9] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British Journal for the Philosophy of Science*, 56(4):843–887, December 2005.

[10] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British Journal for the Philosophy of Science*, 56(4):889–911, December 2005.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[12] M. Kuntz, F. Leitner-Fischer, and S. Leue. From probabilistic counterexamples via causality to fault trees. In F. Flammini, S. Bologna, and V. Vittorini, editors, *Computer Safety, Reliability, and Security*, volume 6894 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin Heidelberg, 2011.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[14] F. Leitner-Fischer and S. Leue. On the synergy of probabilistic causality computation and causality checking. In *SPIN*, 2013.

[15] D. Lewis. *Counterfactuals*. Wiley-Blackwell, 2nd edition, 2001.

[16] J. Muppala, G. Ciardo, and K. S. Trivedi. Stochastic reward nets for reliability prediction. *Communications in reliability, maintainability and serviceability*, 1(2):9–20, 1994.

[17] W. Vesely. *Fault tree handbook*. Nuclear Regulatory Commission, 1987.

[18] S. Wang, A. Ayoub, B. Kim, G. Gossler, O. Sokolsky, and I. Lee. A causality analysis framework for component-based real-time systems. In *Proceedings of RV'13, the 4th International Conference on Runtime Verification*, 2013.