Causality Analysis for Concurrent Reactive Systems

Rayna Dimitrova, Rupak Majumdar, Vinayak S. Prabhu[1]

1 MPI-SWS `{rayna, rupak, vinayak}@mpi-sws.org`

Abstract

We present a comprehensive language theoretic causality analysis framework in the setting of concurrent reactive systems. Our framework allows us to uniformly express a number of causality notions studied in the areas of artificial intelligence and formal methods, as well as define new ones that are of potential interest in these areas. Furthermore, our formalization provides means for reasoning about the relationships between individual notions which have mostly been considered independently in prior work; and allows us to judge appropriateness of the different definitions for various applications in system design. In particular, we consider causality analysis notions for debugging, error resilience, and for liability resolution in concurrent reactive systems. Finally, we derive automata based algorithms for computing various causal sets based on our language theoretic encoding, and derive the algorithmic complexities.

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Causality analysis, which investigates questions of the form "Does event e_1 cause event e_2?" plays an important role in many areas of science, medicine and law. In formal methods, causality analysis has been used to determine the *coverage of specifications* [5] (that is, which parts of the system under scrutiny are relevant for the satisfaction of a specification), to *explain counterexamples* [2] (identify points in a counterexample trace that are relevant for the failure of a temporal specification), to fault tree construction [18], and to *automatically refine system abstractions* [4]. In artificial intelligence, causality-based explanation finding has applications in natural language processing, automated medical diagnosis, vision processing, and planning. Resolving liabilities in a legal setting often relies on establishing the causal relations between potential causes and the occurred damage [3].

Causality definitions based on *counterfactuals*, which are alternative scenarios where the suspected cause e_1 of e_2 did not happen, date back to [13] and have been extensively studied in philosophy (cf. [19]). In computer science, the most prominent and widely used definition of causality is that of [12], in which the authors write "... while it is hard to argue that our definition (or any other definition, for that matter) is the right definition, we show that it deals with the difficulties that have plagued other approaches in the past ...". Halpern and Pearl's approach is based on *structural equations*, which describe causal dependencies between Boolean variables. We extend the Boolean study of causality to the *temporal* setting; specifically, we formalize notions of causality in *concurrent reactive systems* whose behaviors evolve over time. A concurrent reactive system is a composition of interacting components; the system behavior is determined by the *repeated* interaction between the components over time. We consider the setting where component implementations are not available for analysis and the designer can only rely on specifications of their expected behavior. Thus, when analyzing an *error trace* (an execution of the system that violates a desired system-level property), the only available information about the system is the components' specifications and the observed trace.

Recently causality analysis of component-based systems has drawn a lot of attention [8, 7, 21, 6, 9, 23, 10]. The goal is to identify a subset of the components, which have violated their *local* specifications, that are actually responsible for the violation of the *system-level* property. This requires integrating the temporal order of events [16, 17, 1] in the analysis of logical causality. The main challenge lies in defining the set of *counterfactual traces* for a given observed trace **tr**. These are traces used to reason about hypothetical scenarios where a subset of the system components behave in a way that differs from the trace **tr**. Different approaches differ in the way they account for the dependencies between the behaviors of different components, that is, how changing the behavior of one component affects the behavior of others. The available information, a single observed system trace and the components' specifications, is often insufficient to faithfully reconstruct these alternative behaviors. Existing approaches, hence, choose a specific set of trace reconstruction rules as a basis of

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Acces; Article No. 23; pp. 23:1–23:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

their causality notion. However, the suitability of a notion depends on the desired application. For example, while liability resolution requires conservative notions that give high confidence in their determinations of causes of failure, for system analysis and debugging less conservative notions are more appropriate, provided that they are cost-effective and focus on relevant components. One of the limitations of existing work in this area is that the various causality notions have been studied in isolation but no framework for comparing different notions of causality has been provided so far.

We present a language theoretic causality framework for concurrent reactive systems incorporating diverse counterfactual trace sets. A cause for a violation is a component set. Our analysis reasons about two classes of scenarios to determine if component set $\mathfrak{C}$ is a cause (for an observed system fault):

- *Fault Mitigation Capability* analysis asks whether the *correct behavior* of the component set $\mathfrak{C}$ is enough to mitigate the faults of all components (*including those of components not in* $\mathfrak{C}$), by ensuring that the required system property holds.
- *Fault Manifestation* analysis asks whether the observed *faulty behavior* of the component set $\mathfrak{C}$ is enough to manifest a global fault (*i.e.*, a system-property violating global behavior), even if the components not in $\mathfrak{C}$ were to behave correctly.

These two classifications parallel the classifications of [8, 7] of causes into *necessary causes* and *sufficient causes*. However, our analysis is not limited to specific definitions of counterfactual sets. In contrast, we provide a reasoning framework based on generic counterfactual sets, and introduce several natural instantiations.

We will use the following example throughout the paper to illustrate the key notions of our framework for causality analysis.

▶ **Example 1.** Consider a system with three components, C_1, C_2 and C_3, with a common shared resource. Access to the resource is regulated by C_3, and there are in total M units of the resource available per unit of time. In particular, consider a solar battery for which the charge rate is M energy units per time unit. If the initial charge is $E > M$, then the components cannot utilize more than M units of energy for more than $E - M$ steps. Thus, to be safe, we require that in each step the combined consumption should be at most M. This system-level requirement is denoted as φ (in a concrete execution, however, components can consume more than the allowed M units for a small number of steps without dire consequences).

With a view towards robust satisfaction of φ, the local specifications of the components constrain their behaviors further than what is absolutely necessary for the satisfaction of this property. For example, let the specification φ_3 of C_3 require that the resource allocation respects a given safety margin, namely, that the combined allocation by C_3 to all the components should not be more than $M - 1$ units in any step. Furthermore suppose that φ_3 specifies that, if component C_1 or C_2 performs a violation, that is, consumes more units of energy than it has been allocated, then C_3 should attempt to compensate for that by reducing its own consumption. More concretely, if at time t, component C_3 indicates that C_1 should consume at most Δ units in the next time instant $t + 1$, and at $t + 1$ component C_1 consumes $\Delta + \alpha$ instead, then C_3 must decrease its own consumption at time step $t + 2$ (from the consumption at time $t + 1$) by α, if possible (or reduce it to 0 if not), in an attempt to prevent a violation of the global property φ. Note that there is a delay of one time unit for components to react to their inputs. Let the only requirement on C_1 (and also on C_2) be that if C_3 allocates it a resource units in the current step, then it will consume at most a units in the *next* step. Consider the following trace (with the global requirement φ that the combined resource consumption be at most 31), where C_3 allocates at most 30 units in total to all components:

step	1	2	3	4
allocated to C_1 for next time step	6	6	6	6
consumption C_1 at current time step	0	6	6	6
allocated to C_2 for next time step	12	12	12	12
consumption C_2 at current time step	0	18	20	20
allocated to C_3 for current time step	0	7	7	7
consumption C_3 at current time step	0	7	7	7

Observe that the combined consumption is 33 from step 3 onwards, violating the limit 31. C_2 exceeds its limit by 6 units at the first step, and by 8 units after that. C_3 is supposed to decrease its consumption from 7 units in step 2, to 1 unit in step 3 (and then to 0 units in step 4); but it violates its local specification and does not do this.

Had C_3 reduced its consumption as required, the global violation would not have occurred, even in the presence of C_2's observed incorrect behavior. A causality analysis should report the component set $\{C_3\}$ as one of the possible causes for the observed violation.

Both singleton sets $\{C_2\}$ and $\{C_3\}$ of components have the *capability to mitigate* the observed error, *i.e.* the correct behavior of either component would have prevented the violation of the global requirement φ. Dually, both components C_2 and C_3, *i.e.* the component set $\{C_2, C_3\}$, have to behave incorrectly as observed in order for the trace to *manifest* the observed error. ◀

Contributions. We present a systematic, language theoretic study of causality for component-based concurrent reactive systems:

- We first describe a modular decomposition of counterfactual tracesets based on (i) hypotheses on possible incorrect behaviors (differing from the single observed trace); and (ii) interactions between different components due to the concurrent reactive nature of the components.
- Next, we show how composed counterfactual tracesets can be used to define various notions of causality in a uniform fashion (Equations (1) through (4)). Our approach uses basic language theoretic operations to reason about intricate *consistency* issues: issues which arise when repeated component interactions have to be reasoned about (*e.g.*, two components are faulty, we repair one, and this leads to a different sequence of inputs to the unfixed faulty one).
- We demonstrate that the generality and modularity of our definition of causality allow us to seamlessly extend causality analysis to the case of heterogeneous fault models, where different components are examined under different fault scenarios.
- Our unified approach allows us to compare the resulting different causality notions, and the relationships between the causal sets, and thus to indicate the situations in which each of them is most appropriate.
- We present an automata-based method for determining various causal sets in the setting of heterogeneous component-fault models, and derive its algorithmic complexity.

2 Preliminaries

Languages. Let Σ be a non-empty finite alphabet. A *word* or a *trace* $w = \sigma_1, \sigma_2, \ldots, \sigma_m$ over Σ is a finite sequence of letters from Σ. We denote by $w_{[i]}$ the i-th symbol σ_i in the word w, and by $w[i..j]$ the substring $\sigma_i, \ldots, \sigma_j$. Σ^* is the set of all words over Σ, and ϵ is the empty word. A *language* is a set of words. The concatenation of two words u, w is denoted $u \cdot w$; and similarly for languages. For a word w, $\mathsf{len}(w)$ is the length of w. For a language L and a positive integer k, let $L_{|k|}$ denote the words in L which have exactly k letters. A word u is a *prefix* of a word v, denoted $u \preceq v$, iff there exists a word w such that $v = u \cdot w$. For a language L, the language $\mathsf{Prefs}(L)$ consists of the prefixes of words in L. We write $u \prec v$ when u is a strict prefix of v, that is $u \preceq v$ and $u \neq v$. Given two words u, v, let $\mathsf{lcommpref}(u, v)$ denote the longest common prefix of u and v. A language L is said to be *prefix closed* if whenever a word $v \in L$, we have that every prefix of v is also in L, i.e., $\mathsf{Prefs}(L) \subseteq L$[1].

Languages over Variables. For the purpose of modelling reactive systems in which components communicate via shared variables, we let an alphabet be a set of possible valuations of a set of variables over a given finite set. If an alphabet Σ is defined over a set of variables X we denote this by $\Sigma[X]$, omitting $[X]$ when X is clear from the context. Thus, a letter $\sigma \in \Sigma[X]$ is a function $\sigma : X \mapsto \cup_{x \in X} \mathcal{D}_\Sigma(x)$ where $\mathcal{D}_\Sigma(x)$ is the (finite) domain of the variable x. A word w over $\Sigma[X]$ is a sequence of valuations for X, *i.e.* every letter $w_{[i]}$ is a valuation of all variables in X. If $\Sigma[X]$ and $\Pi[Y]$ are alphabets with $Y \subseteq X$, and $w \in \Sigma^*$, then $w|_\Pi$ is the projection of w on Π defined in the usual way.

Alphabet and Language Composition. Given $\Sigma_1[X_1], \ldots, \Sigma_n[X_n]$ for which we have that for every variable x such that $x \in X_i$ and $x \in X_j$ for $i \neq j$, we have $\mathcal{D}_{\Sigma_i}(x) = \mathcal{D}_{\Sigma_j}(x)$ (*i.e.* common variables have the same domain in each alphabet), we define the *composite alphabet* $\Sigma_1[X_1] \parallel \cdots \parallel \Sigma_n[X_n]$ to

[1] Prefix-closed languages need not be regular

be the alphabet $\Sigma[X]$ such that $X = \cup_{i=1}^{n} X$; such that the domain of a variable x is $\mathcal{D}_{\Sigma}(x) = \mathcal{D}_{\Sigma_i}(x)$ for $x \in X_i$. Given languages $L_i \subseteq \Sigma_i^*$ over Σ_i for $i = 1, \dots, n$, we define the *language composition* of $L_1, \dots, L_n$ to be the language: $L_1 \parallel \cdots \parallel L_n = \{w \in \Sigma^* \mid w|_{\Sigma_i} \in L_i \text{ for all } i\}$, over $\Sigma[X]$.

▶ **Example 2** (Languages and composition)**.** We consider a language L_1 over an alphabet $\Sigma_1[X_1]$, with $X_1 = \{x_1, x_2\}$ and domain $\mathcal{D}_{\Sigma_1}(x_1) = \mathcal{D}_{\Sigma_1}(x_2) = \{0, 1\}$; and a language L_2 over an alphabet $\Sigma_2[X_2]$, with $X_2 = \{x_2, x_3\}$, also with Boolean domain. The language L_1 is:

$$\{(x_1\colon b_1^1, x_2\colon b_2^1),\ (x_1\colon b_1^2, x_2\colon b_2^2),\ ..,\ (x_1\colon b_1^m, x_2\colon b_2^m) \mid b_1^j{=}b_2^j \text{ for all } 1{\le}j{\le}m\}$$

i.e. words where the values of x_1 and x_2 are the same. L_2 consists of words with equal valued x_2 and x_3:

$$\{(x_2\colon b_2^1, x_3\colon b_3^1),\ (x_2\colon b_2^2, x_3\colon b_3^2)\,,..,\ (x_2\colon b_2^m, x_3\colon b_3^m) \mid b_2^j{=}b_3^j \text{ for all } 1{\le}j{\le}m\}.$$

The language $L_1 \parallel L_2$ over $\Sigma[\{x_1, x_2, x_3\}]$ is defined as:

$$\left\{ \begin{array}{r|l} (x_1\colon b_1^1, x_2\colon b_2^1, x_3\colon b_3^1),\ (x_1\colon b_1^2, x_2\colon b_2^2, x_3\colon b_3^2),\ \dots & m \ge 0 \text{ and} \\ (x_1\colon b_1^m, x_2\colon b_2^m, x_3\colon b_3^m) & b_1^j{=}b_2^j{=}b_3^j \in \{0,1\} \text{ for all } 1 \le j \le m \end{array} \right\}$$

i.e. words where x_1, x_2, x_3 have the same value at each step. ◀

Component Model. A *component specification* is a tuple $C = (X, \mathsf{inp}(X), \mathsf{out}(X), \Sigma, \varphi)$, where

- $X = \mathsf{inp}(X) \uplus \mathsf{out}(X)$ is the set of variables of the component, consisting of the input variables $\mathsf{inp}(X)$ and the output variables $\mathsf{out}(X)$ (the sets of input and output variables being disjoint);
- Σ is the alphabet, consisting of all possible valuations of the variables X;
- φ is a non-empty prefix-closed language over Σ, specifying the set of correct behaviours of C.

For a letter $\sigma \in \Sigma$ and a variable $x \in X$ we denote with $\sigma(x)$ the value of x according to σ. The input alphabet $\mathsf{inp}(\Sigma)$ of C consists of the possible valuations of the input variables $\mathsf{inp}(X)$, and, similarly, the output alphabet $\mathsf{out}(\Sigma)$ consists of valuations of $\mathsf{out}(X)$.

▶ **Example 3** (Component)**.** Component C_1 from the example in the introduction can be modelled as $C_1 = (X_1, \mathsf{inp}(X_1), \mathsf{out}(X_1), \Sigma_1, \varphi_1)$, where[2] $X_1 = \{x_a^{3,1}, x_d^{1,3}\}$, and $\mathsf{inp}(X_1) = \{x_a^{3,1}\}$, and $\mathsf{out}(X_1) = \{x_d^{1,3}\}$; the alphabet Σ_1 consists of the possible valuations of $x_a^{3,1}$ and $x_d^{1,3}$ ranging over $[0, M]$, and φ_1 contains strings $w \in \Sigma_1^*$ such that either (i) w is the empty string; or (ii) $w_{[1]}(x_d^{1,3}) = 0$ and $w_{[j+1]}(x_d^{1,3}) \le w_{[j]}(x_a^{3,1})$ for all $2 \le j + 1 \le \mathsf{len}(w)$. Intuitively, the value of $x_a^{3,1}$ specifies the units of resource ***allocated*** to C_1 by C_3 for the next step, the value of $x_d^{1,3}$ specifies the units ***depleted*** by C_1 in the current step (this number is given as input to C_3) . The specification φ_1 ensures that at each step C_1's consumption does not exceed the bound specified by the value of $x_a^{3,1}$ in the previous step. ◀

Component Compositions, Systems, & Global Specifications. Given a set of components $\mathfrak{C} = \{C_1, \dots, C_n\}$ where each $C_i = (X_i, \mathsf{inp}(X_i), \mathsf{out}(X_i), \Sigma_i, \varphi_i)$, the component composition $C_1 \| \dots \| C_n$ is defined in case the following two conditions both hold.

1. The sets of output variables are pairwise disjoint, *i.e.*, if $\mathsf{out}(X_i) \cap \mathsf{out}(X_j) = \emptyset$ for $i \neq j$; and
2. the composite alphabet $\Sigma_1[X_1] \parallel \cdots \parallel \Sigma_n[X_n]$ exists.

The composition $C_1 \parallel \cdots \parallel C_n$ is the component $(X_{\mathfrak{C}}, \mathsf{inp}(X_{\mathfrak{C}}), \mathsf{out}(X_{\mathfrak{C}}), \Sigma_{\mathfrak{C}}, \varphi_{\mathfrak{C}})$ defined as follows

- $X_{\mathfrak{C}} = \cup_{i=1}^{n} X_i$ is the set of all variables;
- $\mathsf{out}(X_{\mathfrak{C}}) = \cup_{i=1}^{n} \mathsf{out}(X_i)$, *i.e.*, the set $\mathsf{out}(X_{\mathfrak{C}})$ consist of all output variables of all components.
- $\mathsf{inp}(X_{\mathfrak{C}}) = \left(\cup_{i=1}^{n} \mathsf{inp}(X_i)\right) \setminus \mathsf{out}(X_{\mathfrak{C}})$ *i.e.*, the set of input variables contains those input variables which are are not output variables of any component in $\mathfrak{C}$.
- $\Sigma_{\mathfrak{C}}$ is the composite alphabet $\Sigma_1[X_1] \parallel \cdots \parallel \Sigma_n[X_n]$.
- $\varphi_{\mathfrak{C}}$ is the composite language $\varphi_1 \parallel \cdots \parallel \varphi_n$.

A collection of composable components is called a *system*. Given a system $\mathcal{S} = \{C_1, \dots, C_n\}$, a (global) system *specification* φ is a language over the composite alphabet $\Sigma_1 \parallel \cdots \parallel \Sigma_n$. In this work, we require that the system specification φ be prefix closed, and in addition, that φ contains $\varphi_1 \parallel \cdots \parallel \varphi_n$. Thus, the global requirement is more relaxed than the promised behaviors of the individual

[2] In naming variables in the examples, we follow the convention that a variable $x^{k,i_1,i_2,\dots,i_j}$ (i) is common to the components $C_k, C_{i_1}, \dots, C_{i_j}$; and (ii) is an output variable of component C_k, and an input variable to components $C_{i_1}, \dots, C_{i_j}$.

components. In other words, the system $\{C_1, \ldots, C_n\}$ promises to *implement* or *refine* the global requirement φ. Abusing notation, we let $\mathcal{S}$ also denote the component composition $C_1 \parallel \cdots \parallel C_n$.

Note. The composition of components as defined above implies that components execute synchronously in lock-step[3]. All definitions and result presented in this paper can be easily extended to the asynchronous setting, which we do not do here for the sake of simplicity of the presentation.

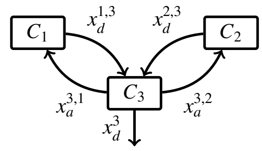

Figure 1 Resource sharing system described in Example 4.

System Traces. A *global trace* of $\mathcal{S}$ is a word $\mathbf{tr} \in \Sigma^*$. The trace $\mathbf{tr}$ is *correct* if $\mathbf{tr} \in \varphi$; otherwise it is an *error* trace. A *local trace* for component C_i is a word $w \in \Sigma_i^*$.

▶ **Example 4** (Systems and Traces)**.** We define component $C_2 = (X_2, \mathsf{inp}(X_2), \mathsf{out}(X_2), \Sigma_2, \varphi_2)$ analogously to C_1 in our running example. Let component $C_3 = (X_3, \mathsf{inp}(X_3), \mathsf{out}(X_3), \Sigma_3, \varphi_3)$, where $\mathsf{inp}(X_3) = \{x_d^{1,3}, x_d^{2,3}\}$, and $\mathsf{out}(X_3) = \{x_a^{3,1}, x_a^{3,2}, x_d^3\}$. The alphabet Σ_3 consists of the possible valuations of the variables in $X_3 = \mathsf{inp}(X_3) \cup \mathsf{out}(X_3)$, the range of each variable being $[0, M]$. The variables $x_d^{1,3}, x_d^{2,3}$ denote the current-step depletions of the resource by C_1 and C_2 respectively, the values of which are read in by component C_3. The value of x_d^3 is the depletion by C_3. The variables $x_a^{3,1}, x_a^{3,2}$ are the allocations of the resource to C_1 and C_2 for the next step. The system is depicted in Figure 1.

The local specification φ_3 of C_3 is defined as containing all words $w \in \Sigma_3^*$ which satisfy each of the following four requirements (letting $\sigma_j = w_{[j]}$, and $\sigma_{j+1} = w_{[j+1]}$, and $\sigma_{j+2} = w_{[j+2]}$),

1. φ_3'''' specifies that for every $j \leq \mathsf{len}(w)$ we have $\sigma_j(x_a^{3,1}) + \sigma_j(x_a^{3,2}) + \sigma_{j+1}(x_d^3) \leq M - 1$; i.e., the planned combined depletion at step $j+1$ should be at most $M - 1$ (leaving a safety margin of 1).
2. φ_3''' specifies that $w_{[1]}((x_d^3) = 0$, *i.e.* C_3 should not deplete the resource at all in the first step.
3. φ_3'' specifies how component C_3 should change its behavior at step $j+2$ based on C_2's behavior at step $j+1$. It requires one of the following conditions to hold.
   - $\sigma_{j+1}(x_{d+1}^{2,3}) \leq \sigma_j(x_a^{3,2})$, *i.e.*, the depletion by C_2 in step $j+1$ is at most what it was allocated to it by C_3 in the previous step.
   - If $\sigma_{j+1}(x_{d+1}^{2,3}) > \sigma_j(x_a^{3,2})$, *i.e.*, if the previous case does not hold, then: $\sigma_{j+2}(x_d^3) \leq \max\left(0, \sigma_{j+1}(x_d^3) - \left(\sigma_{j+1}(x_{d+1}^{2,3}) - \sigma_j(x_a^{3,2})\right)\right)$.

   That is, if C_2 exceeds its allocation by an amount α at step $j+1$, then C_3 reduces its own consumption at step $j+2$ from that at step $j+1$ by α (if possible).
4. φ_3' is the condition analogous to φ_3'' for component C_1, and specifies how component C_3 should change its behavior based on C_1's behavior.

The system specification of $C_1 \parallel C_2 \parallel C_3$ is defined to be the language φ containing words w such that for every j the combined depletion is at most M. Formally, φ equals:

$$\{w \in \Sigma^* \mid \text{ for all } j, \text{ we have } w_{[j]}(x_d^3) + w_{[j]}(x_d^{1,3}) + w_{[j]}(x_d^{2,3}) \leq M\}$$

We present two sample traces, letting $M = 31$ (the global specification φ is that at each step, the combined depletion for that step must not exceed 31). The first trace satisfies φ and is as follows. Note that even though C_2 violates its local spec φ_2 in steps 2 and 3 (as it depletes by 16 units when it was allowed only 10 as specified by $x_a^{3,2}$ in steps 1 and 2), the global specification is still satisfied due to C_3 reducing its own depletion amount at step 3 from 10 (in the previous step) to 4.

j, φ	$1, \varphi$	$2, \varphi$	$3, \varphi$
local specs	$\varphi_1, \varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \varphi_3$
$x_a^{3,1}$ and $x_a^{3,2}$	10	10	10
$x_d^{1,3}$	0	4	10
$x_d^{2,3}$	0	16	16
x_d^3	0	10	4

[3] The components in our examples are such that in every step of the system execution each component reads the values of its input variables that were written in the previous step and updates its output variables. This delay is not imposed by the formal model.

The second trace given to the right violates φ. Component C_3 violates its local specification φ_3 at step 3 because it should have reduced its consumption (from that in step 2) by α where α is the amount by which the resource depletion by C_2 exceeded its allocated 10 units (in this case $\alpha = 6$ units).

j, φ	$1, \varphi$	$2, \varphi$	$3, \varphi$	$4, \neg\varphi$
local specs	$\varphi_1, \varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \neg\varphi_3$	$\varphi_1, \neg\varphi_2, \neg\varphi_3$
$x_a^{3,1}$	10	10	10	10
$x_a^{3,2}$	10	10	10	10
$x_d^{1,3}$	0	6	6	8
$x_d^{2,3}$	0	16	16	16
x_d^{3}	0	8	8	8

3 A Framework for Causality

In this section, we fix a system $\{C_1, \ldots, C_n\}$, where $C_i = (\Sigma_i, \mathsf{inp}(\Sigma_i), \mathsf{out}(\Sigma_i), \varphi_i)$; a specification φ; and an observed trace $\mathbf{tr} \notin \varphi$[4]. We also fix a non-empty collection of components $\mathfrak{C} \subseteq \{C_1, \ldots, C_n\}$ for causality analysis. Let $\overline{\mathfrak{C}}$ be the components not in $\mathfrak{C}$. Assume, w.l.o.g., $\mathfrak{C} = \{C_1, C_2, \ldots, C_{n_\mathfrak{C}}\}$ (thus, $\overline{\mathfrak{C}} = \{C_{1+n_\mathfrak{C}}, \ldots, C_n\}$). Let $\Sigma_\mathfrak{C}$ denote the composition of the alphabets of the components of $\mathfrak{C}$.

3.1 Counterfactual Traces & Faulty Behaviors

Counterfactual Traces. Informally, the set of counterfactual traces for a given observed trace **tr**, consists of traces obtained from **tr** by correcting the behavior of some faulty components. These traces are used to reason about hypothetical scenarios where a subset of the components behave (correctly) in a way that differs from the incorrect behavior in the observed trace **tr**. Depending on the hypothetical scenario, the set of counterfactual traces is obtained as (a subset of) the composition of trace sets of individual component behaviors appropriately altered with respect to the trace **tr**.

In *reactive systems*, the behaviors of individual components are intertwined; This results in consistency dependencies between the component behaviors that must be taken into account. As the effect of the change in behaviors of other components that affect a particular component C_i is not easily determined, there does not exist a unique definition of counterfactual tracesets for C_i that is applicable for all purposes. We present different constructions of counterfactual tracesets, and indicate the situations in which each is useful. In each of these constructions, the set of counterfactual traces for component a C_i whose observed behavior in **tr** is incorrect will include *some of the correct behaviors* of C_i (according to φ_i), as well as *some incorrect behaviors*. The latter are determined according one of the *fault models F1* and *F2* (presented below).

Counterfactual Sets of Incorrect Behaviors: In this paper we consider several possible scenarios regarding the counterfactual behaviors of the incorrectly behaving components, described in the following list. It is important to note that these are just a few representative scenarios among all that can be captured within our framework. For all components C_i,

F1. *the only incorrect* local traces for component C_i that may be included in counterfactual sets are $\mathbf{tr}|_{\Sigma_i}$ and its prefixes. Essentially, this fault model assumes that if the inputs to C_i change, then the faulty behavior disappears, and is replaced by correct behaviors (according to φ_i) over the new input. Thus we assume that the faulty behavior of C_i was *only* for the particular input in $\mathbf{tr}|_{\Sigma_i}$.

F2. *the only incorrect* local traces for component C_i that may be included in counterfactual sets are ones that agree with both: (i) w_{mxp}, where w_{mxp} is the maximal correct prefix (with respect to φ_i) of $\mathbf{tr}|_{\Sigma_i}$; and (ii) $\mathbf{tr}|_{\mathsf{out}(\Sigma_i)}$. Thus, any counterfactual trace must be as the original one of C_i till the first error there, and after that it must follow the same sequence of $\mathsf{out}(\Sigma_i)$ output symbols as $\mathbf{tr}|_{\mathsf{out}(\Sigma_i)}$. This model implies that after the first error in C_i, if the input were to change, C_i would either (a) behave correctly on the new input, or (b) ignore the new input altogether and output the same sequence of output symbols as in the original trace $\mathbf{tr}|_{\mathsf{out}(\Sigma_i)}$.

While each fault model is imperfect, there is not much else that can be done given that the input for the analysis consists only of the properties $\varphi_1, \ldots, \varphi_n$ and a single execution trace **tr**. Thus, there is no mechanism to predict what the output of a component will be when its input changes, without

[4] Global system traces (obtained by composing the local traces of individual components) are denoted in bold font.

paying the cost of running additional simulations. If such additional data is available (as in [23]), it can be easily incorporated in our models. In our work, we focus on the fault models *F1* and *F2*. ◻

Now we present several ways of constructing counterfactual tracesets, which use the notion of the *maximal correct prefix* of the observed trace. The *maximal correct prefix* of the trace $\mathbf{tr}$, denoted $\mathsf{maxcp}(\mathbf{tr})$ is defined to be the maximal prefix $\mathbf{tr}_{\mathsf{mxp}} \preceq \mathbf{tr}$ that satisfies all local specifications, *i.e.* (a) $\mathbf{tr}_{\mathsf{mxp}}$ projected onto Σ_i is a subset of φ_i for all i; and (b) for every prefix $\mathbf{tr}_p$ of $\mathbf{tr}$ such that $\mathbf{tr}_{\mathsf{mxp}}$ is a strict prefix of $\mathbf{tr}_p$, there is a j such that $\mathbf{tr}_p|_{\Sigma_j} \notin \varphi_j$.

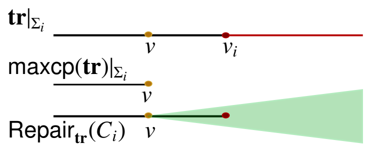

Figure 2 Traceset $\mathsf{Repair}_{\mathbf{tr}}(C_i)$.

Local Counterfactual Tracesets. We define the following counterfactual tracesets for component C_i.

★ $\mathsf{Repair}_{\mathbf{tr}}(C_i)$ defined as: $\{w \in \varphi_i \mid \mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i} \preceq w\}$.
That is, we keep the prefix $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$ for component C_i, and then take all possible correct C_i behavior extensions following $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$; *i.e*, we *repair* the errors following $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$, as well as the effects in C_i of errors in other components after $\mathsf{maxcp}(\mathbf{tr})$. Observe that $\mathsf{Repair}_{\mathbf{tr}}(C_i)$ is a subset of φ_i. Intuitively, this set captures the set of possible outcomes of C_i after $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$, if no error had occurred in *any* component. We illustrate this traceset in Figure 2.

The first trace in the Figure is the local trace for component C_i, obtained as the projection of the global trace on Σ_i. The point v_i denotes the place of the first violation of the local property φ_i by C_i. The point v denotes the place of the first violation of *some* local property φ_k by C_k where k may be different from i. Thus, the portion of $\mathbf{tr}|_{\Sigma_i}$ until v is equal to $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$. The set $\mathsf{Repair}_{\mathbf{tr}}(C_i)$ is obtained by taking the cone of all correct executions of C_i from the prefix $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$.

Observe, as depicted in the Figure, that there might be a strict prefix $\mathbf{tr}_p \prec \mathbf{tr}$ such that $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i} \prec \mathbf{tr}_p|_{\Sigma_i}$, and $\mathbf{tr}_p|_{\Sigma_i} \in \varphi_i$, *i.e.*, component C_i might continue to behave correctly in $\mathbf{tr}|_{\Sigma_i}$ after $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$; however the behavior after $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$ is considered to be *tainted*. This is because after $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$ there is some component which behaves incorrectly, and that incorrect behavior might affect other components. Thus, we consider the cone of all possible behaviors after $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$. Before $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$, no component is in error, and all are behaving according to their specifications; thus, we need not consider alternate traces before $\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}$.

★ $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ is defined as: $\mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_i}) \cup \mathsf{Repair}_{\mathbf{tr}}(C_i)$.
This traceset is obtained by adding all the prefixes of the observed (possibly incorrect) trace $\mathbf{tr}|_{\Sigma_i}$ to the set $\mathsf{Repair}_{\mathbf{tr}}(C_i)$. Thus, this traceset consists of all local traces for C_i, that are considered feasible according to either the observed trace; or to the promised behavior of component C_i after the prefix $\mathsf{maxcp}(\mathbf{tr})$. This set models the faulty behavior of $\mathfrak{C}$ under fault model *F1*.

In $\mathsf{Feasible}(C_i)$, we take the prefix set of the incorrect behavior, instead of only the whole incorrect trace, because we want the causality analysis to be robust: the analysis should consider every intermediate trace prefix which is in error. We also include correct behaviors in $\mathsf{Feasible}(C_i)$, because (i) although we want $\mathsf{Feasible}(C_i)$ to model incorrect behaviors, we do not want other components to *count on* C_i behaving incorrectly; and (ii) correcting the behavior of some components might lead to inconsistencies with the original local incorrect traces.

★ $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$.
This set is used to model the faulty behavior of $\mathfrak{C}$ under fault model *F2*. We first obtain the incorrect traces for C_i under *F2*. Let w_{mxp} be the maximal correct prefix (with respect to φ_i) of $\mathbf{tr}|_{\Sigma_i}$. Let $L^{F2}_{w_{\mathsf{mxp}}}(C_i) \subseteq \Sigma_i^*$ be the language such that $u \in L^{F2}_{w_{\mathsf{mxp}}}(C_i)$ iff $u = w_{\mathsf{mxp}} \cdot v$ for some $v \in \Sigma_i^*$ such that $\mathsf{len}(u) = \mathsf{len}(\mathbf{tr}|_{\Sigma_i})$ and $u|_{\mathsf{out}(\Sigma_i)} = \mathbf{tr}|_{\mathsf{out}(\Sigma_i)}$. Thus, to obtain $L^{F2}_{w_{\mathsf{mxp}}}(C_i)$, we cement the maximal correct local prefix w_{mxp}, and for the positions after that we keep the *same output* as in $\mathbf{tr}|_{\Sigma_i}$ and we allow for all possible inputs. The set $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$ is defined to be:

$$\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i) = \mathsf{Prefs}\left(L^{F2}_{w_{\mathsf{mxp}}}(C_i)\right) \cup \mathsf{Repair}_{\mathbf{tr}}(C_i).$$

Observe that since $L^{F2}_{w_{\mathsf{mxp}}}(C_i)$ contains $\mathbf{tr}|_{\Sigma_i}$, we have $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ to be a subset of $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$.[5]

▶ **Example 5** (Counterfactual Sets). Consider the error trace **tr** from Example 4 and component C_3.
- $\mathsf{Repair}_{\mathbf{tr}}(C_3)$ consists of all traces in φ_3 that agree with $\mathbf{tr}|_{\Sigma_3}$ up to and including position 2 (recall that 3 was the first position at which φ_3 was violated).
- $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_3)$ extends $\mathsf{Repair}_{\mathbf{tr}}(C_3)$ with all prefixes of $\mathbf{tr}|_{\Sigma_3}$.
- $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_3)$ extends $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_3)$ by including all traces in $w \in \Sigma_3^*$ such that: (i) $\mathsf{len}(w) = 4$; and (ii) $\mathbf{tr}|_{\Sigma_3}[1..2]$ is a substring of w; and (iii) w agrees with $\mathbf{tr}|_{\Sigma_3}$ on the variables $\mathsf{out}(X_3)$. ◀

Many of the traces in these local tracesets are infeasible due to interaction with other components. These infeasibilities will be taken care of in the construction of global counterfactual tracesets explained later in this subsection.

In addition to the two counterfactual tracesets above, we have the most expansive tracesets:
(a) φ_i, which is a superset of $\mathsf{Repair}_{\mathbf{tr}}(C_i)$,
(b) $\mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_i}) \cup \varphi_i$, a superset of $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$,
(c) $\mathsf{Prefs}\left(L^{F2}(C_i)\right) \cup \varphi_i$, a superset of $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$.

Notation. For a set $\mathfrak{D} = \{D_1, \dots, D_m\}$ of components we denote $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{D}) = \mathsf{Repair}_{\mathbf{tr}}(D_1) \parallel \cdots \parallel \mathsf{Repair}_{\mathbf{tr}}(D_m)$, and similarly for the functions $\mathsf{Feasible}^{F1}_{\mathbf{tr}}$ and $\mathsf{Feasible}^{F2}_{\mathbf{tr}}$.

Global Counterfactual Tracesets. The global counterfactual tracesets of a system with respect to the component collection $\mathfrak{C}$ are obtained by composing appropriately chosen local counterfactual tracesets, for components both in $\mathfrak{C}$ *and* in $\overline{\mathfrak{C}}$. That is, for each component C_i, we pick a counterfactual traceset T_i, *e.g.*, $T_i = \mathsf{Repair}_{\mathbf{tr}}(C_i)$, or $T_i = \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$, or $T_i = \mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$. The global counterfactual traceset is then $T_1 \parallel \dots \parallel T_n$. Local traces from T_i which become infeasible due to component interactions get automatically eliminated by the language composition definition. In the next section we show what are the appropriate local counterfactual tracesets that need to be chosen; and how global counterfactual sets can be used for various kinds of causality inference.

3.2 Causality Analysis with Counterfactuals

Causality analysis uses counterfactual sets for reasoning about the following two scenarios:
1. **Fault Mitigation Capability:** Would the *correct behavior* of the component set $\mathfrak{C}$ be enough to mitigate the faults of all components (*including those of components that are not in* $\mathfrak{C}$), by ensuring that the global property φ holds?
2. **Fault Manifestation:** Is the observed *faulty behavior* of the component set $\mathfrak{C}$ enough to manifest a global fault (*i.e.*, does it lead to global behaviors that violate φ), even if the components in $\overline{\mathfrak{C}}$ were to behave correctly?

If the answer to the first question above is affirmative, we classify the component set $\mathfrak{C}$ as *fault mitigation-capable*. If the answer to the second question is affirmative, we classify $\mathfrak{C}$ as *fault manifesting*[6]. (In [8, 7], a fault mitigation-capable set is known as a *necessary cause*; and a set which manifests faults is known as a *sufficient cause*.) Here, we use the more reasoning-mechanism explicit names, and try avoid referring to these sets as causes, to keep the trace analysis separate from the philosophical aspects of causality.[7] In Subsection 3.3 we analyze fault mitigation-capable component sets. Fault manifestation analysis is presented in Subsection 3.4.

Remark. Before we formally define the sets, note that correcting an individual component does not always make things better with respect to the global requirement φ, i.e., two wrongs can make a right.

---

[5] Note that in case the observed behavior $\mathbf{tr}|_{\Sigma_i}$ satisfies the local specification φ_i, we have $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$ and $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ both to be equal to $\mathsf{Repair}_{\mathbf{tr}}(C_i)$.

[6] $\mathfrak{C}$ can contain both faulty, and non-faulty components. It follows from our definitions in the following sections that if $\mathfrak{C}$ does not contain *any* faulty components, then $\mathfrak{C}$ is neither fault mitigation-capable, nor fault manifesting.

[7] Readers who are more comfortable with causality terminology can regard "fault mitigation-capable set" as an alias for "necessary cause"; and "fault manifesting set" as an alias for "sufficient cause".

3.3 Causality Analysis: Fault Mitigation

Fault Mitigation-Capable Sets. Intuitively, a component set $\mathfrak{C}$ is fault mitigation-capable if it can, were it to behave correctly, mask the faults of $\overline{\mathfrak{C}}$ in the observed trace **tr** with respect to φ by ensuring that *every* trace in the counterfactual traceset belongs to φ. Here we present the definitions of two possible such sets that arise from two natural choices of counterfactual tracesets.

⋆ **MitigCbl-1**. Component set $\mathfrak{C}$ is fault mitigation-capable if

$$\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Feasible}^{F1}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \subseteq \varphi \tag{1}$$

Thus, we correct the behavior of the components in $\mathfrak{C}$; and take the incorrect *together* with the correct behaviors of components in $\overline{\mathfrak{C}}$, and ask if all the resultant traces are in φ. An obvious question is why the correct behaviors of $\overline{\mathfrak{C}}$ need to be taken – since $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C})$ contains only correct behaviors of $\mathfrak{C}$, composing these correct behaviors with correct behaviors from $\overline{\mathfrak{C}}$ would automatically result in the satisfaction of φ. The reason for this is the following subtlety. Let $\mathfrak{C} = \{C_1\}$ and $\overline{\mathfrak{C}} = \{C_2, C_3\}$. Suppose all components are faulty in the observed trace. If we correct C_1, then the situation can arise where $\mathsf{Repair}_{\mathbf{tr}}(C_1) \parallel \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_2}) \parallel \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_3})$ is the empty set due to inconsistencies between $\mathsf{Repair}_{\mathbf{tr}}(C_1)$ and $\mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_3})$ (and thus $\mathsf{Repair}_{\mathbf{tr}}(C_1) \parallel \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_2}) \parallel \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_3}) \subseteq \varphi$ vacuously), but $\mathsf{Repair}_{\mathbf{tr}}(C_1) \parallel \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_2}) \parallel \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_3)$ is not empty; and moreover is not a subset of φ. That is, including correct behaviors of some components in $\overline{\mathfrak{C}}$ can help us in finding out that correcting the behaviors of the components in $\mathfrak{C}$ does not suffice to ensure satisfaction of the global property.

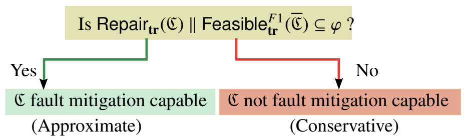

Figure 3 Fault mitigation capability analysis under F1.

Approximation introduced by the analysis. As mentioned previously, the counterfactual analysis procedure can only rely on a single observed trace **tr** and the expected behavior component specifications. It assumes that all global traces resulting from composing local projections of **tr** and correct local traces are *possible executions* of the system. Under this assumption, a *conservative* result is one that relies on the existence of an execution (with certain properties) *in this set* (the set being $\mathsf{Feasible}^{F1}_{\mathbf{tr}}()$).

In particular, the answer "No" in Figure 3, that is, when Equation 1 is *not* satisfied is conservative. A negative answer is given when one of the following two cases arises:

- After correcting the components in $\mathfrak{C}$ the violation of φ will remain under the original *observed* faulty behaviors of the components in $\overline{\mathfrak{C}}$. That is, we have $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\right) \not\subseteq \varphi$.
- After correcting the components in $\mathfrak{C}$ the violation of φ will remain in the case when *some components in* $\overline{\mathfrak{C}}$ *are corrected*. That is, when we have that $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\right) \subseteq \varphi$ but it holds that $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Feasible}^{F1}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \not\subseteq \varphi$.

In both cases there exists a global counterfactual trace in the set of possible executions discussed above that violates φ, and thus Equation 1 is conservative when it gives a "No" answer.

Since Equation 1 is based on the fault model *F1* (see Subsection 3.1), it is based on the assumption that in case the repair of $\mathfrak{C}$ changes the input to the faulty components in $\overline{\mathfrak{C}}$, these components will react correctly (that is, satisfying their local specifications) to the new input. As this assumption is not always guaranteed (as mentioned before, there is no mechanism to predict what happens when we change inputs to faulty components), the "Yes" answer in Figure 3 is *approximate*. That is, in the case when the actual components do not satisfy the fault model *F1* a positive answer need not imply that correcting the components in $\mathfrak{C}$ will result only in executions that satisfy φ (this may or may not be the case, since changing the input to $\overline{\mathfrak{C}}$ may lead to new faulty behaviors).

Our next definition of fault mitigation-capable sets is based on the fault model *F2* from Subsection 3.1 and includes into consideration additional counterfactual behaviors thus allowing for a finer analysis in the cases when Equation 1 is satisfied.

⋆ **MitigCbl-2**. Component set $\mathfrak{C}$ is fault mitigation-capable if

$$\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Feasible}^{F2}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \subseteq \varphi \tag{2}$$

This corresponds to the fault model $F2$ from Subsection 3.1.

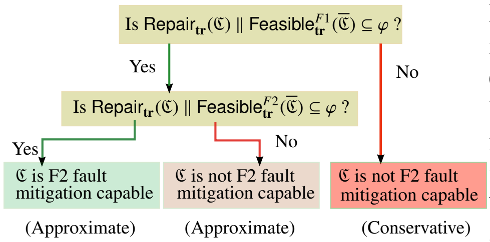

Figure 4 Fault mitigation capability analysis under F2.

We again correct the behavior of the components in $\mathfrak{C}$. For $\overline{\mathfrak{C}}$ components which violate their local specifications in **tr**, we take the observed incorrect *sequence of outputs* (after the local maximal correct prefixes), plus the correct behaviors. Thus, in this analysis, we assume that for a faulty component in $\overline{\mathfrak{C}}$, after the local maximal correct prefix, this component can output the same output sequence as before, even if its input changes (or it can behave correctly on the changed input). For the components that behave correctly in the observed trace **tr** we consider only sets of correct behaviors. A discussion of the approximations mentioned in the figure can be found in the appendix.

▶ **Example 6** (Fault Mitigation Capability)**.** Let us consider again the system and the error trace from Example 4. Using the analysis above, we conclude that under each of the fault models $F1$ and $F2$:

- $\{C_1\}$ is not fault mitigation capable. This is obvious, since the component C_1 behaves correctly in the observed trace **tr**.
- $\{C_2\}$ is fault mitigation capable. In the observed trace **tr**, C_2 violates its safety requirement at positions greater or equal to 2, and the violation at position 4 results in a violation of the global specification φ. Composing the set $\mathsf{Repair}_{\mathbf{tr}}(C_2)$ with $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_1) = \mathsf{Repair}_{\mathbf{tr}}(C_1)$ and $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_3)$ yields traces in φ, since by correcting C_2 we also eliminate the observed violation of φ_3 by C_3 (recall that in **tr** the first violation of φ_2 occurs at position 2 and the first violation of φ_3 at 3).
  In this example, even under the fault model $F2$, where $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_3)$ also includes local traces where the output of C_3 remains as in **tr**, the violation of φ_3 and φ gets ruled out.
- $\{C_3\}$ is fault mitigation capable. In the observed error trace **tr** C_2's consumption at step 2 exceeds the allocated amount in a way that can be tolerated by C_3 in step 3, where the violation of φ_3 occurs. Thus composing any trace in $\mathsf{Repair}_{\mathbf{tr}}(C_3)$ with traces from $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_1) = \mathsf{Repair}_{\mathbf{tr}}(C_1)$ and $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_2)$ results in a trace on which φ is satisfied (even if φ_2 might still be violated). As the elimination of the global violation does not depend on C_2 reacting to the changed input it receives from C_3, this holds also under the fault model $F2$. ◀

In Example 6 we saw that the set $\{C_2\}$ is fault mitigation capable under both fault models $F1$ and $F2$ because the elimination of the global violation did not depend on C_3 reacting to the changed input from C_2. In the Appendix, we give two additional examples which present situations in which this is not the case, and and demonstrate scenarios in which fault mitigation capability classification depends on the fault mode used.

Notes. (A.) For $\mathfrak{C}$ to be fault mitigation-capable, we require that *all* traces in the counterfactual sets of Equations 1 and 2 be in φ. Thus, we have a universal quantifier over the traces in the counterfactual set. A bigger counterfactual set makes it harder to classify $\mathfrak{C}$ as fault mitigation-capable. As a corollary, if $\mathfrak{C}$ is fault mitigation-capable under Equation 2, it is also so under Equation 1 (recall that $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ is a subset of $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$). If $\mathfrak{C}$ is *not* fault mitigation-capable under Eq. 1, it is also not so under Equation 2.
(B.) The conditions in Equations 1 and 2 allow for the possibility that some components in $\overline{\mathfrak{C}}$ might behave correctly, even if their observed behavior in **tr** was incorrect. This makes it harder to classify $\mathfrak{C}$ as being fault mitigation-capable (we recall that since two wrongs can make a right, correcting a component does not always help). ◻

Application. In *fortification*, we want to know if fixing some of the components under our control would suffice to "absorb" the observed errors of the other components so that the global requirement is satisfied; and this notion is what fault mitigation-sets capture.

3.4 Causality Analysis: Fault Manifestation

In this section we analyze fault manifesting sets; which are the component sets $\mathfrak{C}$ such that their observed faulty behavior alone is sufficient to manifest a violation of the global specification φ. An immediate question for fault manifestation is whether the faulty behavior of $\mathfrak{C}$ is such that *some* resultant behavior is faulty w.r.t. φ. The answer to this question is useful in debugging contexts.

Fault Manifesting Sets. Formally, a component set $\mathfrak{C}$ is fault manifesting if its observed faulty behavior alone is enough to manifest in a global error (with respect to φ) in *some* resultant trace, even if the components in $\overline{\mathfrak{C}}$ were to behave correctly. One natural fault manifesting set is as follows.

⋆ **Manifest-1**. Component set $\mathfrak{C}$ is fault manifesting if

$$\mathsf{Feasible}^{F1}_{\mathbf{tr}}(\mathfrak{C}) \ \| \ \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \not\subseteq \varphi \tag{3}$$

The resulting classification analysis is depicted in Figure 5.

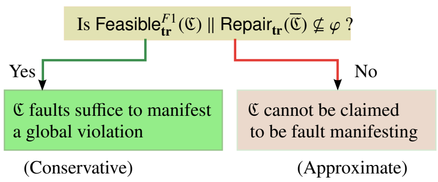

Figure 5 Fault manifestation analysis under F1 (Manifest-1).

In case of an answer "Yes" in Figure 5, we have that there exists a scenario in which the observed behavior of *some* components in $\mathfrak{C}$ is sufficient to lead to a violation of the global specification φ, assuming that the remaining components in $\mathfrak{C}$ (if any) and the components in $\overline{\mathfrak{C}}$ behave correctly. A discussion of the approximation can be found in the appendix.

Application. In *debugging*, we wish to find the group of components, whose erroneous behaviors may cause a violation of the global specification. Of these, groups $\mathfrak{C}$ whose erroneous behaviors are *sufficient* to manifest a global error are the most urgent ones; unless these are fixed, errors will be manifested. These sets are the Manifest-1 sets.

A Stronger Notion of Fault Manifestation. Now we present a stronger definition of fault manifesting sets, in which the possible counterfactual behaviors of $\mathfrak{C}$ components are restricted to prefixes of the observed behavior, and the requirement for the existence of traces violating φ is stronger.

Strong Fault Manifesting Sets. Intuitively, a component set $\mathfrak{C}$ is strong-fault manifesting if its observed faulty behavior alone is enough to manifest in a global error (with respect to φ) in some resultant trace, whether the components in $\overline{\mathfrak{C}}$ were to behave correctly or incorrectly. Recall that $\overline{\mathfrak{C}} = \{C_{1+n_{\mathfrak{C}}}, \dots, C_n\}$, thus $\overline{\mathfrak{C}}$ has $n_{\overline{\mathfrak{C}}} = n - n_{\mathfrak{C}}$ elements. For each component, consider a function $G^{C_i} : \{0,1\} \mapsto \{\mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{C_i}}\right) \mathsf{Repair}_{\mathbf{tr}}(C_i)\}$ defined by:

$$G^{C_i}(0) = \mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{C_i}}\right); \qquad G^{C_i}(1) = \mathsf{Repair}_{\mathbf{tr}}(C_i).$$

Now consider the natural extension to $\overline{\mathfrak{C}}$, where $G^{\overline{\mathfrak{C}}} : \{0,1\}^{n_{\overline{\mathfrak{C}}}} \mapsto \mathsf{Feasible}^{F1}_{\mathbf{tr}}(\overline{\mathfrak{C}})$ defined by:

$$G^{\overline{\mathfrak{C}}}(b_1, b_2, \dots, b_{n-n_{\mathfrak{C}}}) = G^{C_{1+n_{\mathfrak{C}}}}(b_1) \ \| \ G^{C_{2+n_{\mathfrak{C}}}}(b_2) \ \| \dots G^{C_n}(b_{n-n_{\mathfrak{C}}})$$

That is, the boolean vector $(b_1, b_2, \dots, b_{n-n_{\mathfrak{C}}})$ tells us whether to choose $\mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{C_i}}\right)$, or $\mathsf{Repair}_{\mathbf{tr}}(C_i)$ for each component of $\overline{\mathfrak{C}}$ in the composition.

⋆ **Manifest-Strong**. Set $\mathfrak{C}$ is strongly-fault manifesting if

$$\underset{\text{we have}}{\forall (b_1, .., b_{n-n_{\mathfrak{C}}}) \in \{0,1\}^{n-n_{\mathfrak{C}}}} \ \mathsf{Prefs}(\mathbf{tr}|_{\Sigma_{\mathfrak{C}}}) \ \| \ G^{\overline{\mathfrak{C}}}(b_1, b_2, .., b_{n-n_{\mathfrak{C}}}) \not\subseteq \varphi \tag{4}$$

That is, for each component $C_i \in \overline{\mathfrak{C}}$, no matter whether we consider only the observed behavior $\mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{C_i}}\right)$, or the corrected behaviors $\mathsf{Repair}_{\mathbf{tr}}(C_i)$, there will be some resultant trace $\mathbf{tr}'$ in composition with the observed behavior $\mathsf{Prefs}(\mathbf{tr}|_{\Sigma_{\mathfrak{C}}})$ of $\mathfrak{C}$ such that this trace $\mathbf{tr}'$ will violate φ. This means that the observed faulty behavior of $\mathfrak{C}$ is sufficient to manifest in a global error in some trace, no matter which components of $\overline{\mathfrak{C}}$ are repaired or kept as they are.

Equation 4 enables us to make a bullet-proof argument that $\mathfrak{C}$ is for sure to blame for the φ violation. The defining criterion ensures that *no matter which subset of* $\overline{\mathfrak{C}}$ *components were to be corrected*, some resulting trace in composition with the *observed* $\mathfrak{C}$ behavior would have resulted in a φ violation.

4 Fault Models & Language-Theoretic Algorithms

4.1 Analysis of Heterogeneous Fault Models

An important distinguishing feature of our language theoretic framework for counterfactual analysis is its modularity. This modularity yields a powerful reasoning technique that cleanly generalizes existing causality notions, *e.g.*, of [8, 7] to more expressive cases. Previous work required the same fault model for all components. Our approach allows us to drop this requirement by just changing the individual counterfactual sets: since the set of counterfactual traces is constructed locally for each component, we do not have to assume that they follow the same fault model.

Formally, the generalization is done as follows. A *fault-model profile* for a system $\mathcal{S} = \{C_1, \ldots, C_n\}$ is a tuple $\hat{f} = (f_i)_{i=1}^n$ of functions where each $f_i : \Sigma^* \to 2^{\Sigma_i^*}$ maps system traces to a set of local traces for C_i such that for every trace $\mathbf{tr}$, we have $\mathbf{tr}|_{\Sigma_i} \in f_i(\mathbf{tr})$, and $f_i(\mathbf{tr})$ is prefix closed. Intuitively, f_i describes the fault model for component C_i, and given an observed (error) trace $\mathbf{tr} \in \Sigma^*$, the set $f_i(\mathbf{tr})$ is the set of possible local counterfactual traces for C_i (which includes $\mathbf{tr}|_{\Sigma_i}$ since it was observed). In this generalized setting, the sets $\mathsf{Feasible}^{F1}_{\mathbf{tr}}$ and $\mathsf{Feasible}^{F2}_{\mathbf{tr}}$ define two specific functions: $f_i'(\mathbf{tr}) = \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ and $f_i''(\mathbf{tr}) = \mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$. Let

$$\mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}\left(\overline{\mathfrak{C}}\right) = f_k(\mathbf{tr}) \parallel f_{k+1}(\mathbf{tr}) \parallel \cdots \parallel f_n(\mathbf{tr}),$$

where $\overline{\mathfrak{C}} = \{C_k, C_{k+1}, \ldots, C_n\}$ (the set $\mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}(\mathfrak{C})$ is defined similarly) . Using the counterfactual set $\mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}$, we give a general definition of mitigation capability based causality, generalizing Equations 1 and 2, as follows: the component set $\mathfrak{C}$ is fault mitigation-capable under the fault-model profile $\hat{f}$ if

$$\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}\left(\overline{\mathfrak{C}}\right) \subseteq \varphi \tag{5}$$

Similarly, the general definition of fault manifestation based causality as follows: component set $\mathfrak{C}$ is fault manifesting under $\hat{f}$ if

$$\mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \not\subseteq \varphi \tag{6}$$

Employing heterogeneous fault models leads to improved precision of the causality analysis, easily incorporating designer knowledge about the behaviour of components and available simulation data.

Bridging the Gap to Structural-Model based Causality. The seminal work of Halpern and Perl [12] investigated a notion of causality in non-reactive settings which was based on *structural equations* between variables which specified which variables affect which others. The fault model profile $\hat{f}$, and its utilization in causality definitions 5 and 6 bridges the gap between structural-model based causality and causality notions in a reactive setting as follows. A component C_i has an associated variable dependency given by structural equations which specify which variables affect which others (possibly in the future), *e.g.* a change in input variable x at a time-step will lead to a change in output variable y at some point in the future; but another output variable z will remain unaffected by the change in x — this means that y depends on x, and z does not depend on x. This variable dependency can be utilized in the fault model profile f_i for C_i: the set $f_i(\mathbf{tr})$ will only contain strings which satisfy the structural variable dependencies mentioned above. Of course, a change in variable x may lead to a change in y in the future, and if y is an input to some other component C_j, this may lead to a change in some other variable u, *and this change may flow back to C_i in effect changing z.* Thus, we have two manners in which changes in variable values propagate: (i) locally inside a component (perhaps through states), and (ii) in an inter-component fashion in the reactive setting. A fault-model profile based on structural equations captures the first kind of variable change effects. Language composition *automatically* accounts for the second kind of variable change flow for counterfactual reasoning in a modular fashion. Thus, our causality framework using fault-model profiles lays down the theoretical foundations for connecting the work in structural-model based non-reactive causality, to causality in a reactive setting.

4.2 Algorithm Complexity using Language-Theoretic Analysis

We now analyze the time complexity of determining causality based on the language theoretic framework of Section 3. We discuss the language theoretic operations employed, and give bounds for the case where the components are given as finite state automata. For more expressive models, the time bounds correspond to time bounds of analogous language operations.

For an alphabet Σ and a word w, let

- $\mathsf{Cone}_\Sigma(w) = \{w\} \cdot \Sigma^*$, *i.e*, the word w followed by all possible strings in Σ^*; and
- for $\Sigma(X) = \Sigma'(X') \parallel \Xi(X'')$, for some alphabets $\Sigma'(X')$ and $\Xi(X'')$, and for w_p a prefix of w, let
  $\mathsf{AlterRest}_\Xi(w, w_p, \Sigma) = \{w_p \cdot u \mid u \in \Sigma^{|w|-|w_p|} \text{ and } (w_p \cdot u)|_\Xi = w|_\Xi\}$.

The set $\mathsf{AlterRest}_\Xi(w, w_p, \Sigma)$ contains words of length $|w|$ obtained from w by keeping the first $|w_p|$ letters unchanged, and then changing all letters *not* in Ξ to all possible values (this corresponds to changing valuations of variables in $X' \setminus X''$ after w_p).

The counterfactual sets from Section 3.1 can be defined using these languages and basic operations on languages as on the right. Specific algorithms for the case of finite automata to obtain the basic sets on the right are as follows.

$$\begin{aligned}
\mathsf{Repair}_{\mathbf{tr}}(C_i) &= \varphi_i \cap \mathsf{Cone}_{\Sigma_i}(\mathsf{maxcp}(\mathbf{tr})|_{\Sigma_i}),\\
\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i) &= \mathsf{Prefs}\,(\mathbf{tr}_i|_{\Sigma_i}) \cup \mathsf{Repair}_{\mathbf{tr}}(C_i),\\
\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i) &= \begin{array}{r}\mathsf{Prefs}\left(\mathsf{AlterRest}_{\mathsf{out}(\Sigma_i)}\left(\mathbf{tr}|_{\Sigma_i},\, w_{\mathsf{mxp}}, \Sigma_i\right)\right)\\ \cup\, \mathsf{Repair}_{\mathbf{tr}}(C_i),\end{array}
\end{aligned}$$

where where w_{mxp} is the maximal correct prefix (with respect to φ_i) of $\mathbf{tr}|_{\Sigma_i}$.

Recall that a non-deterministic finite automaton (NFA) over an alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \rho, Q_f)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is the input alphabet, $\rho \subseteq Q \times \Sigma \times Q$ is a transition relation, and $Q_f \subseteq Q$ is a set of accepting states. A *deterministic* automaton (DFA) is one where for any $q \in Q$ and $\sigma \in \Sigma$, there is at most one q' such that $(q, \sigma, q') \in \rho$. We denote $\mathcal{L}(\mathcal{A})$ as the language of words in Σ^* accepted by $\mathcal{A}$. Define $|\mathcal{A}| = |Q| + |\rho|$ (thus $|\mathcal{A}| \leq |Q| \times |\Sigma|$). Let the local and global specifications $\varphi_1, .., \varphi_n$ and φ be given as DFAs or NFAs $\mathcal{A}_1, .., \mathcal{A}_n$ and $\mathcal{A}$ respectively. Note that since the specifications are prefix closed, we can assume that all reachable states are final states [14].

The various entities in the previous equations are obtained as follows.

- The string w_{mxp} can be obtained from $\mathbf{tr}$ and φ_i in time $O(|\mathbf{tr}| \cdot |Q_i|^2)$ by running the automaton $\mathcal{A}_i$ on $\mathbf{tr}|_{\Sigma_i}$ (if $\mathcal{A}_i$ is a DFA, this can be done in $O(|\mathbf{tr}|)$ time). Similarly, string $\mathsf{maxcp}(\mathbf{tr})$ can be obtained in $O(|\mathbf{tr}| \cdot \sum_{i=1}^n |Q_i|^2)$ time ($O(n \cdot |\mathbf{tr}|)$ in the DFA case).
- A DFA $\mathcal{D}_i$ with $|\mathbf{tr}_i|_{\Sigma_i}|$ states (and size $|\mathbf{tr}_i|_{\Sigma_i}| + |\Sigma_i|$) can be constructed such that $\mathcal{L}(\mathcal{D}_i) = \mathsf{Cone}_{\Sigma_i}(\mathbf{tr}_i|_{\Sigma_i})$.
- We can construct a DFA $\mathcal{D}'_i$ with $|\mathbf{tr}_i|_{\Sigma_i}|$ states and size such that $\mathcal{L}(\mathcal{D}'_i) = \mathsf{Prefs}(\mathbf{tr}_i|_{\Sigma_i})$ using the standard prefix construction.
- We can construct a DFA for $\mathsf{AlterRest}_{\mathsf{out}(\Sigma_i)}\left(\mathbf{tr}|_{\Sigma_i},\, w_{\mathsf{mxp}}, \Sigma_i\right)$ with $|\mathbf{tr}_i|_{\Sigma_i}|$ states (and size $|\mathbf{tr}_i|_{\Sigma_i}| \cdot |\Sigma_i|$).
  It can be modified to accept $\mathsf{Prefs}\left(\mathsf{AlterRest}_{\mathsf{out}(\Sigma_i)}\left(\mathbf{tr}|_{\Sigma_i},\, w_{\mathsf{mxp}}, \Sigma_i\right)\right)$ by making all states final.

Union and intersection are standard operations on NFAs/DFAs. Thus, the sets $\mathsf{Repair}_{\mathbf{tr}}(C_i)$, and $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ and $\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$ can all be obtained in polynomial time and represented as NFAs of size polynomial in the sizes of $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Consider a fault-model profile $\hat{f} = (f_i)_{i=1}^n$ such that $f_i(\mathbf{tr}) = \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ or $f_i(\mathbf{tr}) = \mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$. Recall Equations 5 and 6. The equations involve taking the parallel composition of the $f_i(\mathbf{tr})$ sets. As we just showed, each $f_i(\mathbf{tr})$ set can be represented as the language of a NFA (or DFA) of polynomial size. The parallel composition of the $f_i(\mathbf{tr})$ sets can be obtained by taking the parallel composition of the corresponding automata using the product construction (in polynomial time). Finally, the equations involve making language inclusion checks, which involve checking $\mathcal{L}(\mathcal{B}_1) \parallel \cdots \parallel \mathcal{L}(\mathcal{B}_n) \subseteq \mathcal{L}(\mathcal{A})$, where B_i are (polynomial sized) automata derived as above for either the $\mathsf{Repair}_{\mathbf{tr}}$ or $\mathsf{Feasible}^{F1}_{\mathbf{tr}}$ or $\mathsf{Feasible}^{F2}_{\mathbf{tr}}$ sets. This check can be performed by checking $\mathcal{L}(\mathcal{B}_1) \parallel \cdots \parallel \mathcal{L}(\mathcal{B}_n) \subseteq \mathcal{L}(\widetilde{\mathcal{A}})$ where $\widetilde{\mathcal{A}}$ is the deterministic automaton for $\mathcal{A}$. Putting everything together, we get the following.

▶ **Theorem 7.** *Let $\hat{f} = (f_i)_{i=1}^n$ be a fault-model profile such that $f_i(\mathbf{tr}) = \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ or $f_i(\mathbf{tr}) =$*

$\mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$. *Let the local and global specifications $\varphi_1, \ldots, \varphi_n$ and φ (such that $\varphi_1 \parallel \ldots \parallel \varphi_n \subseteq \varphi$) be given as NFAs (or DFAs) $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and $\mathcal{A}$ respectively. Given an observed trace $\mathbf{tr} \notin \varphi$, for the fault profile $\hat{f}$, a component set $\mathfrak{C}$ can be determined to be: fault mitigation capable (Equation 5), or fault manifesting (Equation 6) in time (i) polynomial in the sizes of $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and $\mathbf{tr}$; and (ii) exponential in $|\mathcal{A}|$ in case $\mathcal{A}$ is an NFA, or polynomial in $|\mathcal{A}|$ in case $\mathcal{A}$ is a DFA.* ◀

Determining whether $\mathfrak{C}$ is strongly-fault manifesting according to Equation (4) is harder as there is an additional for all quantifier, and thus is exponential time even in the case $\mathcal{A}$ is a DFA.

A careful analysis shows that for fault-model profiles with $f_i(\mathbf{tr}) = \mathsf{Feasible}^{F1}_{\mathbf{tr}}(C_i)$ or $f_i(\mathbf{tr}) = \mathsf{Feasible}^{F2}_{\mathbf{tr}}(C_i)$, the fault mitigation capability (Equation 5) check can be performed only with word-sets that are of length at most $|\mathbf{tr}|$. As a result, the fault mitigation capability problem is in co-NP. A similar argument shows fault manifestation determination to be in NP. This also allows us to check language inclusion, without determinizing $\mathcal{A}$, in time polynomial in $|\mathcal{A}|$ and exponential in $|\mathbf{tr}|$.

For general regular language fault-model profiles when $\varphi_1, \ldots, \varphi_n, \varphi$ are given as NFAs language inclusion for two nondeterministic automata can be encoded in each of the causality analysis questions. Thus, the causality analysis problem is PSPACE-complete (membership in PSPACE follows from PSPACE membership of NFA language inclusion).

5 Discussion

We discuss some of the related work in the Appendix. Our work overcomes the shortcomings of existing work in the reactive setting as follows. As evidenced by works [8, 7, 22, 6, 9], the main challenge in causality analysis for concurrent systems is in the construction of counterfactual sets. Definitions that do not account for the effect of repairing some components on the behavior of others [8] result in vacuous causes. This implies the need for definitions that take component interactions into account. However, the ones existing in the literature are overly complicated, which hinders understanding and ensuring their correctness. For example, in [7] the cone of influence of a set of components is defined by means of the fixpoint of a function g, where g itself is defined by a formula spanning several lines containing six quantified variables, with five of them being trace temporal-position variables (cf. Definition 4 in [7]). Similarly, in [6, 9] the definition of unaffected prefixes is described as a fixpoint computation involving a sequence of four connected definitions. In addition, the minimal unaffected prefixes in [6] are not always composable. More precisely, the expression for tr_i^* in Definition 9 of [6] allows for having different extensions w for different components j, thus resulting in local unaffected prefixes that are *not* consistently extendable taken together, resulting in an empty set of global counterfactual traces. This problem stems from the mixed-up treatment of local and global traces.

Our work demonstrates that we can leverage the theory of language composition to obtain modular and transparent definitions for counterfactual sets. This decomposition allows us to focus on local alternative scenarios when comparing different causality notions; our framework based on language composition takes care of global-level reactive reasoning in a uniform way across the different causal sets. This machinery allows us to (1) define and easily reason about *new* causal sets (eg strongly-fault manifesting sets in Equation (4) that had not been considered before) based on the application need, (2) seamlessly incorporate *heterogeneous fault models* in causal sets (existing work assumes a common fault model across all components), (3) compare different causality notions; and (4) automatically obtain *algorithms* for computing different causal sets based on standard language theoretic operations.

References

1 Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. Symbolic causality checking using bounded model checking. In *SPIN*, volume 9232 of *Lecture Notes in Computer Science*, pages 203–221. Springer, 2015.

2 I. Beer, S. Ben-David, H. Chockler, A. Orni, and R.J. Trefler. Explaining counterexamples using causality. In *CAV 09*, LNCS 5643, pages 94–108. Springer, 2009.

3 F.D. Busnelli. Causation. In *Principles of European Tort Law*, pages 43–63. Springer, 2005.

4 H. Chockler, O. Grumberg, and A. Yadgar. Efficient automatic STE refinement using responsibility. In *TACAS 08*, LNCS 4963, pages 233–248. Springer, 2008.

5 H. Chockler, J.Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Logic*, 9(3):20:1–20:26, 2008.

6 G. Gössler and L. Aştefănoaei. Blaming in component-based real-time systems. In *EMSOFT 14*, pages 7:1–7:10. ACM, 2014.

7 G. Gößler and D.L. Métayer. A general trace-based framework of logical causality. In *FACS 13*, LNCS 8348, pages 157–173. Springer, 2013.

8 G. Gößler, D.L. Métayer, and J.-B. Raclet. Causality analysis in contract violation. In *RV 10*, LNCS 6418, pages 270–284. Springer, 2010.

9 Gregor Gößler and Daniel Le Métayer. A general framework for blaming in component-based systems. *Sci. Comput. Program.*, 113:223–235, 2015. `doi:10.1016/j.scico.2015.06.010`.

10 Gregor Gößler and Jean-Bernard Stefani. Fault ascription in concurrent systems. In Pierre Ganty and Michele Loreti, editors, *Trustworthy Global Computing - 10th International Symposium, TGC 2015, Madrid, Spain, August 31 - September 1, 2015 Revised Selected Papers*, volume 9533 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2015. `doi:10.1007/978-3-319-28766-9_6`.

11 S. Halle. Causality in message-based contract violations: A temporal logic "whodunit". pages 171–180, 2011.

12 J.Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. part I: Causes. *The British journal for the philosophy of science*, 56(4):843–887, 2005.

13 D. Hume. *An Enquiry concerning Human Understanding*. 1748.

14 J-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theor. Comput. Sci.*, 410(47-49):5010–5021, 2009.

15 Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. From probabilistic counterexamples via causality to fault trees. In *SAFECOMP*, volume 6894 of *LNCS*, 2011.

16 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. URL: `http://doi.acm.org/10.1145/359545.359563`, `doi:10.1145/359545.359563`.

17 Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. In *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013.

18 Florian Leitner-Fischer and Stefan Leue. Probabilistic fault tree synthesis using causality computation. *IJCCBS*, 4(2):119–143, 2013. URL: `http://dx.doi.org/10.1504/IJCCBS.2013.056492`, `doi:10.1504/IJCCBS.2013.056492`.

19 D. Lewis. Void and object. In *Causation and Counterfactuals*, pages 277–290. MIT Press, 2004.

20 Chao Wang, Zijiang Yang, Franjo Ivancic, and Aarti Gupta. Whodunit? causal analysis for counterexamples. In *ATVA 2006*, volume 4218 of *LNCS*, pages 82–95, 2006.

21 S. Wang, A. Ayoub, R. Ivanov, O. Sokolsky, and I. Lee. Contract-based blame assignment by trace analysis. In *HiCoNS 13*, pages 117–126. ACM, 2013.

22 S. Wang, A. Ayoub, B. Kim, G. Gößler, O. Sokolsky, and I. Lee. A causality analysis framework for component-based real-time systems. In *RV 13*, LNCS 8174, pages 285–303. Springer, 2013.

23 Shaohui Wang, Yoann Geoffroy, Gregor Gößler, Oleg Sokolsky, and Insup Lee. A hybrid approach to causality analysis. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2015. `doi:10.1007/978-3-319-23820-3_16`.

Appendix

A Examples: Fault Mitigation Analysis

▶ **Example 8** (Fault Mitigation Capability under *F2*). Consider the system of Example 4. Suppose we restrict the correct behaviors of C_3 further by adding the following condition $\varphi_3^\dagger$ to φ_3: for every $1 \le j < \mathsf{len}(w)$, we have $w_{[j]}(x_a^{3,1})+w_{[j]}(x_a^{3,2})+w_{[j+1]}(x_d^3) \ge 29$. That is, component C_3 tries to optimize the resource allocation so that the combined availability (usage by C_3) is at least 29 in each step. All components are faulty in the trace on the left. In the trace, both C_1, and C_2 exceed their allocations by 4 units each at each step (after step 3). Component C_3 is faulty from step 3 on, as it should have decreased its consumption by 5 ($= 16 - 11$) units from the depletion at step 2, but instead it only decreases by 2 units (from 7 to 5), and does not decrease at all in step 4.

j, φ	$1, \varphi$	$2, \varphi$	$3, \neg\varphi$	$4, \neg\varphi$
local specs	$\varphi_1, \varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \varphi_3$	$\neg\varphi_1, \neg\varphi_2, \neg\varphi_3$	$\neg\varphi_1, \neg\varphi_2, \neg\varphi_3$
$x_a^{3,1}$	11	12	12	12
$x_a^{3,2}$	11	12	12	12
$x_d^{1,3}$	0	8	16	16
$x_d^{2,3}$	0	16	16	16
x_d^3	0	7	5	5

Analysis under fault model *F1* (Equation 1) classifies $\{C_3\}$ as fault mitigation-capable, *i.e.* able to absorb the faults of both C_1 and C_2. Intuitively, this seems false: if both C_1 and C_2 are consuming 16 units as observed, there is nothing C_3 can do. The reason why we get this false answer is due to the shortcoming of fault model *F1*. Consider any fix of C_3. Let this fixed word be $\sigma'_1, \sigma'_2, \sigma'_3, \sigma'_4$. A fix requires that component C_3 reduce its resource depletion to 2 at step 3, as C_2 had exceeded its allocation by 5 units in the previous step ($16 - 11$) Because of the new optimized resource allocation requirement introduced at the beginning of the example, this reduction of $\sigma'_3(x_d^3)$ to 2 implies that $28 \ge \sigma'_2(x_a^{3,1}) + \sigma'_2(x_a^{3,2}) \ge 27$, thus, the values of at least one of $x_a^{3,1}, x_a^{3,2}$ must change from the observed 12 units in the trace at step 2 to something higher. However, *F1* assumes that whenever inputs change to a faulty component, the outputs must change to correct ones, thus *F1* implies that the combined consumption of C_1, C_2 will reduce to 28 or lower (from the observed 32). Thus, *F1* forces us to assume that if C_3 gives a *higher* allocation to C_1, C_2, it will result in a *lower* consumption by C_1, C_2 as their inputs have changed.

An analysis under *F2* on the other hand assumes that C_1, C_2 will keep consuming 16 units from step 3 onwards, and thus will *not* classify $\{C_3\}$ as fault mitigation-capable. ◀

▶ **Example 9** (Fault Mitigation Capability under *F1*). Consider the system of Example 4 (with the additional requirement $\varphi_3^\dagger$ added to φ_3). In addition, let component C_2 have another output variable $x_r^{2,3}$ denoting the requested amount for the next to next time step; and let this variable be read by C_3. Let us add to φ_2 the requirement $\varphi_2^\dagger$ that the value requested $x_r^{2,3}$ at step j is at least as much as the depleted amount $x_d^{2,3}$ by C_2 in step $j+2$. Finally, let us add another requirement $\varphi_3^\ddagger$ to φ_3 saying that the value of the allocation to C_2, *i.e.* $x_a^{3,2}$ is equal to its requested resource $x_r^{2,3}$ in the previous time step. Thus, C_3 trusts the estimate of C_2.

Consider the observed trace $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ on the left in which C_2, C_3 are faulty, C_1 is not. φ_3 is violated at step two because $\sigma_1(x_a^{3,1}) + \sigma_1(x_a^{3,2}) + \sigma_2(x_d^3) < 29$ violating the added optimization criteria $\varphi_3^\dagger$. This is a benign violation. The global specification stays violated at step 4 even though the combined utilization is less than 31 as the bound was violated in the previous step (we require φ to be prefix-closed).

j, φ	$1, \varphi$	$2, \varphi$	$3, \neg\varphi$	$4, \neg\varphi$
local specs	$\varphi_1, \varphi_2, \varphi_3$	$\varphi_1, \neg\varphi_2, \neg\varphi_3$	$\varphi_1, \neg\varphi_2, \neg\varphi_3$	$\varphi_1, \neg\varphi_2, \neg\varphi_3$
$x_a^{3,1}$	13	25	7	7
$x_a^{3,2}$	13	5	23	23
$x_d^{1,3}$	0	10	9	7
$x_d^{2,3}$	0	16	23	23
$x_r^{2,3}$	5	6	8	6
x_d^3	0	2	0	0

Intuitively, looking at the example, the problem with C_3 is that it blindly trusted C_2's estimates and did not increase allocation to C_2 by the end of step 2 (and concomitantly decrease allocation to C_1). While the added variable $x_r^{2,3}$ is available to C_3, it is of no use as C_2 is giving incorrect estimates of its future resource usage. Observe that if C_1 is working perfectly (and depleting the resource way less than C_2). It appears that if C_2 had given correct values in its estimates $x_r^{2,3}$, then C_3 could have allocated correctly (telling C_1 to decrease its usage), and avoided a global violation, thus we expect $\{C_2\}$ to be fault mitigation-capable.

We claim $\{C_2\}$ is fault mitigation-capable under *F1*, but not under *F2*. The reason is that under *F2*, even if the inputs to C_3 change (in particular the estimates $x_r^{2,3}$ by component C_2), the behavior of C_3 will be assumed to be the same as observed, with the same old output values of $x_a^{3,1}, x_a^{3,2}$. However, under *F1*, with the changed inputs, the behavior of C_3 is assumed to be different, and correct; and will correctly set the changed $x_a^{3,1}, x_a^{3,2}$ values (in the process telling C_1 to reduce its usage). Thus in this example, *F1* is the fault model which gives the intuitively correct answer to fault mitigation capability, compared to Example 8 where *F2* gave the intuitively correct answer. This example, and Example 8 show that which fault model to choose depends on the application dynamics ◀

B Example: Fault Manifestation Analysis

▶ **Example 10** (Fault Manifestation)**.** We consider the system and the error trace from Example 4 and determine that $\{C_2, C_3\}$ is fault manifesting. The set $\mathsf{Prefs}(\mathbf{tr}|_{\Sigma_2 \| \Sigma_3})$ contains the projection of $\mathbf{tr}$ on the alphabet $\Sigma_2 \parallel \Sigma_3$. Since C_1 behaves correctly in $\mathbf{tr}$, the observed error trace is actually an element of the set of counterfactual traces which implies that the set $\{C_2, C_3\}$ is fault manifesting. ◀

C Casual Sets: Approximations Resulting from Fault Model Assumptions

C.1 MitigCbl-1

Quantifying the confidence in the approximation. In the case of a "Yes" answer to the decision question in Figure 3, we can quantify our confidence the given answer as follows.

– If the set $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \{\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\}$ is non-empty, then it means that the behavior of $\mathfrak{C}$ can be corrected in such a way that the original faulty behavior of $\overline{\mathfrak{C}}$ is compatible with the new behavior of $\mathfrak{C}$. According to Equation 1 we have that every trace in this set satisfies the global specification φ. Thus, in this case, the answer "Yes" is actually *exact*, meaning that repairing $\mathfrak{C}$ suffices to absorb the remaining faults of $\overline{\mathfrak{C}}$ that were observed.

– If, on the other hand,
  - $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \{\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\}$ is empty and
  - $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\right)$ is not empty,

  then we use the maximal prefix $\mathbf{w}_{\mathsf{mxp}} \in \mathsf{Prefs}\left(\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}\right)$ such that $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C}) \parallel \{w_{\mathsf{mxp}}\}$ is non-empty to determine our confidence in the result. The prefix $\mathbf{w}_{\mathsf{mxp}}$ is the longest prefix of $\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}$ which is consistent with the corrected behaviors $\mathsf{Repair}_{\mathbf{tr}}(\mathfrak{C})$ of $\mathfrak{C}$. The confidence in the "Yes" answer is bigger the longer $\mathbf{w}_{\mathsf{mxp}}$ is (after occurrences of errors in $\overline{\mathfrak{C}}$); if *e.g.* $\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}$ is only a one-step extension of $\mathbf{w}_{\mathsf{mxp}}$, then Equation 1 tells us that correcting the components $\mathfrak{C}$ will be enough to absorb the faults of $\overline{\mathfrak{C}}$ of $\mathbf{tr}|_{\Sigma_{\overline{\mathfrak{C}}}}$, except possibly at the last step for which we do not know.

C.2 MitigCbl-2

Approximation introduced by the analysis. Although here we consider a larger set of counterfactual traces than in the previous definition, a positive answer to the question whether Equation 2 is satisfied is again approximate. The reason is that the components in $\overline{\mathfrak{C}}$ are not guaranteed to satisfy the fault model *F2*, and, again, unconsidered behaviors (on which φ might not hold) might arise in the actual system after repairing $\mathfrak{C}$.

Unlike in the previous definition, now an *approximation occurs even in the case of a negative answer*, as depicted in Figure 4. The reason is that we might classify $\mathfrak{C}$ as not being fault mitigation-capable under Equation 2 based on a trace which is not a possible trace of the actual system, while it can happen that in the actual system repairing $\mathfrak{C}$ leads to a completely new output behavior of $\overline{\mathfrak{C}}$ that results in traces on which φ holds. Thus, the "No" answer to the check of Equation 2 is approximate.

Quantifying the confidence in the approximation. The confidence of a "Yes" answer in this case is determined in a way similar to before. The more interesting case is that of a "No" answer.

Recall that $\mathsf{Feasible}^{F1}_{\mathbf{tr}}(\overline{\mathfrak{C}}) \subseteq \mathsf{Feasible}^{F2}_{\mathbf{tr}}(\overline{\mathfrak{C}})$, and thus if $\mathfrak{C}$ is not fault mitigation capable under Equation 1, then it is not mitigation capable under Equation 2 either. Therefore, as shown in Figure 4, it only makes sense to consider Equation 2 when the answer to the check of Equation 1 is "Yes". This allows us to use the confidence in the "Yes" answer for Equation 1 determined as before to determine the confidence in the "No" answer for Equation 2.

- If the "Yes" answer for Equation 1 was exact, then we should have low confidence in the "No" answer for Equation 2.
- Otherwise, the higher the confidence in the "Yes" answer for Equation 1 is, the lower our confidence in the "No" answer for Equation 2 should be.

C.3 Manifest-1

Approximation introduced by the analysis. The "Yes" decision in Figure 5 is conservative, since the witness trace that violates φ is composed of observed and correct behaviors of the components. A "No" answer, however, is bound to be approximate, in the actual system correcting the components in $\overline{C}$ may result in new executions that are not considered as part if our set of counterfactuals.

Quantifying confidence in the approximation. In case of a "No" decision in Figure 5, we quantify our confidence in the "No" answer analogously to the analysis of the "Yes" decision for $\mathsf{MitigCbl\text{-}1}$.

- If the set $\{\mathbf{tr}|_{\Sigma_{\mathfrak{C}}}\} \parallel \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}})$ is non-empty, then it means that some correct behavior of $\overline{\mathfrak{C}}$ is compatible with the observed behavior of $\mathfrak{C}$, and all the traces in the composition satisfy φ. Thus, if $\overline{\mathfrak{C}}$ were to behave correctly, the violation of φ would disappear, irrespective of the observed faulty behavior of $\mathfrak{C}$. This indicates that the confidence in the "No" decision in this case is high.
- If, on the other hand,
  - $\{\mathbf{tr}|_{\Sigma_{\mathfrak{C}}}\} \parallel \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}})$ is empty and
  - $\mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_{\mathfrak{C}}}) \parallel \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}})$ is not empty,

  then we consider the maximal prefix $\mathbf{w}_{\mathsf{mxp}} \in \mathsf{Prefs}\,(\mathbf{tr}|_{\Sigma_{\mathfrak{C}}})$ such that $\{\mathbf{w}_{\mathsf{mxp}}\} \parallel \mathsf{Repair}_{\mathbf{tr}}(\overline{\mathfrak{C}})$ is non-empty, exactly like in the "Yes" decision for $\mathsf{MitigCbl\text{-}1}$, and quantify our confidence according the length of $\mathbf{w}_{\mathsf{mxp}}$ (after occurrences of errors in $\mathfrak{C}$).

D Relationships Between Causal Sets

In this subsection we establish relations between some of the causal sets defined in Subsections 3.3 and 3.4. First, we compare the different sets with respect to the strength of the corresponding causality notions. Recall again that in the literature on causality, fault mitigation-capable sets are called necessary causes, and that fault manifesting sets are called sufficient causes.

▶ Proposition 1 (Fault Mitigation-Capable Sets). If set $\mathfrak{C}$ is fault mitigation-capable under Equation (2), then it is also fault mitigation-capable under Equation (1).

The following proposition formalizes the relationship between fault mitigation-capable sets and fault manifesting sets.

▶ Proposition 2 (Interrelationships).
1. If $\mathfrak{C}$ is a fault mitigation-capable set according to Equation (1), then $\overline{\mathfrak{C}}$ is *not* a fault manifesting set according to Equation (3).
2. If $\mathfrak{C}$ is a fault manifesting set according to Equation (3), then $\overline{\mathfrak{C}}$ is not fault mitigation-capable under Equation (1).

Note that the set of all components $\{C_1, \ldots, C_n\}$ is trivially both a fault mitigation-capable set, and also a fault manifesting set. However, in applications we are interested in such sets that are *minimal* with respect to the subset relation, and identifying such sets is a non-trivial task. The definitions we studied here enjoy the monotonicity properties stated in the proposition below.

▶ Proposition 3 (Monotonicity). **1.** If a set $\mathfrak{C}$ satisfies Equation (1), then any superset $\mathfrak{D} \supseteq \mathfrak{C}$ also satisfies Equation (1).
2. If $\mathfrak{C}$ satisfies Equation (3) any superset $\mathfrak{D} \supseteq \mathfrak{C}$ also satisfies it.

E Computational Complexity of Causality Problems

Let $\mathcal{A}, \mathcal{B}$ be NFAs over an alphabet Ξ, such that $\mathcal{L}(A)$ and $\mathcal{L}(B)$ are prefix closed. Note that for NFAs where all states are final, language inclusion has the same complexity as for general NFA, i.e., it is PSPACE-complete [14]. We will now show how to reduce the question $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$ to the fault mitigation and fault manifestation questions, thus proving their PSPACE-hardness.

We define two components C_1 and C_2 as follows. Assume w.l.o.g. that we have letters $a_1, b_1, a_2, b_2 \notin \Xi$. Let x_1 and x_2 be two variables with domains $O_1 = \{a_1, b_1\}$ and $O_2 = \{a_2, b_2\}$ respectively, and x_Ξ be a variable with domain Ξ.

We define the component $C_1 = (X_1, \mathsf{inp}(X_1), \mathsf{out}(X_1), \Sigma_1, \varphi_1)$, where $X_1 = \{x_\Xi, x_1, x_2\}$, $\mathsf{inp}(X_1) = \{x_2\}$, $\mathsf{out}(X_1) = \{x_1, x_\Xi\}$ and

$$\begin{aligned}\varphi_1 \quad = \quad & (\mathcal{L}(\mathcal{A}) \parallel (\{a_1\} \cdot O_1^*) \parallel (\{a_2\} \cdot O_2^*)) \cup \\ & (\mathcal{L}(\mathcal{B}) \parallel (\{a_1\} \cdot O_1^*) \parallel (\{b_2\} \cdot O_2^*)) \cup \{\epsilon\}.\end{aligned}$$

Intuitively, if the first value of x_2 is a_2, then C_1 has to output strings from $\mathcal{L}(\mathcal{A})$, and if this value is b_2, C_1 has to output strings from $\mathcal{L}(\mathcal{B})$.

For the other component, let $C_2 = (X_2, \mathsf{inp}(X_2), \mathsf{out}(X_2), \Sigma_2, \varphi_2)$, where $X_2 = \{x_2\}$, $\mathsf{inp}(X_2) = \emptyset$, $\mathsf{out}(X_2) = \{x_2\}$ and $\varphi_2 = \{a_2\} \cdot O_2^* \cup \{\epsilon\}$.

Finally, we define the global specification as

$\varphi = \mathcal{L}(\mathcal{A}) \parallel (\{a_1\} \cdot O_1^*) \parallel O_2^* \cup \{\epsilon\}$.

Thus, we clearly have that $\varphi_1 \parallel \varphi_2 \subseteq \varphi$, regardless of $\mathcal{L}(\mathcal{B})$.

Fix the fault profile $\hat{f}$: $f_1(\mathbf{tr}) = \Xi^* \parallel O_1^* \parallel O_2^*$ and $f_2(\mathbf{tr}) = O_2^*$.

Consider the trace $\mathbf{tr}$ of length 1, where for the first letter we have $x_1 = b_1$, $x_2 = b_2$ and $x_\Xi = \xi$ for some letter $\xi \in \Xi$. Clearly $\mathbf{tr} \notin \varphi_1, \varphi_2, \varphi$; $\mathsf{Repair}_{\mathbf{tr}}(C_1) = \varphi_1$, and $\mathsf{CFac}^{\hat{f}}_{\mathbf{tr}}(C_2) = \{b_2\} \cdot O_2^* \cup \varphi_2$. Thus, the set $\mathfrak{C}_1 = \{C_1\}$ is fault mitigation-capable w.r.t. $\hat{f}$ iff

$$\begin{gathered}\varphi_1 \parallel (\{b_2\} \cdot O_2^*) \subseteq \mathcal{L}(\mathcal{A}) \parallel (\{a_1\} \cdot O_1^*) \parallel O_2^* \\ \text{iff} \\ \mathcal{L}(\mathcal{B}) \parallel (\{a_1\} \cdot O_1^*) \parallel (\{b_2\} \cdot O_2^*) \subseteq \mathcal{L}(\mathcal{A}) \parallel (\{a_1\} \cdot O_1^*) \parallel O_2^* \\ \text{iff} \\ \mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A}).\end{gathered}$$

Similarly, the set $\{C_2\}$ is *not* fault manifestation-capable iff $\varphi_1 \parallel (\{b_2\} \cdot O_2^*) \subseteq \mathcal{L}(\mathcal{A}) \parallel (\{a_1\} \cdot O_1^*) \parallel O_2^*$ iff $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$.

Given the automata $\mathcal{A}$ and $\mathcal{B}$, in time polynomial in their size we can construct NFAs for φ_1 and φ by extending their alphabet by $O_1 \parallel O_2$. The NFAs for φ_2, $f_1(\cdot)$ and $f_2(\cdot)$ are of constant size. Thus, we can reduce $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$ to checking fault mitigation/ fault manifestation for a suitable fault model.

F Additional Related Work

In this section we discuss some of the most prominent definitions of causes from the literature and highlight connections to our definitions of causal sets from Equations 1 through 6.

Causality for Structural Equations. The paper [12] gives a definition of "actual cause" in a setting of structural equations over Boolean variables, where the structural equations describe the causal dependencies between these variables. Actual causality is based on counterfactual as well as on contingency dependency: only contingencies that "do not interfere with active causal processes" are considered. The major difference from our (and related work) on concurrent reactive systems is that [12] assumes that a full model of the system is known. In contrast, our work deals with (concurrent) systems for which we are only given a set of correct behaviors, and *one single incorrect behavior*. In addition, most of the work of [12] deals with acyclic variable dependencies, while concurrent reactive systems are usually cyclic.

Recently, the structural equation model approach by Halpern and Pearl was extended to reason about models of event-based concurrent systems [17, 1] in which temporal logic formulas are used to describe the causal process of a violation. Similarly to the original approach, these methods do not face the challenge of constructing counterfactual executions for black-box systems, as they work with a given system model. In fact, they integrate causality checking in the model-checking process.

Necessary Causes. The work of [7], building upon their earlier work [8], presents a causality definition that takes into account the effect of changing the behavior of one component on others in a reactive setting. In [6] the reasoning based on necessary causes (fault mitigation-capable sets, in our terms) is extended to the real-time setting; here the definition of counterfactual traces requires that for each component the difference between the local counterfactual traces and the observed local trace is minimal. This difference is minimized locally for each individual component, which, a careful analysis shows, can construct local traces that are not composable. The analysis in [22] can also yield inconsistent counterfactual traces, leading to erroneous results. The fault ascription analysis in [10] is parameterized by a counterfactual operator, but only a single concrete definition based on the idea of closeness of counterfactuals to the observed behaviours is provided. In contrast, we propose and discuss several fault models.

Sufficient causes. The notion of sufficient causes has not been explored much in the literature. Definitions based on universal quantification have been studied in [8] and in [7]. Our definition of fault manifesting sets presented in Subsection 3.4, on the other hand, is an existential one and requires that some resultant behavior is faulty.

Contributory causes. [21] studies a conservative version of the so called *contributory causes* (where one is interested in the ratio between the number of traces that satisfy φ after replacing all components in $\mathfrak{C}$ with correct ones and the total number of such alternative traces). The analysis is focused on a special case where a set of components is determined to be a cause (called culprit) iff this ratio is 1, which is essentially the same as the classical definition of necessary cause. The key difference between [21] and previous work is in the way the set of counterfactual traces is defined: it assumes that faulty components in $\overline{\mathfrak{C}}$ will produce the same output letter sequence as in the observed trace **tr** *even if their inputs change*, while components that have not violated their local properties on **tr** will react correctly to changed input. The set of counterfactual traces obtained using our fault model *F2* is larger, an therefore it is more difficult to characterize a set as a cause based on *F2* than based on [21].

Root causes. The notion of *root causes* for contract violations in message-based systems has been studied in [11]. The problem studied there is different from ours, as it does not capture mitigation. The root cause is determined by the shortest non-compliant prefix of the error trace, regardless of whether this violation could have been mitigated by another component. As a consequence, there is exactly one root cause, while there can be multiple mitigation-capable sets, which will not be discovered by the algorithm in [11].

Counterexample analysis. In [20] the authors perform causality analysis of counterexample traces, relying on a single error trace and the program source, without considering counterfactual traces. The key difference to our work is that they do not reason about concurrent reactive systems, but work on the level of variables in a single program, over which they compute weakest preconditions. Other works [15, 18] derive fault trees from probabilistic counterexamples employing counterfactuals in the flavour of the notion by Pearl and Halpern. Since they also work in the single-component setting, they do not face the challenges we address.