Towards Causal Explanations of Property Violations in Discrete Event Systems

Gregor Gössler, Thomas Mari, Yannick Pencolé, Louise Travé-Massuyès

Towards Causal Explanations of Property Violations in Discrete Event Systems*

Gregor Gössler[1], **Thomas Mari**[1], **Yannick Pencolé**[2] and **Louise Travé-Massuyès**[2]

[1] Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
e-mail: {gregor.goessler, thomas.mari}@inria.fr
[2] LAAS-CNRS, CNRS, Univ. Toulouse, F-31400, Toulouse, France
e-mail: {ypencole, louise}@laas.fr

Abstract

Model-Based Diagnosis of discrete event systems (DES) usually aims at detecting failures and isolating faulty event occurrences based on a behavioural model of the system and an observable execution log. The strength of a diagnostic process is to determine *what* happened that is consistent with the observations. In order to go a step further and explain *why* the observed outcome occurred, we borrow techniques from causal analysis. As opposed to the classical fault diagnosis problem, we consider that a system is failing as soon as a specific behavioural property is violated by the current run of the system. We then formally define different notions of explanation for DES in order to extract the relevant part of a property violation that can be understood by a human operator.

1 Introduction

As embedded software implements increasingly complex functionalities and takes over decision making even in critical situations, the ability to automatically construct explanations for the system behaviour is crucial during the whole software life cycle: at design time to quickly localise errors and gain confidence in the system; at runtime to understand the sources of failures and help assigning legal liability in the case of an accident. To this end explanations should be both *complete*, that is, retain all information about the system execution that is causally relevant to the failure at hand, for a given notion of causality, and *concise*, that is, make abstraction of irrelevant information.

The construction of formal explanations has been the subject of research work in different communities such as AI [18], molecular biology (explanation of how a product is brought about by a series of reactions) [7], and computer science (explanation of counter-examples from model checking) [16] that developed, often independently, a set of techniques aiming at extracting relevant information from a system trace. In this paper we discuss and put in perspective several of these notions of explanation, discuss their shortcomings on the simple model of an adaptive cruise control system, and propose a new definition of explanation that overcomes several issues.

*This work has been partially supported by the French ANR project DCore (ANR-18-CE25-0007).

Our approach is situated at the crossroads between *model-based diagnosis* [13; 8] and *causal analysis*. The strength of a diagnostic process is usually to determine *what* happened that is consistent with the observations (consistency-based diagnosis) or to explain symptomatic observations with fault modes (abductive diagnosis based on logic entailment). Our paper addresses the construction of *formal explanations* for DES. Classically, model-based diagnosis of DES aims at detecting failures and isolating faulty event occurrences based on a normal/abnormal behavioural models of the system used to check the consistency of an observable execution log [20; 2]. In this paper, we consider that a system fails as soon as a specific behavioural property is violated by the current run of the system and, in order to go a step further, our objective is to explain *why* the violation has occurred based on the observed outcome. We then formally define different notions of explanation for DES in order to extract the relevant part of a property violation that can be understood by a human operator.

The paper is organised as follows. Section 2 presents the formal framework that is used throughout this paper and describes the problem statement. Section 3 proposes a first attempt to define an explanation as a subsequence of events and describes why such definition is not satisfactory. Section 4 then presents a new definition, called *choice-based explanation*, and proposes an algorithm to compute choice-based explanations for the violation of a property, these being consistent with the observations of the system. Section 5 presents the related work and a comparative analysis with our proposal. Finally, Section 6 resumes the contributions and outlines some perspectives for future work.

2 Framework

2.1 Formal preliminaries

The framework that we use throughout this paper relies on classical automata. An *automaton* $\mathcal{A} = (S, \Sigma, \delta, F, s_0)$ is a 5-tuple where S is a set of states, Σ is a set of events, $\delta \subseteq S \times \Sigma \times S$ is a set of transitions, $F \subseteq S$ is the set of accepting states, and $s_0 \in S$ is the initial state. $\mathcal{A}$ is *deterministic* if whenever $(s, e, s_1) \in \delta$ and $(s, e, s_2) \in \delta$, $s_1 = s_2$. For a state s, we define its preset ${}^\bullet s := \{s' \in S \mid \exists e \in \Sigma : (s', e, s) \in \delta\}$ and postset $s^\bullet := \{s' \in S \mid \exists e \in \Sigma : (s, e, s') \in \delta\}$ as the states preceding and following s in terms of transitions. A trace $t \in \Sigma^*$ is a finite sequence of events. The concatenation of a trace t with some event a is written $t \cdot a$. The length of a trace t is written $|t|$ and it is the number of occurrences of events. The set of accepting

traces of the automaton $\mathcal{A}$ is written $T(\mathcal{A})$. Given a trace t, we can build an automaton $ex(t)$ such that $T(ex(t)) = \{t\}$. An example is given in Figure 1 for the trace $a{\cdot}a{\cdot}br{\cdot}a{\cdot}cr{\cdot}br$.

We also recall the definition of trace projection below.

Definition 1 (Projection) *Let Σ, Σ' be two sets of events such that $\Sigma' \subseteq \Sigma$. Let $\pi_{\Sigma'} : \Sigma^* \to \Sigma'^*$ be the projection defined by induction on traces:*

$$\forall e \in \Sigma, \pi_{\Sigma'}(e) = \begin{cases} e & \text{if } e \in \Sigma' \\ \epsilon & \text{otherwise} \end{cases}$$

and $\forall t \in \Sigma^*, \forall e \in \Sigma : \pi_{\Sigma'}(t \cdot e) = \pi_{\Sigma'}(t) \cdot \pi_{\Sigma'}(e)$ where ϵ is the neutral element for the concatenation.

In the following, we also distinguish the notion of trace (i.e. event sequence) from the notion of *automaton execution*.

Definition 2 (Execution) *Given an automaton $\mathcal{A} = (S, \Sigma, \delta, F, s_0)$, an execution ρ is defined as a finite alternating sequence of states in S and events in Σ ending by a state $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \ldots \xrightarrow{e_{k-1}} s_k$ such that for all $i \in [0, k-1], (s_i, e_i, s_{i+1}) \in \delta$. An execution is accepting if the last state is accepting. The length of an execution ρ, written $|\rho|$, is the number of events in ρ. The trace of an execution $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \ldots \xrightarrow{e_{k-1}} s_k$ is the sequence of events $e_0 \cdot e_1 \cdot \ldots \cdot e_{k-1}$.*

We end this preliminary section with the synchronization operation on automata based on shared events. When two automata share an event, this event must occur simultaneously in both. The other events occur asynchronously, which means that they only occur in their respective automaton.

Definition 3 (Synchronization) *The synchronization of two automata $\mathcal{A}_i = (S_i, \Sigma_i, \delta_i, F_i, s_{0i})$, $i = 1, 2$, is the automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \{S_1 \times S_2, \Sigma_1 \cup \Sigma_2, \delta, F_1 \times F_2, (s_{01}, s_{02})\}$ where $((s_1, s_2), e, (s'_1, s'_2)) \in \delta \iff$*

$$\begin{cases} & (e \in \Sigma_1 \cap \Sigma_2 \wedge (s_1, e, s'_1) \in \delta_1 \wedge (s_2, e, s'_2) \in \delta_2) \\ \vee & (e \in \Sigma_1 \setminus \Sigma_2 \wedge (s_1, e, s'_1) \in \delta_1 \wedge s_2 = s'_2) \\ \vee & (e \in \Sigma_2 \setminus \Sigma_1 \wedge (s_2, e, s'_2) \in \delta_2 \wedge s_1 = s'_1) \end{cases}$$

2.2 Problem statement

We address the problem of defining and computing different notions of explanation for the violation of a given behavioural property by a DES under partial observation. We consider that a behavioural model of the system is available and is represented as a deterministic automaton $\mathcal{A} = (S, \Sigma, \delta, F, s_0)$. This automaton can be monolithic or can result from a multi-component model, in which case the system is modeled by a set of interacting automata. This interaction is defined by the synchronization of the common events of each automaton $\mathcal{A} = \mathcal{A}_1 \| \ldots \| \mathcal{A}_n$ [20]. The system has a prefix-closed behavioural description. That means that every prefix of behaviour respecting the behavioural description also respects the behavioural description so $S = F$. The set of events is partitioned into the set of observable events Σ_o and the set of unobservable events Σ_{uo}. We make the assumption that there are no unobservable cycles possible in $\mathcal{A}$, in other words we assume that the number of unobservable events that can be generated by the system between any two observable events is bounded. Any execution of the system can be associated with a *log* $L \in \Sigma_o^*$ that is a finite sequence of observable events. As the system is partially observable, a log L can be associated with several possible traces (these are the traces that are *consistent* with the log L). Formally, the set of traces consistent with the log L is given by:

$$tr(L) = \{t \in \Sigma^* \mid \pi_{\Sigma_o}(t) = L\} \cap T(\mathcal{A})$$

The log L can be represented as an automaton $\mathcal{L} := ex(L)$ such that $T(\mathcal{L}) = \{L\}$. An example is given in Figure 1 for the log $a \cdot a \cdot br \cdot a \cdot cr \cdot br$.

We aim at investigating the reasons why a given safety property in the system is violated. A safety property is respected by the system as long as the current execution leads to a state where the property is satisfied. If the execution of the system leads to a state that does not satisfy the property, then we say that the property is violated. The violation of a safety property is permanent, i.e. all continuations of an execution violating the safety property also violate the property. Such a property can be defined as a prefix-closed event language over Σ such that, if the execution of the system generates a trace of this language, it leads to a state that satisfies the property. In the following, we denote by $\mathcal{P}$ an observer of the *violation of the property*. The observer $\mathcal{P}$ is a complete automaton associated with the safety property that accepts as language only the traces of the executions violating the safety property.

We can now formally represent as an automaton:

1. the set of executions of the system that violate the safety property: $\mathcal{A} \| \mathcal{P}$ (see Figure 2);
2. the set of executions of the system that are consistent with a given log L: $\mathcal{A} \| \mathcal{L}$.
3. the set of executions of the system that violate the safety property *and* that are consistent with a given log L: $\mathcal{A} \| \mathcal{P} \| \mathcal{L}$ (see Figures 5, 7).

Note that the synchronization being an associative operation, $\mathcal{A} \| \mathcal{P} \| \mathcal{L}$ is perfectly defined as $(\mathcal{A} \| \mathcal{P}) \| \mathcal{L} = \mathcal{A} \| (\mathcal{P} \| \mathcal{L})$.

A state of $\mathcal{A} \| \mathcal{P} \| \mathcal{L}$ is a tuple (s_a, s_p, s_l) where s_a, s_p and s_l are the respective states in $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{L}$. We use the notation $(s_a, s_p, s_l)_{\mathcal{A} \| \mathcal{P}} = (s_a, s_p)$ to pick the state in $\mathcal{A} \| \mathcal{P}$ associated with a state in $\mathcal{A} \| \mathcal{P} \| \mathcal{L}$. Thanks to this association, in the rest of the paper, we implicitly extend the functions defined on the states of $\mathcal{A} \| \mathcal{P}$ on the states of $\mathcal{A} \| \mathcal{P} \| \mathcal{L}$.

2.3 Case study

An example inspired from a cruise controller for a car will be used all along this paper as an illustration. The cruise controller has 3 discrete variables which are the speed $v \in \{0, \ldots, 3\}$ of the car, the distance $d \geq 0$ with respect to the front car and the state of the sensor $sensor \in \{ok, ko\}$. A state $s \in S$ of the system is a triple $(v, d, sensor)$. The events Σ of the system are the following:

- a represents an acceleration during one time unit, modeled by a transition from $(v, d, sensor)$ to $(v + 1, d + (1 - v), sensor)$;
- br represents braking during one time unit, modeled by a transition from $(v, d, sensor)$ to $(v - 1, d + (1 - v), sensor)$;
- cr represents cruising at constant speed during one time unit, modeled by a transition from $(v, d, sensor)$ to $(v, d + (1 - v), sensor)$;
- and $fail$ represents a failure of the sensor that measures the distance of the car with respect to the front car, modeled by a transition from (v, d, ok) to (v, d, ko). In the failure mode, the sensor reports a distance twice the real one.

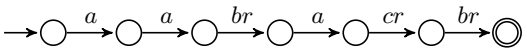

Figure 1: $ex(a \cdot a \cdot br \cdot a \cdot cr \cdot br)$

For the sake of simplicity, we assume that the front car has a constant speed of 1.

The initial state is $s_0 = (0, 2, ok)$, meaning that the car is stopped at distance 2 of the front car with a correctly working sensor. Because the front car is going at constant speed 1 and the car is going at speed v, the distance between the two cars after a transition labelled cr (no acceleration and no braking) is $d + (1 - v)$ except for the $fail$ transition, which does not induce any dynamics on the car. States with $d = 0$ model a collision and there is no outgoing transition. The $fail$ event happens at most once in an execution. The cruise controller relies on the measurement v^m of v that we suppose here never faulty ($\forall s \in S$, $v^m = v$) and the measurement d^m of d. If the distance sensor is healthy then $d^m = d$, but if the sensor is failing, the cruise controller then believes that the distance is actually twice the real distance (i.e. $d^m = 2d$).

Figure 2 shows part of the cruise controller automaton $\mathcal{A}$ for $d \in [0, 4]$. The role of $\mathcal{A}$ is to enable acceleration (event a), breaking (event br) or cruise (event cr) depending on the measured distance between cars. For the purpose of this paper we may consider $\mathcal{A}$ to be given. Intuitively, as long as $sensor = ok$, (1) the distance d is maintained in the interval $[1, 4]$ in order to enable platooning and avoid collision; (2) once a speed $v \geq 1$ is reached, v is maintained in the interval $[1, 3]$; and (3) in each state at least one of the events a, br, cr is enabled. Once a *fault* event has occurred, the behavior of the controller in each *observed* state is similar to its nominal behavior. Remember, however, that the faulty distance sensor reports twice the actual distance and that the decision of allowing cruse or acceleration is taken from the erroneous distance value.

The observer $\mathcal{P}$ is an automaton that accepts all traces in Σ^*, and enters a distinct sink state if and only if a violation of the safety property $d > 0$ is observed, that is, whenever a trace leads the system to a state $(0, v, sensor)$. An accepting trace of $\mathcal{A}\|\mathcal{P}$ is in $T(\mathcal{A})$ and violates the safety property.

All along the paper, we will consider that the system has produced the log $L = a \cdot a \cdot br \cdot a \cdot cr \cdot br$ accepted by the automaton $\mathcal{L} = ex(L)$ shown in Figure 1.

The question to be answered is: what kind of *explanations* can we provide about the violation of the safety property $d > 0$ based on the execution log L and the model $\mathcal{A}$?

3 Sub-sequence-based explanations

In this section, we base the explanation of a property violation on sub-sequences of traces. More specifically, we aim at retaining in the explanation only the events relevant to the violation.

A sub-sequence of a sequence of events $w \in \Sigma^*$ is a sequence of events $s_w \in \Sigma^*$ such that there exists a monotone function $\psi : [0, |s_w| - 1] \to [0, |w| - 1]$ such that $s_w = [w(\psi(i))]_{i \in [0, |s_w|-1]}$. We write $u \subseteq v$ when u is a sub-sequence of v.

Definition 4 (Sub-sequence based explanations) *Given a behavioural model $\mathcal{A}$, an observer $\mathcal{P}$ and a log L, sub-sequence based explanations are the minimal (with respect to $\subseteq$) traces w such that:*

- *$\exists w' : w'$ is accepted by $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$ and $w \subseteq w'$, and*
- *w is accepted by $\mathcal{A}\|\mathcal{P}$.*

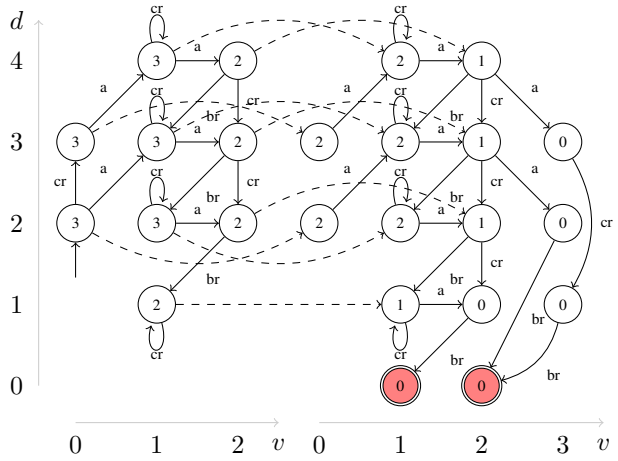

Figure 2: The cruise controller automaton $\mathcal{A}$. The axes indicate the speed v (in abscissa) and the actual distance d (in ordinate). The left part of the automaton represents the behaviours when the sensor is correct, i.e. state ok, and the right part of the automaton represents the behaviours when the sensor is in failure mode, i.e. state ko, and it transmits twice the actual distance. The observable events have thick lines, the unobservable events have dashed lines. The product automaton $\mathcal{A}\|\mathcal{P}$ has the same structure, with the red states indicating a violation of the safety property $d > 0$, i.e., they are the accepting states of $\mathcal{A}\|\mathcal{P}$. The numbers in the states are levels of choice and are detailed in Section 4.

Sub-sequence based explanations are the minimal accepting traces w of the behavioural model that violate the safety property, and that are sub-sequences of some word w' of A that violates the property and is consistent with the log. Hence, the observable events of w are a sub-sequence of the log.

We briefly sketch how to construct such sub-sequence-based explanations. Adding for each transition (s, a, s') of $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$ a silent transition (s, τ, s') and determinizing the result yields an automaton that accepts exactly the sub-words w of some word w' accepted by $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$. By further composing the automaton with $\mathcal{A}\|\mathcal{P}$ we obtain an automaton that accepts exactly the words w satisfying the two conditions of Definition 4. The minimal accepted words are then the sub-sequence based explanations.

Example 1 *An automaton accepting exactly the words w that satisfy the two conditions of Definition 4 is shown in Figure 3. Its minimal accepted words coincides in this example with the set of all accepted words, which include $(a \cdot a \cdot fail + a \cdot fail \cdot a + fail \cdot a \cdot a) \cdot a \cdot br$: after two acceleration events, the failure of the sensor allows for a third acceleration to be performed, and the execution ends with a braking event. Two events of the log, the first braking (br) and the first cruise (cr) have been removed by this explanations. The first braking in the log compensates one of the past accelerations. This means that in the log, one of the accelerations is not relevant concerning the violation of*

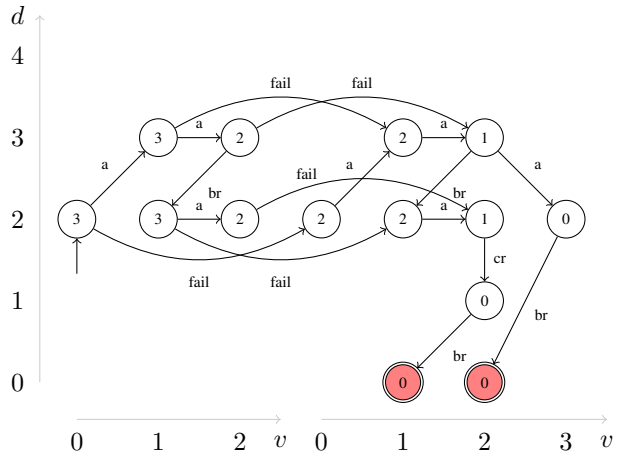

Figure 3: The minimal words accepted by the automaton are the subsequence-based explanations for the running example.

the safety property. The last braking is not relevant either. At the point of the execution where the last braking is done, the violation of the safety property is unavoidable within a finite number of steps.

A shortcoming of sub-sequence based explanations is the relevance of the events in the explanation. If we have a system that systematically performs an initialization then all the executions will share a common prefix and therefore also the sub-sequence based explanation. The initialization might not be related to the violation of the property and the explanation may lack conciseness. In other words, sub-sequence based explanations may contain events which are not relevant to the violation of the property. Consequently, sub-sequence based explanations may not be the best way to explain failures, particularly for large systems.

4 Choice-based explanations

In this section we present choice-based explanations that, as indicated by the name, rely on a notion of *choice*. We first define the notion of *level of choice*, which is a special case of the *fate and free will* layers of [16] in the case where the set of environment variables is empty. Our notion of choice state corresponds to the boundaries of layers $i > 0$.

Definition 5 (Level of choice) *Let $\xi : S \to \mathbb{N} \cup \{\infty\}$.*

We say that ξ is a level of choice in the automaton $\mathcal{A}\|\mathcal{P} = (S, \Sigma, \delta, S_0, F)$ if :

1. *$\forall s \in F$, $\xi(s) = 0$*
2. *$\forall s \in S$ such that s is not co-reachable, $\xi(s) = \infty$*
3. *$\forall s \in S \setminus F$, if $s^\bullet \neq \emptyset$ and $\exists i \in \mathbb{N} \cup \{\infty\}, \{\xi(s') \mid s' \in s^\bullet\} = \{i\}$ then $\xi(s) = i$. Those states s are called non-choice states.*
4. *$\forall s \in S \setminus F$, if $s^\bullet \neq \emptyset$ and $|\{\xi(s') \mid s' \in s^\bullet\}| > 1$ then $\xi(s) = 1 + \min_{s' \in s^\bullet}(\xi(s'))$. Those states s are called choice states.*
5. *Maximality: ξ is maximal among the functions fulfilling the preceding conditions.*

The executions of the behavioural model $\mathcal{A}$ violating the safety property end in an accepting state, i.e. in a state of F. The first condition of Definition 5 requires that the level of choice of accepting states is 0, it means that no choice has to be done in those states to violate the safety property because the violation already happened.

The second condition requires the level of choice of the states that are not co-reachable to be ∞; this means that in those states, no choice can be done to reach a violation.

The third condition requires that, given a state s, if the level of choice of the successors is uniform, then the level of choice of s is the same as its successors. Those states are qualified as non-choice states because the decision about the out-going transition does not impact the level of choice.

The fourth condition requires that, given a state s, if the level of choice of the successors is not uniform, then the level of choice is equal to the increment by one of the lowest level of choice in the postset $s^\bullet$. Choice states are the counterpart of non-choice states. When the decision about the outgoing transition impacts the level of choice, this decision can be made so as to decrease the level of choice. When, in a state, it is possible to decrease the level of choice by choosing a transition, it is also possible to increase the level of choice by 0 or more.

The last condition is required because we do not want infinite executions with a finite number of choice states.

Example 2 *Figure 2 represents $\mathcal{A}\|\mathcal{P}$ for the cruise controller presented in the case study. The accepting states, in red, are the ones violating the property because in these states $d = 0$. The level of choice is indicated for each state. We see that level 0 occurs for more states than the accepting states. In those states the violation is inevitable. For example, in state $(3, 3, ko)$, the sensor is faulty. The sensor sends the double of the real distance to the cruise controller. The controller believes that the car is at a distance 6 of the front car and that applying cr is safe. However, the real trace is $(3, 3, ko) \xrightarrow{cr} (3, 1, ko)$ and once in $(3, 1, ko)$, even breaking cannot avoid to violate the property since it leads to the state $(2, 0, ko)$. Notice that the level of choice of the initial state is 3, it means that 3 "bad" choices are needed to violate the safety property from the initial state.*

We propose the following fixpoint algorithm to compute the levels of choice.

Definition 6 (Computing the level of choice) *Given $\mathcal{A}\|\mathcal{P} = \{S, \Sigma, \delta, S_0, F\}$, we define the sets $(F_{i,j})_{\mathbb{N}\times\mathbb{N}\cup\{\infty\}}$ as follows. For any $i, j \in \mathbb{N} \cup \{\infty\}$ let*

$$\begin{cases} F_{0,0} &= F \\ F_{i,j+1} &= F_{i,j} \cup \{s \in {}^\bullet F_{i,j} \mid s^\bullet \subseteq F_{i,j}\} \\ F_{i,\infty} &= \cup_{j=1}^{\infty} F_{i,j} \\ F_{i+1,0} &= {}^\bullet F_{i,\infty} \setminus \bigcup_{0 \leq k \leq i} F_{k,\infty} \end{cases}$$

Let $\xi_a : S \to \mathbb{N} \cup \{\infty\}$ such that $\xi_a(s) = i \iff s \in F_{i,\infty}$, and if $\forall i, s \notin F_{i,\infty}$ then $\xi_a(s) = \infty$.

The sets $(F_{i,j})_{\mathbb{N}\times\mathbb{N}\cup\{\infty\}}$ can be computed in the lexicographic order on (i, j) with a finite number of steps. The computation of ξ_a is given by the sets $(F_{i,j})$. If $s \in F_{i,j}$ then s is in the level of choice i and s is a choice state if and only if j is 0. To compute ξ_a, we first compute $F_{0,0}$ then we compute $F_{0,j}$ using the $F_{i,j+1}$ equation until the fixed point $F_{0,j+1} = F_{0,j}$ is reached and $F_{0,\infty} = F_{0,j}$ with the third equation. In level 0, all the states are non-choice states. For the other levels, we compute by induction on i. For each level, the first step is to compute the choice states $F_{i,0}$ which are the predecessor of the states of level $i - 1$

that are not already at some level lower than i. The second step is to compute the non-choice states by fixed-point iteration like for level 0. The algorithm stops when there are no more choice states, i.e. $F_{i_{max}+1,0} = \emptyset$. When the algorithm stops, all the co-reachable states are in some level $F_{i,\infty}$.

Theorem 1 (Correctness of the algorithm) *The function ξ_a is a level of choice as defined in Definition 5.*

When an event occurs in a non-choice state it is not relevant to be part of the explanation. A choice transition is a transition $s \xrightarrow{e} s'$ of $\mathcal{A}\|\mathcal{P}$ such that s is a choice state. Considering the value $\xi(s') - \xi(s)$, a choice transition can be interpreted as follows. If $\xi(s') = \xi(s) - 1$ the transition can be seen as an *erroneous action* that drives the system closer to the violation. If $\xi(s') = \xi(s)$ then the transition can be seen as a delay, it means that the violation is only delayed until reaching the next choice state at the same level. If $\xi(s') > \xi(s)$ then the transition repairs past erroneous actions by increasing the level of choice. Those transitions are relevant to understand the causes of the violation. The extreme case is when $\xi(s') = \infty$, it means that the rest of the execution is safe. In an execution, the impact of an event depends on the future events. If there is a loop in the execution, then the erroneous actions present in the loop are compensated by transitions that increase the levels of choice and they might not be related to the violation at the end of the execution. The following notion characterises the transitions of an execution that are not compensated.

Definition 7 (Effective choice transitions) *Given $\mathcal{A}_{|||} = \mathcal{A}\|\mathcal{P}\|\mathcal{L}$, a transition $s \xrightarrow{e} s'$ of an accepting execution ρ of $\mathcal{A}_{|||}$ is a effective choice transition if*

$$\forall s'' \in \rho, s < s'' \implies \xi(s'') < \xi(s)$$

where $s < s''$ means that the state s occurs before the state s'' in the execution ρ.

Let $\delta_c(\mathcal{A}_{|||})$ be the set of effective choice transitions for some accepting execution of $\mathcal{A}_{|||}$.

The effective choice transitions sum up the violation in the case a of unique execution. They point at the erroneous actions that are not compensated by some subsequent actions. An example is given in Figure 4 where the red transitions are the effective choice transitions. Given a graph ξ versus the steps of the execution, it is visually easy to identify the effective choice transitions by computing the maximum of ξ on the suffix of the execution.

We also want to provide the human operator with a concise representation of alternative safe executions. This is done by defining *safe alternatives*.

Definition 8 (Safe alternatives) *Given the automata $\mathcal{A}\|\mathcal{P}$ and $\mathcal{A}_{|||} = \mathcal{A}\|\mathcal{P}\|\mathcal{L}$, an effective choice transition $s \xrightarrow{e'} s'$, and an event $e \in \Sigma$, the couple (s, e) is a safe alternative if and only if there exists a state q' of the automaton $\mathcal{A}\|\mathcal{P}$ such that:*

- $s_{\mathcal{A}\|\mathcal{P}} \xrightarrow{e} q' \in \mathcal{A}\|\mathcal{P}$
- $\forall s'' \in s^\bullet, \xi(q') > \xi(s'')$

where $s_{\mathcal{A}\|\mathcal{P}}$ is the associated state of s in $\mathcal{A}\|\mathcal{P}$.

Let $\delta_{alt}(\mathcal{A}_{|||})$ be the set of all safe alternatives.

Safe alternatives are not contained in executions whose trace is in $tr(L)$. Therefore, there is no transition $s \xrightarrow{e} s'''$ in $\mathcal{A}_{|||}$. This condition is ensured by the condition $\forall s'' \in s^\bullet, \xi(q') > \xi(s'')$. If there is a transition $s \xrightarrow{e} s'''$ then $s'''_{\mathcal{A}\|\mathcal{P}} = q'$ and $\forall s'' \in s^\bullet, \xi(q') > \xi(s'')$ does not hold because $\xi(q') = \xi(s''')$. In an execution violating the safety property, taking a safe alternative instead of an effective choice transition creates an alternative execution, prefix of a safe execution.

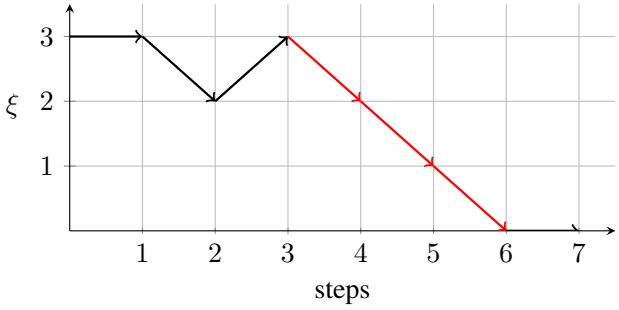

Figure 4: Level of choice versus steps of execution for the trace $(0, 2, ok) \xrightarrow{a} (1, 3, ok) \xrightarrow{a} (2, 3, ok) \xrightarrow{br} (1, 2, ok) \xrightarrow{a} (2, 2, ok) \xrightarrow{fail} (2, 2, ko) \xrightarrow{cr} (2, 1, ko) \xrightarrow{br} (1, 0, ko)$

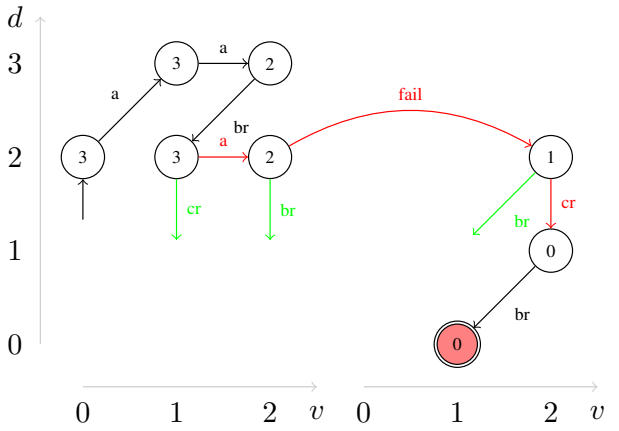

Figure 5: $\mathcal{A}\|\mathcal{P}\|\mathcal{L}'$ For the log $L' = a \cdot a \cdot br \cdot a \cdot fail \cdot cr \cdot br$. The level of choice is written in the states. In this case $fail$ is observable.

Definition 9 (Effective choice explanation) *Given a log L, a behavioural model $\mathcal{A}$ and an observer $\mathcal{P}$, let $\mathcal{A}_{|||} = \mathcal{A}\|\mathcal{P}\|\mathcal{L} = (S, \Sigma, \delta, s_0, F)$. Let $\delta_e^{nd} = \{(s, \sigma, sink) \mid (s, \sigma) \in \delta_{alt}(\mathcal{A}_{|||})\} \cup \delta_c(\mathcal{A}_{|||}) \cup \{(s, \tau, s') \mid \exists \sigma \in \Sigma, (s, \sigma, s') \in \delta \setminus \delta_c(\mathcal{A}_{|||})\}$, where $sink$ is a fresh sink state. We define $\mathcal{A}_e^{nd} = \{S \cup \{sink\}, \Sigma, \delta_e^{nd}, s_0, F\}$ as the non-deterministic automaton where the labels of all transitions that are not effective choice transitions have been replaced with τ.*

The effective choice explanation is the automaton $\mathcal{A}_e = (S', \Sigma, \delta_e, s'_0, F')$ obtained by determinization of $\mathcal{A}_e^{nd}$ where $S' \subset 2^S$, $s'_0 \in S'$ and $F' \subset S'$.

In the case the system is fully observable, $tr(L)$ is a singleton. Then, only a unique execution is to be considered to compute effective choice transitions.

Example 3 *Suppose that all events of our running example are observable. As a log, we take $L' = a \cdot a \cdot br \cdot a \cdot fail \cdot cr \cdot br$ and $\mathcal{L}' = ex(L')$. We have $\{L'\} = tr(L')$. Figure 5*

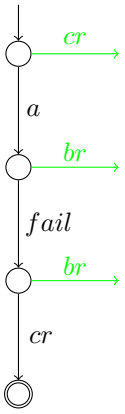

Figure 6: Effective choice explanation $\mathcal{A}_e$ for the fully observable system $\mathcal{A}$, the observer $\mathcal{P}$ and the log L' as described in the example 3.

shows $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$. As expected, there is only one accepted trace and it violates the safety property. We have added safe alternatives in green and computed effective choice transitions for the only execution. The first 3 transitions $(0,2,ok) \xrightarrow{a} (1,3,ok) \xrightarrow{a} (2,3,ok) \xrightarrow{br} (1,2,ok)$ lead to the state $(1,2,ok)$ which has the same level of choice as the initial state. The reason is that the event br repairs past erroneous actions, there is a compensation and those states are not directly related to the violation. For the last transition in black $(2,1,ko) \xrightarrow{br} (1,0,ko)$ the state $(2,1,ko)$ is already in the level of choice 0, it means that the violation is unavoidable. To compute the effective choice explanation, we replace the labels of these transitions drawn in black with τ. These transitions are not causally relevant for the violation of the safety property. With the determinization we obtain the effective choice explanation in Figure 6. In this automaton, the transitions in black are the effective choice transitions and the transitions in green are the safe alternatives (where the sink state is not shown). We can interpret this explanation: there is an acceleration in a state of choice at level 3 then a failure of the sensor that allows a cruise event which leads to a state where the violation of the safety property is unavoidable. The violation could have been avoided if instead of the first acceleration there had been a $cruise$ event, or instead of the failure of the sensor there had been a braking or instead of the $cruise$ event there had been a braking.

In the case the system is partially observable, there are different executions consistent with the log L, i.e. $tr(L)$ is not a singleton. We compose the explanations of each possible execution by considering all the effective choice transitions for every accepting execution as defined in Definition 7 to form the explanation.

Example 4 *Let us return to the running example with partial observability as defined in the case study. Figure 7 shows $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$. We have added safe alternatives in green and computed effective choice transitions for all accepting executions. Each one of those executions is consistent with the behavioural model $\mathcal{A}$ and with the log L. To compute the effective choice explanation, we change the transitions in black into τ-transitions, exactly like in the previous example. After determinization we obtain the effective choice*

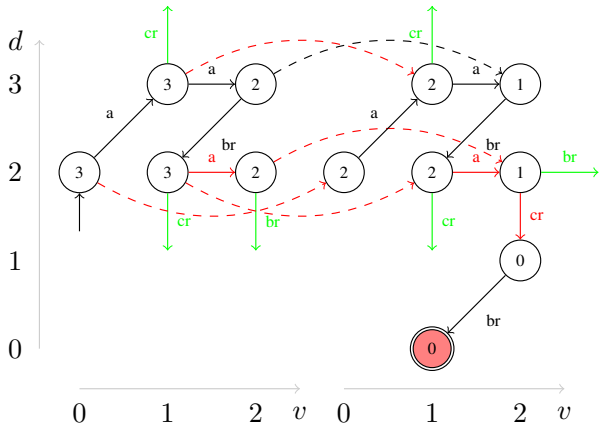

Figure 7: $\mathcal{A}\|\mathcal{P}\|\mathcal{L}$ with the log $L = a \cdot a \cdot br \cdot a \cdot cr \cdot br$, effective choice transitions in red, safe alternatives in green, the sink node is not represented.

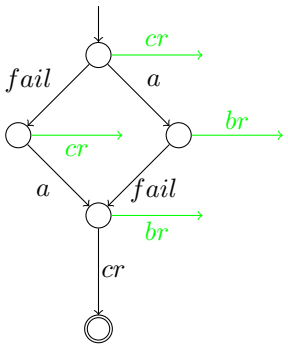

Figure 8: Effective choice explanation $\mathcal{A}_e$ for the partial observable system $\mathcal{A}$, the observer $\mathcal{P}$ and the log L as described in Example 4.

explanation in Figure 8. Contrary to the fully observable case, we have branching here. We can interpret this explanation as follows: there are 3 stages entailing the violation. In the first two stages, the acceleration and the failure of the sensor can occur in any order. Those two events lead to a state where the regulator wrongly enabled the cruise event cr. The last stage is the cr event that makes the violation unavoidable. In each non accepting state, the transitions of the safe alternatives explain how the violation could have been avoided.

5 Related work

The present work is closely related to the fault diagnosis problem in DES initially defined in [20]. In this seminal work, the system is represented as an automaton that explicitly represents the normal and abnormal behaviours. In particular, a system run is abnormal as soon as it contains an unobservable fault event. The fault diagnosis problem then consists in determining the set of fault events that can have happened in the partially observable runs that are consistent with the given log L. In the original fault diagnosis problem, there is no attempt to actually provide any kind of detailed explanation of the fault, the only re-

lationship between the diagnosed fault and the observation is the consistency with the model. On the other hand, several other approaches aim at fully reconstructing the complete set of runs that are consistent with the observations [2; 19] without any attempt to synthesize any more concise explanations of the faults. Fault diagnosis has then been extended to diagnose more complex behaviours, similar to our property violations, that are called either *logic specifications* in [21] or *event patterns* in [15; 10] but still the returned result is only the set of possible patterns that have occurred that are consistent with the observations. More recently, the work in [5] aims at providing to a human operator an interpretable explanation of the diagnosed fault as a subset of the observations/logs L called *critical observations*. Like choice-based explanations, critical observations can be represented as an automaton, but they only gather observable events. Critical observations then ensure that, whatever the full log L is, and as soon as L contains the critical observation, the diagnosis candidate F (set of fault events) remains the same (L is a sufficient condition for the fault diagnosis F). The present work follows the same principle that is to provide to a human operator an interpretable explanation for the violation of a given partially observable property, however, we propose explanations that are not necessarily fully observable but still interpretable with the help of the available model of the system.

Our proposal for generating explanations is also related to the *reveals* relation in a DES as defined in [11]. An event e reveals an event e' if e' occurs in any run of the system where e occurs: the occurrence of e is then a sufficient condition for the occurrence of e'. More generally speaking, explanation is a notion that is intrinsic to abductive diagnosis problems as logically defined in [6; 4] and more recently in [17]. In an abductive diagnosis problem, the purpose is to explain a set of *symptomatic observations* $OBS_S \subseteq L$ by fault modes M in the system usually represented by abducible predicates. Here the relationship between the explanation and the observation is logical entailment ($\models$) that is stronger than simple consistency. This means that there might be no diagnosis candidate (so no explanation) for a given OBS_S. Our proposal does not rely on symptomatic observations as in abductive diagnosis but on the violation of a partially observable property. If this violation generates a characteristic set of observations, then this set could be considered as symptomatic.

Several authors have proposed a construction of explanations for the satisfaction of a property P by an execution trace based on sub-sequences of the trace that are sufficient to entail P, such as explanatory diagnoses in the Situation Calculus [18] and causal compression in rule-based rewriting systems [7]. Our definition of subsequence-based explanations is similar to the NESS test ("necessary element of a sufficient set") in law [14] as it identifies minimal subsequences that are sufficient to entail the property violation. However, subsequence-based explanations do not convey any information about the outcome produced by alternative branches in a non-deterministic system. This issue is addressed by choice-based explanations.

Our definition of effective choices is inspired by [16], which leverages game theory to explain counterexample traces from model-checking by exhibiting the portions of the trace in which the system could have avoided the violation of an expected property no matter how the environment behaves. However, our approach disregards events that are "compensated" by other events later in the trace, hence yielding a more concise explanation. A similar idea of highlighting choice states in which the execution could have taken a different outcome is followed in [1], but without the fixpoint construction of [16] and in our approach, early faults that contribute to the system failure may be discarded from the explanation there. In contrast to our approach, [16; 1] assume full observability.

Counterfactual causation has been studied in many disciplines as a precise assessment of individual causes that contribute to bring about an effect. The influential definition of *actual causality* on a *structural equations model* (SEM) [12] has subsequently been adapted to enable reasoning about system dynamics. For instance, [3] defines causality for partially ordered sets of events. Again, these approaches assume full observability.

6 Conclusion

We have presented two constructions of explanations that are able to cope with partial observability of events. The first is based on minimal sub-sequences of the traces of the log that entail a violation of the property. The second approach is based on a construction of layers similar to [16], in which the explanation is constructed from the choices that definitely move the system closer to the violation of the property. Both approaches are complementary: while subsequence-based explanations are well suited to "condense" the execution trace in sequential portions of the model but are prone to keep non-pertinent parts such as initialisation sequences in the explanation, effective choice explanations highlight the "fateful" choices in an execution, as well as alternative events that would have helped avoid the outcome. Effective choice explanations are therefore able to explain failures stemming from non-deterministic choices, such as concurrency bugs.

In this work we have focused on defining explanations in terms of behaviors over events. In future work we intend to study the use of information from states such as valuations of variables (or abstractions thereof) to convey additional information in the explanations. The development of algorithms for efficient and/or on-the-fly construction of explanations is also left for future work. More fundamentally, we are interested in developing a principled approach in the spirit of [9] to construct definitions of explanations based on a set of formal requirements.

References

[1] Gianluca Barbon, Vincent Leroy, and Gwen Salaün. Debugging of concurrent systems using counterexample analysis. In *Fundamentals of Software Engineering - 7th International Conference, FSEN 2017, Tehran, Iran, April 26-28, 2017, Revised Selected Papers*, pages 20–34, 2017.

[2] Pietro Baroni, Gianfranco Lamperti, Paolo Pogliano, and Marina Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183, May 1999.

[3] Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. Symbolic causality checking using bounded model checking. In Bernd Fischer and Jaco Geldenhuys, editors, *Model Check-*

ing Software, pages 203–221, Cham, 2015. Springer International Publishing.

[4] Vittorio Brusoni, Luca Console, Paolo Terenziani, and Daniele Theseider Dupré. A spectrum of definitions for temporal model-based diagnosis. *Artificial Intelligence*, 102(1):39–79, June 1998.

[5] Cody James Christopher and Alban Grastien. Formulating event-based critical observations in diagnostic problems. In *26th International Workshop on Principles of Diagnosis (DX-15)*, pages 119–126, 2015.

[6] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.

[7] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Chris Thompson-Walsh, and Glynn Winskel. Graphs, rewriting and pathway reconstruction for rule-based models. In D. D'Souza, T. Kavitha, and J. Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 276–288. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012.

[8] Teresa Escobet, Anibal Bregon, Belarmino Pulido, and Vicenç Puig, editors. *Fault Diagnosis of Dynamic Systems: Quantitative and Qualitative Approaches*. Springer, 2019.

[9] Gregor Gössler and Jean-Bernard Stefani. Causality analysis and fault ascription in component-based systems. Research Report RR-9279, Inria - Research Centre Grenoble – Rhône-Alpes, June 2019.

[10] Houssam-Eddine Gougam, Audine Subias, and Yannick Pencolé. Supervision patterns: formal diagnosability checking by Petri net unfolding. *IFAC Proceedings Volumes*, 46(22):73 – 78, 2013. 4th IFAC Workshop on Dependable Control of Discrete Systems.

[11] Stefan Haar, Christian Kern, and Stefan Schwoon. Computing the reveals relation in occurrence nets. In Giovanna D'Agostino and Salvatore La Torre, editors, *Gandalf*, volume 54 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–44, Minori, Italy, June 2011.

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

[13] Walter Hamscher, Luca Console, and Johan de Kleer, editors. *Readings in Model-based Diagnosis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[14] Herbert Lionel Adolphus Hart and Tony Honoré. *Causation in the law*. Oxford University Press, 2nd edition, 1985.

[15] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *2006 8th International Workshop on Discrete Event Systems*, pages 262–268, July 2006.

[16] HoonSang Jin, Kavita Ravi, and Fabio Somenzi. Fate and free will in error traces. *STTT*, 6(2):102–116, 2004.

[17] Roxane Koitz-Hristov and Franz Wotawa. Applying algorithm selection to abductive diagnostic reasoning. *Applied Intelligence*, pages 1–19, 5 2018.

[18] Sheila A. Mcllraith. *Explanatory diagnosis: Conjecturing actions to explain observations*, pages 155–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[19] Yannick Pencolé and Marie-Odile Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(1):121–170, May 2005.

[20] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[21] Shengbing Jiang and Ratnesh Kumar. Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Transactions on Automatic Control*, 49(6):934–945, June 2004.