Causality for General LTL-definable Properties

Georgiana Caltais
georgiana.caltais@uni-konstanz.de

Sophie Linnea Guetlein
linnea.guetlein@uni-konstanz.de

Stefan Leue
stefan.leue@uni-konstanz.de

Department for Computer and Information Science
University of Konstanz, Germany

In this paper we provide a notion of causality for the violation of general Linear Temporal Logic (LTL) properties. The current work is a natural extension of the previously proposed approach handling causality in the context of LTL-definable safety properties [20, 19]. The major difference is that now, counterexamples of general LTL properties are not merely finite traces, but infinite lasso-shaped traces. We analyze such infinite counterexamples and identify the relevant ordered occurrences of causal events, obtained by unfolding the looping part of the lasso shaped counterexample sufficiently many times. The focus is on LTL properties from practical considerations: the current results are to be implemented in QuantUM, a tool for causality checking, that exploits explicit state LTL model checking.

1 Introduction

The importance and complexity of software driven systems is steadily increasing. Software plays a central rôle in daily used objects, such as computers and mobile phones, but also in other areas, for example medical systems, aircraft and automobiles. Particularly in these latter areas, software failures may entail major environmental harm and/or serious injuries of humans. Software systems whose malfunction has such serious consequences are also called safety-critical systems. This paper addresses methods to analyze models of such systems for the detection of ordered sequences of events that can be considered causal for the malfunctioning of such a system. Of particular importance in this setting is the identification of actual causes, *i.e.*, sequences of events that are indispensable for the malfunctioning to occur and not just mere “noise” in the system execution.

Model-checking [1] is a formal verification technique for systematically checking whether certain temporal requirements are satisfied by a system model. Given a state-based model M of the considered system and a property specification φ, model checkers return a counterexample if φ is not satisfied by M. This counterexample typically is an execution trace that includes a violation of the property φ and can be used to understand the cause of the property violation and to fix the problem. However, counterexamples can be very long and often contain numerous events that are of no relevance to the violation of φ. Furthermore, there can be a very large number of counterexample traces in M that all lead to the violation of φ.

In precursory work [20, 19], model checking of reachability properties has been extended to causality checking by considering all traces in a model that violate a system safety property expressed by a reachability property φ. Inspired by the actual cause conditions defined in the Structural Equation Model (SEM) for causality in systems [15, 14] the work in [20, 19] defines actual cause conditions on ordered sequences of events that correspond to computations in a Transition System model.

In this paper we provide a notion of causality for the violation of general Linear Temporal Logic (LTL) properties. Counterexamples of such properties can always be represented by ω-regular expressions of the form uv^{ω}, where u and v are regular expressions [22]. Counterexamples of this type are also often referred to as "lasso-shaped". They consist of an initial path fragment that can witness the violation of a "something bad never happens"-kind of property (*i.e.*, a safety property) followed by a loop that can witness the violation of a "something good eventually happens"-kind of property (*i.e.*, a liveness property). In particular, for the case of "pure" safety properties, the lasso ends in a self-loop state where the property is violated.

Consider, for an example, the behaviour of an elevator system as follows:

Example 1 (Elevator)**.** *The elevator can commute between three floors (0,1,2). On each floor there is a button that can be pressed in order to call the elevator. Whenever a button on some floor is pressed, the elevator will try to go to that floor immediately. If two buttons are pressed, the elevator will go to the lower floor first. We use the identifier E_i to denote the event "elevator is on floor i", and B_i to denote "button on floor i is pressed", for $i \in \{0,1,2\}$.*

A liveness property is, for example, that whenever a button on the second floor is pressed, the elevator will go there eventually. Assume we are interested in finding those sequences of events that lead to a hazard in which the button on the second floor is pressed, but the elevator never arrives at the second floor. A corresponding counterexample is a lasso-shaped execution in which $B2$ can be observed, whereas $E2$ occurs neither along the initial path fragment after $B2$, nor in the loop.

In this paper, causality checking for general definable LTL properties is done by identifying the counterexample of a temporal property with a so-called Event Order Logic formula. This formula encodes the relevant ordered occurrences of causal events, and is obtained by unfolding the loop-counterexample "sufficiently many times".

Related work. The idea of exploiting counterexamples as a debugging aid, in order to understand what determined a certain system failure, has been addressed by other works as well. We refer, for instance, to the results in [2] that uses the notion of causality in [15, 14] and provides the user a visual explanation of the failure by marking causes as red dots along the counterexample trace. In [2], causes for the violation of an LTL property are computed via an over-approximation algorithm. For another example, we refer to the work in [17], where errors in system models are elicited from similar counterexamples witnessing the violation of liveness properties.

It is certain that imposing minimality conditions on the size of causal explanations is desirable. In our work, we adapt the approach from [15, 14], and formalise a notion of causality that is minimal with respect to the number of events (*i.e.*, system actions) it encompasses. In a similar spirit, [23] proposes a methodology for computing shortest counterexamples for symbolic model checking of so-called LTL with past formulae. Other "nice to have" properties of causality such as, compositionality, for instance, were addressed in [10, 9, 11, 12, 5].

For a more comprehensive survey on principles, algorithms and applications of counterexample analysis, we refer to [6].

Contributions. In this paper we introduce a notion of causality with respect to the violation of general LTL properties. This is an extension of the work in [19], where causality was handled in the context of safety LTL properties. Our main contributions include an adaptation of the so-called Event Order Logic (EOL) in [19] in order to enable compact (finite) descriptions of what caused the violation of a system failure. The proposed notion of causality incorporates a series of properties to be satisfied by the EOL formulae characterising property violations. A notion of soundness and completeness depending on a complete enumeration of all bad traces (*i.e.*, counterexamples) and good traces is also established. We show that causality in the sense of [19] is equivalent with causality in the current paper, for the case of safety LTL properties.

Structure of paper. Section 2 briefly introduces the formal framework for analysing counterexamples witnessing the violation of LTL system properties in the context of transition system models. The corresponding extension of EOL to describe such counterexamples is provided in Section 3. Section 4 introduces the proposed notion of causality. In Section 5 we discuss soundness and completeness of our approach. Section 6 draws the conclusions and provides pointers to future work.

2 Preliminaries

In this section we introduce the formal framework for analysing counterexamples witnessing he violation of Linear Temporal Logic [16] system properties in the context of transition systems.

Definition 1 (Transition Systems (TS's))**.** *A TS is a tuple $T = (S, Act, \to, I, AP, L)$, where S is a finite set of states, Act is a set of actions, $\to \subseteq S \times Act \times S$ is a transition relation, $I \subseteq S$ is a set of initial states, AP is a set of atomic propositions, and $L : S \to \mathcal{P}(AP)$ is a function associating to states in S a set of atomic propositions in AP.*

For $(s,\alpha,s') \in\ \to$ we also write $s \xrightarrow{\alpha} s'$. In the remainder of this paper, for each $\alpha \in Act$, we consider an atomic proposition or event variable *$a_\alpha \in AP$ such that: given $s' \in S$, it holds that $a_\alpha \in L(s')$ whenever there exists $s \in S$ with $s \xrightarrow{\alpha} s'$.*

We define an execution, or execution trace *of T, as a possibly infinite sequence $\sigma = s_0\alpha_1 s_1\alpha_2 s_2\ldots$ with $s_0 \in I$ and $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $i \geq 0$. Moreover, for all $i \geq 0$ we write $\sigma[i\ldots]$ to represent the execution $s_i\alpha_{i+1}s_{i+1}\ldots$. Additionally, for all $0 \leq i < j$ we write $\sigma[i..j]$ to represent the finite execution $s_i\alpha_{i+1}\ldots\alpha_j s_j$.*

Given an execution trace $\sigma' = s'_0\alpha'_1 s'_1\alpha'_2\ldots$, we write $s_0\alpha_1 : \sigma'$ as a shorthand for the execution trace $\sigma = s_0\alpha_1 s'_0\alpha'_1 s'_1\alpha'_2\ldots$. For simplicity of notation, we sometimes write $\alpha'_1\alpha'_2\ldots$, or $a_{\alpha'_1}a_{\alpha'_2}\ldots$ to equivalently represent σ'.

Remark 1. *In this paper, we consider TS's without terminal states, i.e., TS's for which all executions are infinite. Observe that this is not a limitation. Finite executions ending in a terminal state s can be straightforwardly extended to infinite executions via a transition $s \xrightarrow{\lambda} s_\lambda$ such that s_λ has a self-loop labelled λ, i.e., $s_\lambda \xrightarrow{\lambda} s_\lambda$.*

We further focus on formalising properties of TS's. A safety property can be seen as a requirement that some bad event never happens. More formally, a property P_{safe} is a safety property if and only if every path, or execution that *violates* P_{safe} has a finite prefix that can not be extended to a path satisfying P_{safe}. Intuitively, this means that if a safety property is violated, this violation already happens after the model has passed a finite sequence of states and after this finite sequence the violation is unrecoverable. Consequently, if we want to check whether a safety property is satisfied or not, it suffices to only look at finite paths of the system. A well known approach for reasoning on the violation of safety properties is model-checking implemented via simple depth first search (DFS) algorithms [1]. These algorithms check whether starting from an initial state we can find a path to some state of the model where the bad event specified by the safety property happens. If such a path can be found, the property is violated.

A liveness property requires that some good event eventually happens. It follows that when checking whether a liveness property is satisfied or not it does *not* suffice to only look at finite execution fragments of the system. Orthogonally to the model-checking of safety properties, reasoning on the violation of liveness properties is performed via nested depth first search (NDFS) [7]. This algorithm searches for an infinite path in the model, such that the good event described by the liveness property does not hold along that path. If such a path can be found, the property is violated.

Linear time (LT) properties can be expressed in terms of a safety and a liveness property, based on the *Decomposition Theorem* 3.37 in [1]. Consequently, reasoning on the violation of LT properties requires an NDFS-based approach.

We further provide a brief overview on *Linear Temporal Logic* (LTL) [16] – a formalism to describe system properties. Intuitively, LTL formulae range over the atomic proposition *true*, that holds in any state of a transition system and, respectively, over atomic propositions a satisfied within a state s whenever the labelling function indicates so. Recursively, LTL formulae are defined as *disjunctions* ($\shortmid$), *conjunctions* (&) and *negations* ($\sim$) of formulae. The *next* (X) operator indicates the satisfiability of a property starting with the "next" state, whereas the *until* (U) operator indicates the satisfiability of a formula ϕ_1 all the time until a formula ϕ_2 is finally satisfied. The *eventually* ($\Diamond$) and *generally* ($\Box$) operators indicate the satisfiability of a formula "at some point" in the future and, respectively, "all the time".

Definition 2 (Linear Temporal Logic (LTL))**.** *LTL formulae over the set AP of atomic propositions are built according to the following grammar:*

$$\phi, \phi_1, \phi_2 ::= true \mid a \mid \phi_1 \shortmid \phi_2 \mid \phi_1 \mathbin{\&} \phi_2 \mid\sim \phi \mid X\phi \mid \phi_1 \; U \; \phi_2 \mid \Diamond\phi \mid \Box\phi \qquad (a \in AP)$$

LTL formulae are interpreted over transition systems without terminal states $T = (S, Act, \rightarrow, I, AP, L)$*. Let* $\sigma = s_0\alpha_1 s_1 \alpha_2 s_2 \ldots$ *be an execution trace in* T*. The following hold:*

- $\sigma \vDash true$
- $\sigma \vDash a$ *iff* $a \in L(s_0)$
- $\sigma \vDash\sim \phi$ *iff not* $\sigma \vDash \phi$
- $\sigma \vDash \phi_1 \shortmid \phi_2$ *iff* $\sigma \vDash \phi_1$ *or* $\sigma \vDash \phi_2$
- $\sigma \vDash \phi_1 \mathbin{\&} \phi_2$ *iff* $\sigma \vDash \phi_1$ *and* $\sigma \vDash \phi_2$
- $\sigma \vDash X\phi$ *iff* $\sigma[1..] \vDash \phi$
- $\sigma \vDash \phi_1 \; U \; \phi_2$ *iff* $\exists k \geq 0 : \sigma[k..] \vDash \phi_2$ *and* $\forall 0 \leq i < k : \sigma[i..] \vDash \phi_1$
- $\sigma \vDash \Diamond\phi$ *iff* $\exists k \geq 0 : \sigma[k..] \vDash \phi$
- $\sigma \vDash \Box\phi$ *iff* $\forall k \geq 0 : \sigma[k..] \vDash \phi$

We write $(\varphi_1 \rightarrow \varphi_2)$ *as a syntactic sugar for* $(\sim \varphi_1 \shortmid \varphi_2)$*, and* $(\varphi_1 \leftrightarrow \varphi_2)$ *as a syntactic sugar for* $((\sim \varphi_1 \rightarrow \varphi_2)\&(\sim \varphi_2 \rightarrow \varphi_1))$*.*

We say that a transition system T satisfies a LTL formula ϕ*, written as* $T \vDash \phi$*, if and only if for all executions* σ *of T it holds that* $\sigma \vDash \phi$*.*

If a system model, or TS in our setting, does not satisfy some given LTL-definable property P, there must be an execution witnessing the violation of the property. Such executions are called *counterexamples*. For safety properties, counterexamples are *finite* execution fragments that start in an initial state of the system and lead to an undesired state where "something bad" actually happens. For liveness properties, counterexamples must be *infinite* executions, because every finite path can still be extended to a path satisfying the liveness property and does, therefore, not suffice as an example for the violation of the property. An infinite path that violates a liveness property is lasso-shaped.

3 Event Order Logic

The work in [19] introduces the so-called Event Order Logic (EOL). Intuitively, formulae in EOL are used to express causality classes for counterexamples. Causality classes can be seen as generalized counterexamples. A causality class represents several counterexamples, all leading to the property violation in the "same way". Such counterexamples may only differ in some other events that are not essential for the property violation.

In the case of liveness properties, our counterexamples must be lasso shaped, that is, they contain a loop at the end. The sequence "$E0,B2,(B1,E1,B0,E0)^{\omega}$" is a counterexample of $\varphi := \Box(B2 \to \Diamond E2)$ in the elevator model. In words, the elevator is stuck between the ground floor and the first floor, as $(B1,E1,B0,E0)^{\omega}$ indicates that the sequence $(B1,E1,B0,E0)$ keeps repeating forever. We can say that the cause of the property violation is that the buttons on the ground floor and the first floor are pressed repeatedly and between that, the elevator never has the chance to go to the second floor.

We see that the cause of a liveness property violation must consist of some events happening at the beginning and then some other events happening again and again (in the loop). Hence, in what follows, we propose an extension of EOL in [19] with so-called *infinite* formulae to express infinite causal behaviours.

The work in [19] introduces two kinds of EOL formulae: *simple* and *complex*. Intuitively, simple EOL formulae, usually denoted by ϕ, are built over event variables a_{α} that are atomic propositions witnessing the execution of actions α *at some point in the future*. The satisfiability of atomic propositions is different within the frameworks of EOL and LTL: the latter assumes satisfiability with respect to the initial state of a trace, whereas the former has an "eventually" component. This difference is formalized in Definition 4 providing the semantics of EOL.

Similarly to the case of LTL, simple EOL formulae are inductively defined using *negation* ($\neg$), *conjunction* ($\wedge$) and *disjunction* ($\vee$). As a consequence of the observation above, formulae of shape $\phi_1 \wedge \phi_2$ (respectively, $\phi_1 \vee \phi_2$) read as: eventually ϕ_1 will hold and (respectively, or) eventually ϕ_2 will hold.

For technical reasons related to the semantics of the aforementioned infinite EOL formulae, we split the complex EOL formulae in [19] into: *I-complex* and *G-complex*, respectively. We refer to Remark 2 for a more detailed explanation.

I-complex formulae, usually denoted by ψ, include simple EOL formulae, *conjunctions* ($\wedge$) and *disjunctions* ($\vee$) of I-complex formulae. An I-complex formula $\psi_1 \dot{\wedge} \psi_2$ has an "ordered-and"-like semantics and reads: first ψ_1 holds and then ψ_2. Last, but not least, an I-complex formula $\psi_1 \dot{\wedge}_{<} \phi \dot{\wedge}_{>} \psi_2$ reads: first ψ_1 holds, then ψ_2 holds and in between the "interval" determined by the satisfiability of ψ_1 and ψ_2 the simple EOL formula ϕ holds all the time.

G-complex formulae, usually denoted by θ, range over I-complex formulae and encompass two more temporal operators: $\dot{\wedge}_{]}$ that has an "until"-like semantics, and $\dot{\wedge}_{[}$ that has an "after"-like semantics. More precisely, $\phi \dot{\wedge}_{]} \theta$ reads: θ will hold at some point in the future and until then, the simple EOL formula ϕ holds all the time. Orthogonally, $\theta \dot{\wedge}_{[} \phi$ reads: at some point θ holds, and after that, ϕ holds all the time.

Observe that the simple and, respectively, G-complex formulae in this paper have the same expressive power as the simple and, respectively, complex EOL formulae originally proposed in [19].

To express infinite causal behaviour, we extend the EOL in [19] with the so-called *infinite* formulae, usually denoted by ξ. These are formulae built over the new logical symbol $\dot{\wedge}^{\omega}$. For a G-complex formula θ and an I-complex formula ψ we write $\theta \dot{\wedge}^{\omega} \psi$ to express that first θ holds and then ψ happens infinitely many times.

Formally, as the new EOL we obtain the following:

Definition 3 (Extended Event Order Logic (EOL) – Syntax)**.** Simple EOL formulae *over a set $\mathcal{A}$ of event variables are formed according to the following grammar:*

$$\phi, \phi_1, \phi_2 ::= \top \mid a_\alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \qquad (a_\alpha \in \mathcal{A}).$$

Complex EOL formulae *are of two kinds:*

- I-complex EOL formulae*, formed according to the following grammar:*

$$\psi, \psi_1, \psi_2 ::= \phi \mid \psi_1 \dot{\wedge} \psi_2 \mid \psi_1 \dot{\wedge}_< \phi \dot{\wedge}_> \psi_2 \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2$$

  where ϕ is a simple EOL formula.

- G-complex EOL formulae*, formed according to the following grammar:*

$$\theta ::= \psi \mid \phi \dot{\wedge}_] \theta \mid \theta \dot{\wedge}_[\phi$$

  where ϕ is a simple EOL formula and ψ an I-complex EOL formula.

Infinite EOL formulae *are formed according to the following grammar:*

$$\xi ::= \theta \dot{\wedge}^\omega \psi$$

where θ is a G-complex EOL formula and ψ is an I-complex formula.

We want an infinite execution σ to satisfy an infinite EOL formula $\xi = \theta \dot{\wedge}^\omega \psi$ if and only if (a) the events in θ occur in σ in the order specified by θ, and (b) the events of ψ occur in σ in the order specified by ψ, infinitely many times.

As an example, consider the following execution σ in a TS:

$$\begin{tikzcd} \sigma = s_0 \arrow[r, "\alpha_1"] & s_1 \arrow[r, bend left, "\alpha_2"] & s_2 \arrow[l, bend left, "\alpha_3"] \end{tikzcd} \tag{1}$$

It is easy to see that σ satisfies the formula

$$\xi = a_{\alpha_1} \dot{\wedge}^\omega (a_{\alpha_2} \dot{\wedge} a_{\alpha_3})$$

where a_{α_i} is the event variable corresponding to α_i for $i \in \{1,2,3\}$. We see that σ contains a finite part $\sigma_1 = s_0 \xrightarrow{\alpha_1} s_1$ determining the event variable a_{α_1} to occur, and a finite part in the loop $\sigma_2 = s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1$ where a_{α_2} and a_{α_3} occur. Thus, an intuitive approach to decide whether an infinite execution trace satisfies an infinite EOL formula $\xi = \theta \dot{\wedge}^\omega \psi$ is to split σ into an initial final trace σ_1 and finite trace σ_2 inside of the loop, and check if σ_1 satisfies θ and if σ_2 satisfies ψ, respectively.

The following example shows that it is not enough to split the lasso shaped execution trace σ into a first part σ_1 that contains all states up to the loop and a second part σ_2 that consists of the loop executed only once. Consider the execution σ in (1) and the EOL formula $\xi' = a_{\alpha_1} \dot{\wedge} a_{\alpha_3} \dot{\wedge}^\omega (a_{\alpha_3} \dot{\wedge} a_{\alpha_2})$. Our execution σ also satisfies ξ' because in σ the event a_{α_1} happens before a_{α_3} and after that, a_{α_2} happens after a_{α_3} infinitely many times. Hence, in this case, the finite traces guaranteeing the satisfiability of ξ' are as follows:

$$\sigma_1 = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \tag{2}$$

is obtained by concatenating the sequence in σ up to the loop, with one unfolding of the loop, whereas

$$\sigma_2 = s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \tag{3}$$

is the unfolding of the loop twice.

By following a similar pattern, consider σ in (1) and the EOL formula

$$\xi'' = a_{\alpha_1} \dot{\wedge} a_{\alpha_3} \dot{\wedge} a_{\alpha_2} \dot{\wedge} a_{\alpha_3} \dot{\wedge} a_{\alpha_2} \dot{\wedge}^{\omega} (a_{\alpha_2} \dot{\wedge} a_{\alpha_3} \dot{\wedge} a_{\alpha_2} \dot{\wedge} a_{\alpha_2} \dot{\wedge} a_{\alpha_2}).$$

We want σ to satisfy this formula as well, but σ_1 in (2) and σ_2 in (3) do not satisfy their corresponding EOL formulae in ξ''. We have to extend σ_1 until the loop has been executed three times, while σ_2 is defined by unfolding the loop four times. Hence, we get:

$$\begin{array}{rcl} \sigma_1 & = & s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \\ \sigma_2 & = & s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_1 \end{array} \tag{4}$$

Intuitively, σ satisfies an infinite EOL formula $\xi = \theta \dot{\wedge}^{\omega} \psi$ whenever σ can be split into two finite executions σ_1 and σ_2, "large enough" to satisfy θ and ψ, respectively.

Remark 2. *As can be seen in Definition 3, we choose to classify the complex EOL formulae in [19] into I-complex formulae ψ and G-complex formulae θ. This is because we want to allow only interval-like formulae ψ as right-hand side of the $\dot{\wedge}^{\omega}$ operator. The occurrence of $\phi \dot{\wedge}_{]} \psi$ or $\psi \dot{\wedge}_{[} \phi$ in a cycle does not make sense unless $\psi = \phi$, case in which the corresponding formulae ξ can be equivalently expressed in terms of formulae $\theta \dot{\wedge}^{\omega} (\phi \dot{\wedge} .. \dot{\wedge} \phi)$, where $\phi \dot{\wedge} .. \dot{\wedge} \phi$ stands for finite ordered conjunctions ($\dot{\wedge}$) of simple formulae ϕ.*

The EOL semantics is translated to the setting of general LTL- properties and infinite loop-traces as follows:

Definition 4 (EOL – Semantics)**.** *Let $T = (S, Act, \to, I, AP, L)$ be a transition system without terminal states. Let ϕ, ϕ_1, ϕ_2 be simple EOL formulae, let ψ, ψ_1, ψ_2 be complex EOL formulae, let θ be a G-complex EOL formula, and let ξ be an infinite EOL formula. Let $\mathcal{A}$ be a set of event variables and let a_α, a_{α_i} range over arbitrary event variables in $\mathcal{A}$.*

The satisfiability of EOL formulae ($\vDash_e$) is defined over execution traces $\sigma = s_0 \alpha_1 s_1 \alpha_2 \ldots$ in T.

For simple EOL *formulae we define:*

- $\sigma \vDash_e \top$*, i.e.,* $\top$ *(true) is trivially satisfied by all traces*
- $\sigma \vDash_e a_\alpha$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e a_\alpha$ *iff* $\exists 0 < j \le r : s_{j-1} \xrightarrow{\alpha} s_j$
- $\sigma \vDash_e \neg\phi$ *iff not* $\sigma \vDash_e \phi$
- $\sigma \vDash_e \phi_1 \wedge \phi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \phi_1 \wedge \phi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \phi_1$ *and* $\sigma[0..r] \vDash_e \phi_2$
- $\sigma \vDash_e \phi_1 \vee \phi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \phi_1 \vee \phi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \phi_1$ *or* $\sigma[0..r] \vDash_e \phi_2$

For I-complex EOL *formulae we define:*

- $\sigma \vDash_e \psi_1 \dot{\wedge} \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1 \dot{\wedge} \psi_2$ *iff*
  $\exists 0 < j \le k < r : \sigma[0..j] \vDash_e \psi_1$ *and* $\sigma[k..r] \vDash_e \psi_2$
- $\sigma \vDash_e \psi_1 \dot{\wedge}_< \phi \dot{\wedge}_> \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1 \dot{\wedge}_< \phi \dot{\wedge}_> \psi_2$ *iff*
  $\exists 0 < j \le k < r : \sigma[0..j] \vDash_e \psi_1$ *and* $\sigma[k..r] \vDash_e \psi_2$ *and* $\forall l$ *s.t.* $j \le l < k : \sigma[l..l+1] \vDash_e \phi$
- $\sigma \vDash_e \psi_1 \wedge \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1 \wedge \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1$ *and* $\sigma[0..r] \vDash_e \psi_2$
- $\sigma \vDash_e \psi_1 \vee \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1 \vee \psi_2$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \psi_1$ *or* $\sigma[0..r] \vDash_e \psi_2$

For G-complex EOL *formulae we define:*

- $\sigma[i..r] \vDash_e \phi \wedge_] \theta$ *iff* $\exists i \leq j < r : \sigma[j..r] \vDash_e \theta$ *and* $\forall k$ *s.t.* $i \leq k < j : \sigma[k..k+1] \vDash_e \phi$
- $\sigma[i..r] \vDash_e \theta \wedge_[\phi$ *iff* $\exists i < j \leq r : \sigma[i..j] \vDash_e \theta$ *and* $\forall k$ *s.t.* $j \leq k < r : \sigma[k..k+1] \vDash_e \phi$
- $\sigma \vDash_e \phi \wedge_] \theta$ *iff* $\exists 0 \leq r : \sigma[r..] \vDash_e \theta$ *and* $\forall j$ *s.t.* $0 \leq j < r : \sigma[j..j+1] \vDash_e \phi$
- $\sigma \vDash_e \theta \wedge_[\phi$ *iff* $\exists 0 < r : \sigma[0..r] \vDash_e \theta$ *and* $\forall j$ *s.t.* $j \geq r : \sigma[j..j+1] \vDash_e \phi$

Let $\sigma = s_0\alpha_1 s_1 \alpha_2 \dots s_l \alpha_{l+1} s_{l+1} \dots \alpha_{l+m} s_{l+m} \alpha_{l+m+1} s_{l+1} \alpha_{l+2} \dots$ be an infinite execution trace of T with a loop consisting of m states and starting with s_{l+1}. Let $\sigma_2^j = \sigma[l..l+j*m-1]$ be the unfolding of the loop for j times.

For infinite EOL formulae $\xi = \theta \wedge^\omega \psi$ we define:

- $\sigma \vDash_e \xi$ *iff* $\exists i \geq 0, j \geq 0 : \sigma_1 = \sigma[0..i]$ *and* $\sigma_2^j = \sigma[l..l+j*m-1]$ *and* $\sigma_1 \vDash_e \theta$ *and* $\sigma_2^j \vDash_e \psi$.

Definition 5 (EOL Formulae over Executions)**.** *Let $\sigma = s_0\alpha_1 s_1 \alpha_2 \dots s_l \alpha_{l+1} \dots s_{l+m-1} \alpha_{l+m} s_l \alpha_{l+1} \dots$ be an infinite execution trace of T with a loop consisting of m states and starting with s_l. The EOL formula over σ is defined as: $\xi_\sigma := a_{\alpha_1} \wedge \dots \wedge a_{\alpha_l} \wedge^\omega (a_{\alpha_{l+1}} \wedge \dots \wedge a_{\alpha_{l+m}})$.*

The following definition will give us a possibility of comparing two EOL formulae. Whenever we have an EOL formula ξ_1 and extend it to an EOL formula ξ_2 by adding some events to the formula, the set of executions that satisfy the EOL formula ξ_2 will be a subset of those executions that satisfy the EOL formula ξ_1. Intuitively, this holds as in ξ_2 we have more constraints on the represented execution traces. Therefore, for two infinite EOL formulae ξ_1 and ξ_2 we will use the notation $\xi_1 \subseteq \xi_2$ to express that every execution σ that satisfies ξ_2 also satisfies ξ_1. In that case, ξ_1 can be seen as a generalized form of ξ_2.

Definition 6 (EOL Formulae Subset Relationship)**.** *Let $\xi_1$andξ_2 be infinite EOL formulae.*

- $\subseteq$: $\xi_1 \subseteq \xi_2$ *iff every execution σ that satisfies ξ_2 also satisfies ξ_1. Intuitively, this means that the set of events in ξ_1 is a subset of the events in ξ_2.*
- $\subset$: $\xi_1 \subset \xi_2$ *iff* $\xi_1 \subseteq \xi_2$ *and* $\xi_1 \neq \xi_2$.

As an example we consider the EOL formulae $\xi_1 = E0 \wedge B1 \wedge E1$ and $\xi_2 = E0 \wedge B1 \wedge B2 \wedge E1$. In every execution σ that satisfies ξ_2, the events $E0$, $B1$ and $E1$ will happen one after the other (but there can be other events happening between them). Therefore, every execution σ that satisfies ξ_2 also satisfies ξ_1. Hence, it holds that $\xi_1 \subseteq \xi_2$.

4 Causality for general LTL-definable properties

In this section, we formally define the notion of actual causality (AC) for general LTL-definable properties. The definition follows its counterpart in [19]. The latter is an adoption of the actual causality in [15], to the context of concurrent systems. Next, we provide a brief reminder of the causal setting in [15].

In [15], systems under analysis are formalized as structural equation models. Intuitively, structural equations are used to describe causal influence of variables in the system. The set of all variables is partitioned into the set U of *exogenous* variables that are irrelevant with respect to the causal effect, and the set V of *endogenous* variables that are considered to have a meaningful, potentially causal effect. The set $X \subseteq V$ contains all events that jointly might represent a cause. A signature $\mathcal{S}$ is defined as a tuple $(\mathcal{U}, \mathcal{V}, \mathcal{R})$, where $\mathcal{U}$ is a finite set of exogenous variables, $\mathcal{V}$ is a finite set of endogenous variables, and $\mathcal{R}$ associates with every variable $Y \in \mathcal{U} \cup \mathcal{V}$ a nonempty set $\mathcal{R}(Y)$ of possible values for Y. A structural equation model over a signature $\mathcal{S}$ is defined in [15] as tuple $M = (\mathcal{S}, \mathcal{F})$, where $\mathcal{F}$ associates with each variable $X \in \mathcal{V}$ a function denoted F_X that defines the values of all variables in X given the values of all other variables in $\mathcal{U} \cup \mathcal{V}$. Consider a structural equation model $M = (\mathcal{S}, \mathcal{F})$, a vector $\vec{X}$ of variables in

$\mathcal{V}$, and vectors $\vec{x}$ and $\vec{u}$ of values for the variables in $\vec{X}$ and $\mathcal{U}$. $M_{\vec{X}\leftarrow\vec{x}}$ denotes the structural equation model for which variables in $\vec{X}$ are set to $\vec{x}$. Given a signature $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$, a formula of the form $X = x$, for $X \in \mathcal{V}$ and $x \in \mathcal{R}(X)$, is called a primitive event. A basic causal formula over $\mathcal{S}$ is one of the form $[Y_1 \leftarrow y_1, ..., Y_k \leftarrow y_k,]\varphi$ where φ stands for the effect, or hazard, and $Y_1, ...Y_k$ and X are variables in $\mathcal{V}$. The formula $[Y_1 \leftarrow y_1, ..., Y_k \leftarrow y_k,]\varphi$ is abbreviated as $[\vec{Y} \leftarrow \vec{y}]\varphi$. Intuitively, $[\vec{Y} \leftarrow \vec{y}]\varphi$ states that φ holds in a setting in which the values of the variables in $\vec{Y}$ are set to the values in $\vec{y}$. A causal formula ψ is a Boolean combination of basic causal formulae. We write $(M, \vec{u}) \models_{SM} \psi$ whenever ψ is true in the structural model M, given the context defined by $\vec{u}$. Additionally, $\vec{X} = \vec{x}$ stands for a conjunction of primitive events of the form $X_1 = x_1 \wedge .... \wedge X_k = x_k$.

An actual cause with respect to the hazard, or effect φ is defined in [15] as follows:

Definition 7 (Actual cause [15])**.** *$\vec{X} = \vec{x}$ is an actual cause of φ in $(M, \vec{u})$ if the following three actual cause conditions (AC) hold:*

AC1: $(M, \vec{u}) \models_{SM} (\vec{X} = \vec{x}) \wedge \varphi$.

AC2: *There exists a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}, \vec{w})$ of the variable in $(\vec{X}, \vec{W})$ such that:*

1. $(M, \vec{u}) \models_{SM} [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}']\neg\varphi$
2. $(M, \vec{u}) \models_{SM} [\vec{X} \leftarrow \vec{x}, \vec{W} \leftarrow \vec{w}', \vec{Z}' \leftarrow \vec{z}^*]\varphi$ *for all subsets $\vec{Z}'$ of $\vec{Z}$*

AC3: *$\vec{X}$ is minimal, in the sense that no subset of $\vec{X}$ satisfies conditions AC1 and AC2.*

Intuitively, in [15], condition AC1 states that there is a setting in which both the cause and the effect occur. AC2(1) expresses a necessity condition. It says that for $\vec{X} = \vec{x}$ to be a cause of φ, there must be a setting $\vec{x}'$ such that if $\vec{X}$ is set to $\vec{x}'$, φ would not have occurred. However, as stated in [15], AC2(1) might be too permissive as it allows to change the values of the variables in both X and W. Hence, the change of φ form true to false could be caused by a change of a variable in X or W. The set W enables expressing so-called "contingent dependencies". For an intuition, consider two events "Alice presses button B2" and "Bob presses button B2" that enable the elevator to reach the second floor of a building. We say that the elevator reaching the second floor depends on Alice pressing the button, under the contingency that Bob did not press the button. AC2(2) constrains AC2(1) by keeping the values of the variables in X at their original values and only changing the variables in W. AC2(2) corresponds to a sufficiency condition. Intuitively, setting $\vec{X}$ to $\vec{x}$, guarantees that φ holds. The minimality condition in AC3 ensures that only those elements that are essential with respect to φ are part of the cause.

Actual causality in the context of TS's and LTL-definable properties is defined as an adoption of [15] to the setting of concurrent systems, in the spirit of [19]. In our setting, $\neg\varphi$ represents the hazard, or the effect.

Definition 8 (Causality for LTL)**.** *Let $T = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states. An EOL formula ξ is considered a cause for the violation of the LTL specifiable property φ, if the following conditions are satisfied:*

- *AC1: There exists an infinite execution σ in T such that $\sigma \models_e \xi$ and $\sigma \not\models \varphi$*
- *AC2(1): There exists an infinite execution σ in T such that $\sigma \not\models_e \xi$ and $\sigma \models \varphi$*
- *AC2(2): For all infinite executions σ'' in T with $\sigma'' \models_e \xi$ it holds that $\sigma'' \not\models \varphi$.*
- *AC3: The EOL formula ξ is minimal, i.e., there does not exist an EOL formula ξ' with $\xi' \subset \xi$ that also satisfies conditions AC1 and AC2.*

AC1 above resembles its counterpart in Definition 7 in the sense that it identifies a setting σ that satisfies both the cause ξ and the effect φ. AC2(1) entails a necessity condition that identifies a setting witnessed by $\neg\xi$ in which the violation of φ would not occur. We fully formalise necessity as a completeness result in Theorem 1, Section 5. AC2(2) is a sufficiency result, stating that satisfying the cause ξ is enough to guarantee the violation of φ. AC3 is the minimality condition which states that no true subset of ξ satisfies conditions AC1 and AC2. Intuitively, ξ is in the "most general form possible". Moreover, note that our definition of causality does not employ a notion of "contingency". This is because our approach to causality checking is based on a complete exploration of the traces within a TS model and enables the explicit identification of all potential causes.

Example 2. *If we want to show that $\xi = E_0 \wedge B2 \wedge^{\omega} (B1 \wedge E1 \wedge B0 \wedge E0)$ is causal with respect to the violation of the LTL property $\varphi = \Box(B2 \rightarrow \Diamond E2)$, we need to show that AC1, AC2 and AC3 are fulfilled for ξ.*

Consider the infinite execution:

$$\sigma = E_0 B_2 (B_1 E_1 B_0 E_0)^{\omega} \tag{5}$$

Informally, (5) states that when at floor E_0, after pressing button B_2, only alternations of actions press B_i and reach E_i are possible, where $i \in \{1,2\}$. Note that E_2 is never reached, even if B_2 was pressed.

Moreover, consider a behaviour in which the elevator stops at floor E_2 infinitely many times after B_2 being pressed once:

$$\sigma' = E_0 B_2 (B_1 E_1 E_2 B_0 E_0)^{\omega} \tag{6}$$

At this point, we can infer the following:

- *AC1 is satisfied as, for σ in (5), $\sigma \vDash_e \xi$ and $\sigma \nvDash \varphi$ hold.*
- *AC2(1) holds for $\sigma'' = E_0 (B_2 E_2 B_1 E_1)^{\omega}$, for instance.*
- *AC2(2) is not fulfilled as, for instance, for σ' in (6), $\sigma' \vDash_e \xi$ and $\sigma' \vDash \varphi$ hold.*

Hence, ξ is not causal, as it does not prohibit the occurrence of E_2.

We observe that in order to compute causes for a property violation according to Definition 8 it is not sufficient to start with an execution trace σ that is a counterexample for the property φ, build the EOL formula ξ_σ over σ and generalize it (in the sense of Definition 6) until it satisfies conditions AC1-AC3. Example 2 shows that also the non-occurrence of events can be causal for the violation of a general LTL-property. We further introduce a method to compute the events over σ whose non-occurrence is causal for the violation of φ, and encode this information within the cause ξ.

We proceed by first defining a valuation function with respect to a set of event variables $\mathcal{M}$. This function maps an execution trace σ to the subset of event variables of $\mathcal{M}$ occurring in σ.

Definition 9 (Valuation Function)**.** *Given a transition system $T = (S, Act, \rightarrow, I, AP, L)$ and a finite set of event variables $\mathcal{M} = \{a_{\alpha_1}, \ldots, a_{\alpha_n}\}$, we define the valuation function $val_{\mathcal{M}}$ as a function on the set of execution traces of T to the set $\mathcal{P}(\mathcal{M})$ – the powerset of $\mathcal{M}$. Let σ be an execution trace of T. Then:*

$$val_{\mathcal{M}}(\sigma) := \{a_\alpha \in \mathcal{M} : \sigma \vDash_e a_\alpha\}.$$

Definition 10 (Non-Occurrence of Events)**.** *Let $T = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states, φ an LTL-definable property, σ an execution trace over T with $\sigma \nvDash \varphi$ and ξ_σ the EOL formula built over σ. Let $\mathcal{A}$ be the set of event variables, let $\mathcal{Z}$ be the set of event variables occurring in ξ_σ and let $\mathcal{W} := \mathcal{A} \backslash \mathcal{Z}$. We say that that Q is the subset of event variables whose non-occurrence in σ is causal for the property violation of φ, if*

1. *ξ_σ satisfies AC1 and AC2(1).*
2. *There exists an execution trace σ'' with $\sigma'' \vDash_e \xi_\sigma$, $val_{\mathcal{Z}}(\sigma) = val_{\mathcal{Z}}(\sigma'')$, $val_{\mathcal{W}}(\sigma) \neq val_{\mathcal{W}}(\sigma'')$ and $\sigma'' \vDash \varphi$.*
3. *$Q \subseteq W$ is the smallest set s.t. for all execution traces σ'' with $\sigma'' \vDash_e \xi_\sigma$ and $val_{\mathcal{Z}}(\sigma) = val_{\mathcal{Z}}(\sigma'')$ and $val_Q(\sigma) = val_Q(\sigma'') = \varnothing$ we have $\sigma'' \nvDash \varphi$.*

As above, let σ be a counterexample for the property φ, let ξ_σ be the EOL formula built over σ and assume ξ_σ satisfies the first two conditions of the definition above.

We can now compute the subset Q and determine the location of the event variables $a_\alpha \in Q$ in the EOL formula $\xi_{\sigma''}$ built over a σ'' which we choose as in condition 2 above. We then compare ξ_σ to $\xi_{\sigma''}$ and prohibit the occurrence of a_α in ξ_σ in the same locations as they occur in $\xi_{\sigma''}$. We repeat the procedure for all σ'' as in 2 above, and build ξ_σ in an incremental fashion. This way we obtain a new EOL formula ξ_σ that also satisfies condition AC2(2).

If, based on Definition 6, there is a generalization ξ' of ξ_σ that also satisfies AC1 and AC2, we replace ξ_σ by ξ'. We repeat this procedure until ξ_σ is in the most general form possible and, therefore, also satisfies AC3. Since ξ_σ satisfies AC1-AC3 by construction, it is a cause for the violation of φ.

In the context of Example 2, from traces σ'' satisfying condition 2, we obtain intermediate formulae ξ_σ of shape:

$$\begin{array}{ll}
\sigma'' = E_0B_2E_2(B_1E_1B_0E_0)^\omega & \xi_\sigma = (E0 \mathbin{\dot\wedge} B2 \mathbin{\dot\wedge}_{[} \neg E2) \mathbin{\dot\wedge}^{\omega} (B1 \mathbin{\dot\wedge} E1 \mathbin{\dot\wedge} B0 \mathbin{\dot\wedge} E0) \\
\sigma'' = E_0B_2(B_1E_1E_2B_0E_0)^\omega & \xi_\sigma = (E0 \mathbin{\dot\wedge} B2 \mathbin{\dot\wedge}_{[} \neg E2) \mathbin{\dot\wedge}^{\omega} (B1 \mathbin{\dot\wedge} E1 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} B0 \mathbin{\dot\wedge} E0) \\
\sigma'' = E_0B_2(B_1E_1B_0E_2E_0)^\omega & \xi_\sigma = (E0 \mathbin{\dot\wedge} B2 \mathbin{\dot\wedge}_{[} \neg E2) \mathbin{\dot\wedge}^{\omega} (B1 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} E1 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} B0 \mathbin{\dot\wedge} E0) \\
\sigma'' = E_0B_2(B_1E_1E_2B_0E_0)^\omega & \xi_\sigma = (E0 \mathbin{\dot\wedge} B2 \mathbin{\dot\wedge}_{[} \neg E2) \mathbin{\dot\wedge}^{\omega} (B1 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} E1 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} B0 \mathbin{\dot\wedge}_{<} \neg E2 \mathbin{\dot\wedge}_{>} E0) \\
 & \dots
\end{array} \tag{7}$$

On top of the formula incrementally derived as in (7), the repeated generalisation procedure entails the EOL formula:

$$\xi_\sigma = (B2 \mathbin{\dot\wedge}_{[} \neg E2) \mathbin{\dot\wedge}^{\omega} (\neg E2) \tag{8}$$

which is semantically equivalent with $B2 \mathbin{\dot\wedge} \neg E2$. Observe that ξ_σ satisfies AC1 for σ in (5) and AC2. ξ_σ also satisfies AC3, because every superset of ξ_σ will either violate AC1 or AC2. Thus, the EOL formula ξ_σ satisfies AC1-AC3 .

Conditions AC1-AC3 do not imply that the order of the occurring events is causal. Whether the order of events occurring in an EOL formula ξ that satisfies AC1-AC3 is causal or not, can be checked by the following Order Condition (OC). Note that ξ can be causal even if the OC is not satisfied.

Definition 11 (Order Condition (OC))**.** *Let $T = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states. Let σ be an infinite execution trace violating an LTL-definable property φ. Let ξ be the EOL formula built over σ. Let $\mathcal{Z}$ be the set of event variables occurring in ξ and let $\mathcal{W} := \mathcal{A} \backslash \mathcal{Z}$. Consider a set of pairs of event variables over $Y \subseteq \mathcal{Z}$:*

$$\{(a_{\alpha_i}, a_{\alpha_j}) \mid a_{\alpha_i} \mathbin{\dot\wedge} a_{\alpha_j} \text{ occurs in } \xi\}.$$

Let $\xi_\wedge$ be the formula obtained by replacing the occurrences $a_{\alpha_i} \mathbin{\dot\wedge} a_{\alpha_j}$ in ψ with $a_{\alpha_i} \wedge a_{\alpha_j}$.

The order condition *(OC) states that the order of events a_{α_i} and a_{α_j} as above is not causal if the following holds:* exists σ' such that $\sigma' \nvDash \varphi$ and $val_{\mathcal{A}}(\sigma) = val_{\mathcal{A}}(\sigma')$ and $\sigma' \nvDash_e \xi$ and $\sigma' \vDash_e \xi_\wedge$.

Note that for the EOL formula ξ_σ in (8) the following holds: $\xi_\sigma = \xi_{\sigma\wedge}$. As a consequence, the order of events in ξ_σ is causal. We say that ξ_σ satisfies OC.

The following result states the equivalence with the original notion of causality in [19], for the case of safety LTL properties.

Corollary 1. *Let $T = (S, Act, \to, I, AP, L)$ be a transition system without terminal states. Let φ be a safety LTL property. A G-complex EOL formula θ is a cause in the sense of Definition 8 if and only if it is a cause in the sense of [19].*

Proof Sketch. First, recall that counterexamples witnessing the violation of safety properties are finite. Hence, in [19], the satisfiability of EOL formulae (characterising such counterexamples) was established based on finite traces. Nevertheless, as can be seen from Definition 4, satisfiability of G-complex formulae can be defined via finite traces as well. In fact, the semantics of G-complex formulae and EOL formulae as in [19] coincide. These being said, the equivalence of the two notions of causality in the context of safety properties follows immediately by case analysis on AC1–AC3. AC1 and, respectively, AC3 in Definition 8 are identical to their counterparts in [19]. From AC2(1) and AC2(2) in Definition 8 we can infer AC2(1) in [19]. AC2(2) in Definition 8 implies AC2(2) in [19], whereas AC2(1) and AC2(2) in [19] imply their counterparts in Definition 8. □

We further introduce a definition of causality classes for general LTL-properties. Intuitively, causality classes can be interpreted as "generalized counterexamples".

Definition 12 (Causality Class)**.** *Let $T = (S, Act, \to, I, AP, L)$ be a transition system without terminal states and let φ be a general LTL formula. Every infinite EOL formula $\xi = \theta \wedge^{\omega} \psi$ that is considered a cause for the violation of φ, i.e., every infinite EOL formula ξ that satisfies AC1-AC3 and OC, defines a causality class CC_ξ. CC_ξ is defined as the set of all valid execution traces in T that satisfy ξ.*

For example, the EOL formula $\xi_\sigma = (B2 \wedge_{[} \neg E2) \wedge^{\omega} (\neg E2)$ in (8) satisfies AC1–AC3 and OC and, therefore, defines a causality class. Moreover, note that one execution can belong to more than one causality class.

5 Completeness and Soundness

We say that causality checking is *complete* whenever for each possible execution trace that violates an LTL property in the transition system under analysis, there exists a causality class representing this trace. Therefore, completeness can be seen as a necessity condition. The completeness of causality checking depends on a complete enumeration of all bad and good traces in the system.

Theorem 1 (Completeness)**.** *Let $T = (S, Act, \to, I, AP, L)$ be a transition system without terminal states. Let σ range over infinite execution traces in T violating an LTL-definable property φ, i.e., $\sigma \not\models \varphi$. For each such σ there exists a causality class of φ containing this trace.*

Proof. Let φ be a general LTL property and let $\xi = \xi_1 \vee \ldots \vee \xi_n$ be the disjunction of all EOL formulae ξ_i that satisfy AC1–AC3 and OC. Let σ be a trace such that $\sigma \not\models \varphi$. We have to show that $\sigma \in CC_{\xi_i}$ for some $i \in 1, \ldots, n$. Assume that σ is not contained in any causality class, that is $\sigma \not\models_e \xi_i$ for all EOL formulae ξ_i that satisfy the conditions AC1–AC3 and OC.

Let ξ_σ be the EOL formula representing σ, and let $\mathcal{Z}$ and $\mathcal{W}$ be the corresponding event variable partitioning, with respect to *Act*. Since $\sigma \models_e \xi_\sigma$ it follows that ξ_σ is excluded from ξ by one of the AC1–AC3 tests. We will show that this is not the case for any of the conditions AC1–AC3 or OC.

- AC1 is satisfied because for σ it holds that $\sigma \vDash_e \xi_\sigma$ and $\sigma \nvDash \varphi$.
- AC2(1) holds given the assumption that there exist n EOL formulae in ξ that satisfy AC1–AC3 and OC.
- If AC2(2) fails, then there exists an infinite execution σ'' with $\sigma'' \vDash_e \xi_\sigma$ (and, hence, $val_{\mathcal{Z}}(\sigma) = val_{\mathcal{Z}}(\sigma'')$) but $\sigma'' \vDash \varphi$. Let $\xi_{\sigma''}$ be the EOL formula derived from σ''. We can now transform ξ_σ to a new formula ξ'_σ by prohibiting the occurrence of a_α in ξ_σ, in the same locations as they occur in $\xi_{\sigma''}$. Consequently, we still have $\sigma \vDash_e \xi'_\sigma$ and $\sigma \vDash \xi'_\sigma$ but now, $\sigma'' \nvDash_e \xi'_\sigma$. Thus, σ'' does not influence the satisfaction of AC2(2) by ξ'_σ. If ξ'_σ still doesn't satisfy AC2(2), *i.e.*, if there is another σ''' with $\sigma''' \vDash_e \xi$ and $val_{\mathcal{Z}}(\sigma) = val_{\mathcal{Z}}(\sigma''')$ but $\sigma''' \vDash \varphi$, we repeat the procedure from above. Since the action alphabet is finite and the EOL formulae are finitely representable, this procedure will stop after finitely many steps. The resulted EOL formula ξ'_σ satisfies AC2(2).
- If AC3 excludes ξ_σ, then there must be some $\xi'_\sigma \subset \xi_\sigma$ that satisfies AC1 and AC2. We still have $\sigma \vDash \xi'_\sigma$ by Definition 8 .

In all cases we obtain an EOL formula ξ'_σ that satisfies AC1-AC3 and OC, such that $\sigma \vDash \xi'_\sigma$. Thus, σ is contained in the causality class $CC_{\xi'_\sigma}$. □

We define a causality checking result to be *sound* if whenever the events described by a causality class occur, the property violation occurs.

Theorem 2 (Soundness)**.** *Let $T = (S, Act, \to, I, AP, L)$ be a transition system without terminal states. Each execution trace σ of T contained in a causality class of a general LTL property φ is a bad trace, i.e. $\sigma \nvDash \varphi$.*

Proof. Let σ be contained in the causality class CC_ξ for some EOL formula ξ. Since ξ defines a causality class, it must, by definition, satisfy AC1-AC3 and OC. In particular, ξ must satisfy AC2(2). Since $\sigma \in CC_\xi$ we have $\sigma \vDash_e \xi$ by definition of causality classes. It follows from AC2(2) that $\sigma \nvDash \varphi$. □

6 Conclusions

We have presented an approach for extending causality checking towards general LTL-definable properties. To this end, we have reconsidered the actual cause conditions AC1-AC3 and OC, and adapted them to the lasso-shaped counterexamples that general LTL properties entail.

For practical reasons related to the implementation of the causality checking procedure, our current results are limited to LTL-definable properties. Nevertheless, in the future, we consider extending the formal framework of causality checking to the more general case of $\omega-$ regular linear-time properties [1]. It should be pointed out that the described adaption can be straightforwardly extend to general ω-regular properties, corresponding to the expressiveness of Büchi automata, which are a strictly larger class of properties than LTL [25].

As already mentioned, we consider implementing the current causality checking approach in an automated tool. Of particular interest is QuantUM [18], a tool that enables the semi-formal specification of systems in terms of SysML [21] and applies LTL model-checking for determining what caused the violation of a safety LTL property. Recall that our notion of causality relies on the complete enumeration of system traces. Hence, the main challenge is to determine all (lasso-shaped) counterexamples in an efficient way (*e.g.*, on-the-fly [8, 24, 3, 4]). Further future research comprises significant case studies in order to asses the scalability of our approach.

Acknowledgements. The authors are grateful for the useful comments received from the anonymous reviewers of CREST 2018.

References

[1] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[2] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard J. Trefler (2009): *Explaining Counterexamples Using Causality*. In Ahmed Bouajjani & Oded Maler, editors: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, *Lecture Notes in Computer Science* 5643, Springer, pp. 94–108. Available at `https://doi.org/10.1007/978-3-642-02658-4_11`.

[3] Vincent Bloemen, Alfons Laarman & Jaco van de Pol (2016): *Multi-core on-the-fly SCC decomposition*. In Rafael Asenjo & Tim Harris, editors: *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, Barcelona, Spain, March 12-16, 2016*, ACM, pp. 8:1–8:12, doi:10.1145/2851141.2851161. Available at `http://doi.acm.org/10.1145/2851141.2851161`.

[4] Vincent Bloemen & Jaco van de Pol (2016): *Multi-core SCC-Based LTL Model Checking*. In Roderick Bloem & Eli Arbel, editors: *Hardware and Software: Verification and Testing - 12th International Haifa Verification Conference, HVC 2016, Haifa, Israel, November 14-17, 2016, Proceedings*, *Lecture Notes in Computer Science* 10028, pp. 18–33, doi:10.1007/978-3-319-49052-6_2. Available at `https://doi.org/10.1007/978-3-319-49052-6_2`.

[5] Georgiana Caltais, Stefan Leue & Mohammad Reza Mousavi (2016): *(De-)Composing Causality in Labeled Transition Systems*. In Gregor Gößler & Oleg Sokolsky, editors: *Proceedings First Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies, CREST@ETAPS 2016, Eindhoven, The Netherlands, 8th April 2016.*, *EPTCS* 224, pp. 10–24. Available at `https://doi.org/10.4204/EPTCS.224.3`.

[6] Edmund M. Clarke & Helmut Veith (2003): *Counterexamples Revisited: Principles, Algorithms, Applications*. In Nachum Dershowitz, editor: *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, *Lecture Notes in Computer Science* 2772, Springer, pp. 208–224. Available at `https://doi.org/10.1007/978-3-540-39910-0_9`.

[7] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper & Mihalis Yannakakis (1992): *Memory-Efficient Algorithms for the Verification of Temporal Properties*. *Formal Methods in System Design* 1(2/3), pp. 275–288, doi:10.1007/BF00121128. Available at `https://doi.org/10.1007/BF00121128`.

[8] Jean-Michel Couvreur, Alexandre Duret-Lutz & Denis Poitrenaud (2005): *On-the-Fly Emptiness Checks for Generalized Büchi Automata*. In Patrice Godefroid, editor: *Model Checking Software, 12th International SPIN Workshop, San Francisco, CA, USA, August 22-24, 2005, Proceedings*, *Lecture Notes in Computer Science* 3639, Springer, pp. 169–184, doi:10.1007/11537328_15. Available at `https://doi.org/10.1007/11537328_15`.

[9] Gregor Goessler & Lacramioara Astefanoaei (2014): *Blaming in component-based real-time systems*. In Tulika Mitra & Jan Reineke, editors: *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, ACM, pp. 7:1–7:10, doi:10.1145/2656045.2656048. Available at `http://doi.acm.org/10.1145/2656045.2656048`.

[10] Gregor Gößler, Daniel Le Métayer & Jean-Baptiste Raclet (2010): *Causality Analysis in Contract Violation*. In: *Runtime Verification - First International Conference, RV 2010*, *Lecture Notes in Computer Science* 6418, Springer, pp. 270–284, doi:10.1007/978-3-642-16612-9_21. Available at `http://dx.doi.org/10.1007/978-3-642-16612-9{_}21`.

[11] Gregor Gößler & Daniel Le Métayer (2015): *A general framework for blaming in component-based systems*. *Sci. Comput. Program.* 113, pp. 223–235, doi:10.1016/j.scico.2015.06.010. Available at `https://doi.org/10.1016/j.scico.2015.06.010`.

[12] Gregor Gößler & Jean-Bernard Stefani (2016): *Fault Ascription in Concurrent Systems*. In: *Trustworthy Global Computing - 10th International Symposium, TGC*, *Lecture Notes in Computer Science* 9533, Springer, pp. 79–94, doi:10.1007/978-3-319-28766-9. Available at `http://dx.doi.org/10.1007/978-3-319-28766-9`.

[13] Nicolas Halbwachs & Lenore D. Zuck, editors (2005): *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*. *Lecture Notes in Computer Science* 3440, Springer, doi:10.1007/b107194. Available at `https://doi.org/10.1007/b107194`.

[14] Joseph Y. Halpern (2015): *A Modification of the Halpern-Pearl Definition of Causality*. In Qiang Yang & Michael Wooldridge, editors: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, AAAI Press, pp. 3022–3033. Available at `http://ijcai.org/Abstract/15/427`.

[15] Joseph Y. Halpern & Judea Pearl (2002): *Causes and Explanations: A Structural-Model Approach. Part II: Explanations*. *CoRR* cs.AI/0208034. Available at `http://arxiv.org/abs/cs.AI/0208034`.

[16] Michael Huth & Mark Dermot Ryan (2004): *Logic in computer science - modelling and reasoning about systems (2. ed.)*. Cambridge University Press.

[17] Tsutomu Kumazawa & Tetsuo Tamai (2011): *Counterexample-Based Error Localization of Behavior Models*. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann & Rajeev Joshi, editors: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, *Lecture Notes in Computer Science* 6617, Springer, pp. 222–236, doi:10.1007/978-3-642-20398-5_17. Available at `https://doi.org/10.1007/978-3-642-20398-5_17`.

[18] Florian Leitner-Fischer & Stefan Leue (2011): *QuantUM: Quantitative Safety Analysis of UML Models*. In Mieke Massink & Gethin Norman, editors: *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages, QAPL 2011, Saarbrücken, Germany, April 1-3, 2011.*, *EPTCS* 57, pp. 16–30, doi:10.4204/EPTCS.57.2. Available at `https://doi.org/10.4204/EPTCS.57.2`.

[19] Florian Leitner-Fischer & Stefan Leue (2013): *Causality Checking for Complex System Models*. In Roberto Giacobazzi, Josh Berdine & Isabella Mastroeni, editors: *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, *Lecture Notes in Computer Science* 7737, Springer, pp. 248–267, doi:10.1007/978-3-642-35873-9_16. Available at `https://doi.org/10.1007/978-3-642-35873-9_16`.

[20] Florian Leitner-Fischer & Stefan Leue (2013): *Probabilistic fault tree synthesis using causality computation*. *IJCCBS* 4(2), pp. 119–143, doi:10.1504/IJCCBS.2013.056492. Available at `https://doi.org/10.1504/IJCCBS.2013.056492`.

[21] OMG (2007): *OMG Systems Modeling Language (OMG SysML), V1.0*. Technical Report, Object Management Group. Available at `http://www.omg.org/spec/SysML/1.0/PDF`.

[22] Doron A. Peled (2001): *Software Reliability Methods*. Texts in Computer Science, Springer, doi:10.1007/978-1-4757-3540-6. Available at `http://u.cs.biu.ac.il/~doronp/srm.html`.

[23] Viktor Schuppan & Armin Biere (2005): *Shortest Counterexamples for Symbolic Model Checking of LTL with Past*. In Halbwachs & Zuck [13], pp. 493–509, doi:10.1007/978-3-540-31980-1_32. Available at `https://doi.org/10.1007/978-3-540-31980-1_32`.

[24] Stefan Schwoon & Javier Esparza (2005): *A Note on On-the-Fly Verification Algorithms*. In Halbwachs & Zuck [13], pp. 174–190, doi:10.1007/978-3-540-31980-1_12. Available at `https://doi.org/10.1007/978-3-540-31980-1_12`.

[25] Pierre Wolper (1983): *Temporal Logic Can Be More Expressive*. *Information and Control* 56(1/2), pp. 72–99, doi:10.1016/S0019-9958(83)80051-5. Available at `https://doi.org/10.1016/S0019-9958(83)80051-5`.