Causality Analysis for Concurrent Reactive Systems

Rayna Dimitrova  Rupak Majumdar  Vinayak S. Prabhu
MPI-SWS
{rayna,rupak,vinayak}@mpi-sws.org

ABSTRACT
We present a comprehensive language-theoretic and modular framework for causality analysis in the setting of concurrent reactive systems. Our framework allows us to express a vast number of causality notions studied in the areas of artificial intelligence and formal methods, as well as define and analyze new ones of interest. Furthermore, our formalization provides means for reasoning about the relationships between individual notions which have mostly been considered independently in prior work; and hence judge appropriateness of the different definitions for various application domains. In particular, we consider causality analysis for debugging, error resilience, and for liability resolution in concurrent reactive systems. Finally, we exhibit the utility of our modular framework by demonstrating its ability to seamlessly combine heterogeneous component fault models in an automata based causality determination approach.

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 [4] (that is, which parts of the system under scrutiny are relevant for the satisfaction of a given correctness criterion), to explain counterexamples [1] (identify points in a counterexample trace that are relevant for the failure of a temporal specification), to fault tree construction [14], and to automatically refine system abstractions [3]. In the field of artificial intelligence, causality-based explanation finding has applications in natural language processing, automated medical diagnosis, vision processing, and planning, among others. Finally, resolving liabilities in a legal setting often relies on establishing the causal relations between potential causes and the occurred damage [2].

Causality definitions based on counterfactuals, which are alternative scenarios where the suspected cause \(e_1\) of \(e_2\) did not happen, date back to [10] and have been extensively studied in philosophy (cf. [15]). In computer science, the most prominent and widely used definition of causality is that of [9]. Finding the right notion of causality among the plethora of existing definitions is a difficult task and often depends on the intended application: Halpern and Pearl write in their seminal paper [9] “... 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 the 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 the implementation of components is 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 [7, 6, 17, 5], especially because such systems are at the core of many safety-critical applications. The goal is to identify a subset of the components, which have violated their specifications, that are actually responsible for the violation of the system-level property. This requires integrating the temporal order of events [13] 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 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.

In this work, we present a language theoretic causality framework for concurrent reactive systems incorporating diverse counterfactual trace sets. A cause for a violation is a set of components. Our analysis reasons about two types of scenarios to determine if component set \(\mathcal{C}\) is a cause (for a given observed system fault):

- **Fault Mitigation Capability** analysis asks whether the correct behavior of the component set \(\mathcal{C}\) is enough to mitigate the faults of all components (including those of components not in \(\mathcal{C}\)), by ensuring that the required system property holds.
- **Fault Manifestation** analysis asks whether the observed faulty
behavior of the component set $C$ is enough to manifest a global fault (i.e., a system-property violating global behavior), even if the components not in $C$ were to behave correctly. These two classifications parallel the classifications of [7, 6] 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 $C_1$, 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 $\varphi$ (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 $\varphi$, 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 $\varphi_1$ of $C_1$ require that the resource allocation respects a given safety margin, namely, that the combined allocation by $C_1$ to all the components should not be more than $M - 1$ units in any step. Furthermore suppose that $\varphi_2$ 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 $C_1$ should attempt to compensate for that by reducing its own consumption. More concretely, if at time $t$, component $C_1$ indicates that $C_1$ should consume at most $\Delta$ units in the next time instant $t + 1$, and at $t + 1$ component $C_1$ consumes $\Delta + \alpha$ instead, then $C_1$ must decrease its own consumption at time step $t + 2$ (from the consumption at time $t + 1$) by $\alpha$, if possible (or reduce it to 0 if not), in an attempt to prevent a violation of the global property $\varphi$. 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 $C_1$ 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 $\varphi$ that the combined resource consumption be at most 31), where $C_1$ allocates at most 30 units in total to all components:  

<table>
<thead>
<tr>
<th>Step</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

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. $C_1$ 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 reduction. Had $C_1$ 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 approach should report the component set $\{C_1\}$ 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 $\varphi$. 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.
- We next, show how composed counterfactual tracesets can be used to define various notions of causality in a uniform fashion (Equations 1 through 3). Our approach uses basic language theoretic operations to reason about intricate consistency issues: issues which arise when 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). The various causality notions, and their relationships are illustrated with the help of several running examples.
- 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.
- We present an automata-based method for causality analysis in the presence of heterogeneous component-fault models, and determine its algorithmic complexity.
- 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. Furthermore, we compare our causality notions to existing related work on causality.

## 2. PRELIMINARIES

**Languages.** Let $\Sigma$ be a non-empty finite alphabet. A word or a trace $w = \sigma_1, \sigma_2, \ldots, \sigma_m$ over $\Sigma$ is a finite sequence of letters from the alphabet $\Sigma$. We denote by $w[i]$ the $i$-th symbol $\sigma_i$ in the word $w$, and by $w[i:j]$ the substring $\sigma_i, \ldots, \sigma_j$. We denote with $\Sigma^n$ the set of all words over $\Sigma$, and with $\emptyset$ the empty word. A language is a set of words. The concatenation of two words $u, v$ is denoted $u \cdot v$, and similarly for languages. For a word $w$ we denote with $|w|$ 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 $u \preceq v$, if there exists a word $w$ such that $v = u \cdot w$. For a language $L$, the language Prefix($L$) consists of the prefixes of words in $L$. We write $u < v$ when $u$ is a strict prefix of $v$, that is, $u \preceq v$ and $u \neq v$.

Languages are prefix closed, i.e., for any word $w$, $\text{Prefix}(w)$ is a prefix of $w$.

**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 $\Sigma$ 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 $\sigma : X \mapsto \mathbb{R}$. We denote $\Sigma^n(X)$ for $n \geq 1$.

### 2.1 Alphabet and Language Composition

**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 $\Sigma_i(x) = \Sigma_j(x)$ (i.e. common variables have the same domain in each alphabet), we define the composite alphabet $\Sigma_1[X_1] \sqcup \cdots \sqcup \Sigma_n[X_n]$ to be the alphabet $\Sigma(X)$ such that $X = \bigcup_{i=1}^n X_i$, and such that the domain of a variable $x$ is $\Sigma(x) = \Sigma_i(x)$ for $x \in X_i$. Given languages $L_i \subseteq \Sigma_i$ over $\Sigma_i$, for $i = 1, \ldots, n$, we denote the set $\bigcup_{i=1}^n L_i$ over $\Sigma$.

---

1. Prefix-closed languages need not be regular
define the language composition of \( L_1, \ldots, L_n \) to be the language: \( L_1 \cdots L_n = \{ w \in \Sigma^* | w_{[i]} \in L_i \text{ for all } i \} \), over \( \Sigma(X) \).

**Example 2** (Languages and Compositions). We consider a language \( L_1 \) over an alphabet \( \Sigma_1[X_1] \), with \( X_1 = \{ x_1, x_2 \} \) and domain \( \Delta_2(x_1) = \Delta_2(x_2) = \{ 0, 1 \} \); and a language \( L_2 \) over an alphabet \( \Sigma_2[X_2] \) with \( X_2 = \{ x_3, x_4 \} \), also with Boolean domain.

The language \( L_1 \) is defined as:
\[
(x_1; b_1^1, b_2), (x_2; b_3, b_4^5), \ldots, (x_i; b_{i1}^1, b_{i2}^2) | b_1 = b_2 \text{ for all } 1 \leq j \leq m \]
i.e. words where the values of \( x_1 \) and \( x_2 \) are the same.

The language \( L_2 \) is of the similar language of words having equal value \( x_3 \) and \( x_4 \):
\[
(x_3; b_3, b_4^5), (x_4; b_3, b_4^5), \ldots, (x_i; b_{i3}^3, b_{i4}^4) | b_3 = b_4 \text{ for all } 1 \leq j \leq m \]
i.e. words where \( x_3, x_4 \) have the same value at each step.

**Component Model.** A component specification is a tuple \( C = (X, \text{inp}(X), \text{out}(X), \Sigma, \varphi) \), where
- \( X = \text{inp}(X) \cup \text{out}(X) \) is the set of variables of the component, consisting of the input variables \( \text{inp}(X) \) and the output variables \( \text{out}(X) \) (the sets of input and output variables being disjoint);
- \( \Sigma \) is the alphabet, consisting of all possible values of the variables \( X \);
- \( \varphi \) is a non-empty prefix-closed language over \( \Sigma \). Intuitively, the language \( \varphi \) specifies 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 \( \sigma \). The input alphabet \( \text{inp}(\Sigma) \) of \( C \) consists of the possible values of the input variables \( \text{inp}(X) \), and, similarly, the output alphabet \( \text{out}(\Sigma) \) consists of values of \( \text{out}(X) \).

**Example 3** (Component). Component \( C_1 \) from the example described in the introduction can be modelled as \( C_1 = (X_1, \text{inp}(X_1), \text{out}(X_1), \Sigma_1, \varphi_1) \), where \( X_1 = \{ x_{11}, x_{12} \} \) and \( \text{inp}(X_1) = \{ x_{11}, x_{12} \} \); and \( \text{out}(X_1) = \{ x_{13}, x_{14} \} \); the alphabet \( \Sigma_1 \) consists of the possible values of \( x_{11} \) and \( x_{12} \) ranging over \( \{ 0, M \} \), and \( \varphi_1 \) contains strings \( w \in \Sigma_1 \) such that either \( w \) is the empty string; or (ii) \( w_1(x_{12}^1) = 0 \) and \( w_1(x_{14}) \leq w_1(x_{11}) \) for all \( 2 \leq j + 1 \leq \text{len}(w) \).

Intuitively, the value of \( x_{13} \) specifies the units of resource allocated to \( C_1 \) by \( C_2 \) for the next step, the value of \( x_{14} \) specifies the units of resource depleted by \( C_1 \) in the current step (this number is also given as input to \( C_1 \)). The specification \( \varphi_1 \) ensures that at each step \( C_1 \)'s consumption does not exceed the specified by the value of \( x_{13} \) in the previous trace step.

**Component Compositions, Systems, & Global Specifications.** Given a set of components \( \mathcal{C} = \{ C_1, \ldots, C_n \} \) where each \( C_i = (X_i, \text{inp}(X_i), \text{out}(X_i), \Sigma_i, \varphi_i) \), the composition \( C_1 \cdots C_n \) is defined in case the following two conditions both hold:
1. The sets of output variables are pairwise disjoint, i.e., if \( \text{out}(X_i) \cap \text{out}(X_j) = \emptyset \) for \( i \neq j \); and
2. the composite alphabet \( \Sigma_1 \cdots \Sigma_n \) is the same.

The composition \( C_1 \cdots C_n \) is the component \( (X_1 \cup \cdots \cup X_n, \text{inp}(X_1) \cup \cdots \cup \text{inp}(X_n), \Sigma_1 \cdots \Sigma_n, \varphi_1 \cdots \varphi_n) \) defined as follows:
- \( X_0 = \bigcup_i X_i \) is the set of all variables;
- \( \text{out}(X_0) = \bigcup_i \text{out}(X_i) \), i.e., the set of output variables is the set containing each output variable of every component;
- \( \text{inp}(X_0) = \left( \bigcup_i \text{inp}(X_i) \right) \setminus \text{out}(X_0) \), i.e., the set of input variables contains those input variables which are not output variables of any component in \( \mathcal{C} \).

\(^2\)In naming variables in the examples, we follow the convention that a variable \( x_{i,j}^{\text{out}} \) (i) is common to the components \( C_i, C_{i+1}, \ldots, C_n \); and (ii) is an output variable of component \( C_i \), and an input variable to components \( C_{i+1}, \ldots, C_n \).

\(^3\)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 system specification of $C_j \parallel C_k \parallel C_i$ is defined to be the language $\psi$ containing words $w$ such that for every $j$ the combined depletion is at most $M$. Formally, $\psi$ equals:

$$\{w \in \Sigma^* \mid \text{for all } j, \text{ we have } w_1 [x_j^1(x_j^2 + w_1(x_j^3) + w_1(x_j^4)] \leq M\}$$

We present two sample traces, letting $M = 31$ (the global specification $\psi$ is that at each step, the combined depletion for that step must not exceed 31). The first trace satisfies $\psi$ and is as follows:

<table>
<thead>
<tr>
<th>$j, \psi$</th>
<th>1, $\psi$</th>
<th>2, $\psi$</th>
<th>3, $\psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>local specs</td>
<td>$\phi_1, \phi_2, \phi_3$</td>
<td>$\phi_2, \phi_3, \phi_1$</td>
<td>$\phi_1, \phi_2, \phi_3$</td>
</tr>
<tr>
<td>$x_1^3$ and $x_2^3$</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$x_1^4$</td>
<td>0</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>$x_1^5$</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$x_1^6$</td>
<td>0</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

Note that even though component $C_2$ violates its local spec $\phi_2$ in steps 2 and 3 (as it depletes by 16 units when it was allowed only 10 as specified by $x_2^3$ in steps 1 and 2), the global specification is still satisfied due to $C_1$ reducing its own depletion amount at step 3 from 10 (in the previous step) to 4.

The second trace given below violates $\psi$.

| $j, \psi$ | 1, $\psi$ | 2, $\psi$ | 3, $\psi$ | 4, $\psi$ |
|-----------|------------|------------|------------|
| local specs | $\phi_1, \phi_2, \phi_3$ | $\phi_2, \phi_3, \phi_1$ | $\phi_1, \phi_2, \phi_3$ | $\phi_1, \phi_2, \phi_3$ |
| $x_1^3$ | 10 | 10 | 10 | 10 |
| $x_1^4$ | 10 | 10 | 10 | 10 |
| $x_1^5$ | 0 | 6 | 6 | 8 |
| $x_1^6$ | 0 | 16 | 16 | 16 |
| $x_1^7$ | 0 | 8 | 8 | 8 |

Component $C_1$ violates its local specification $\phi_1$ at step 3 because it should have reduced its consumption (from that in step 2) by $\alpha$ where $\alpha$ is the amount by which the resource depletion by $C_2$ exceeded its allocated 10 units (in this case $\alpha = 6$ units).

3. A FRAMEWORK FOR CAUSALITY

In this section, we fix a system $C_1, \ldots, C_n$, where $C_i = (\Sigma_i, |i|, \Sigma_i)$, out($\Sigma_i$), $\psi_i$; a specification $\psi_i$; and an observed trace $t_r / \psi_i$. We also fix a non-empty collection of components $C \subseteq \{C_1, \ldots, C_n\}$ for causality analysis. Let $C$ be the collection of components not in $C$. Without loss of generality, assume $C = \{C_1, C_2, \ldots, C_m\}$ (thus, $C = \{C_1, \ldots, C_n\}$, and let $\Sigma_C$ denote the composition of the alphabets of the components of $C$.

3.1 Counterfactual Traces & Faulty Behaviors

Counterfactual Traces. Informally, the set of counterfactual traces for a given observed trace $t_r$ consists of traces obtained from $t_r$ 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 $t_r$. Depending on the hypothetical scenario to be reasoned about, the set of counterfactual traces is obtained as (a subset of) the composition of trace sets of appropriately altered individual component behaviors, where the alteration is with respect to the trace $t_r$.

In reactive systems, the behaviors of individual components are intertwined; the behavior of one component affects that of others. 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 traces for $C_i$ that is applicable for all purposes. We present different constructions of counterfactual traces, and indicate the situations in which each is useful. In each

$\text{maxcp}(\text{tr}_{i_k}) = \psi_i,$

$\text{Repair}_{i_k}(C_i) = \psi_i.$

Figure 2: Traceset $\text{Repair}_{i_k}(C_i)$.

of these constructions, the set of counterfactual traces for component $C_i$ whose observed behavior in $t_r$ is incorrect will include some of the correct behaviors of $C_i$ (according to $\psi_i$), 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 $\text{tr}_{i_k}$ and its prefixes. Essentially, this fault model assumes that if the inputs to component $C_i$ change, then the faulty behavior disappears, and is replaced by correct behaviors (according to $\psi_i$) over the new input. Thus we assume that the faulty behavior of $C_i$ was only for the particular input in $t_r_{i_k}$.

**F2. the only incorrect** local traces for component $C_i$ that may be included in counterfactual sets are ones that agree with both: (i) $\text{mxp}_{i_k}$, where $\text{mxp}_i$ is the maximal correct prefix (with respect to $\psi_i$) of $t_r_{i_k}$, and (ii) $\text{tr}_{\text{out}(\Sigma_i)}$. 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 $\text{out}(\Sigma_i)$ output symbols as $t_r_{\text{out}(\Sigma_i)}$. 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 $t_r_{\text{out}(\Sigma_i)}$.

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 $\phi_1, \ldots, \phi_n$ and a single execution trace $t_r$. Thus, there is no mechanism to predict what the output of a component will be when its input changes, without paying the cost of running additional simulations or monitoring system executions. If such additional data is available, 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 a trace $t_r$, denoted $\text{maxcp}(t_r)$ is defined to be the maximal prefix $t_r_{\text{mxp}}$ of $t_r$ that satisfies all local specifications, i.e. (a) $t_r_{\text{mxp}}$ projected onto $\Sigma_i$ is a subset of $\phi_i$ for all $i$; and (b) for every prefix $t_r'$ of $t_r$ such that $t_r_{\text{mxp}}$ is a strict prefix of $t_r'$, there is a $j$ such that $t_r_{i_j} \notin \phi_i$.

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

- **$\text{Repair}_{i_k}(C_i)$** defined as: $\{w \in \psi_i \mid \text{maxcp}(\text{tr}_{i_k}) \subseteq w\}$. That is, we keep the prefix $\text{maxcp}(\text{tr}_{i_k})$ for component $C_i$, and then take all possible correct $C_i$ behavior extensions following $\text{maxcp}(\text{tr}_{i_k})$; i.e. we repair the errors following $\text{maxcp}(\text{tr}_{i_k})$, as well as the effects in $C_i$ of errors in other components after $\text{maxcp}(t_r)$. Observe that $\text{Repair}_{i_k}(C_i)$ is a subset of $\psi_i$. Intuitively, this set captures the set of possible outcomes of $C_i$ after $\text{maxcp}(\text{tr}_{i_k})$, if no error had occurred in any component.

We illustrate this traceset in Figure 2. The first trace is the local trace for component $C_i$, obtained as the projection of the global trace on $\Sigma_i$. The point $\nu$ denotes the place of the first violation of the local property $\psi_i$ by component $C_i$. The point $\nu$ denotes the place of the first violation of some local property $\psi_i$ by component $C_i$, obtained as the projection of the global trace on $\Sigma_i$. The point $\nu$ denotes the place of the first violation of the local property $\psi_i$ by component $C_i$. The point $\nu$ denotes the place of the first violation of some local property $\psi_i$ by
component $C_i$ where $k$ may be different from $i$. Thus, the portion of $\text{tr}_{i_k}$ until $v$ is equal to $\text{maxcp}(\text{tr}_{i_k})$. The set $\text{Repair}_{i}(C)$ is obtained by taking the cone of all correct executions of $C_i$ from the prefix $\text{maxcp}(\text{tr}_{i_k})$.

Observe, as depicted in the Figure, that there might be a strict prefix $\text{tr}_v < \text{tr}$ such that $\text{maxcp}(\text{tr}_{i_k} \bowtie \text{tr}_v)$ and $\text{tr}_{i_k} \not\subseteq \phi_i$, i.e., component $C_i$ might continue to behave correctly in $\text{tr}_{i_k}$ after $\text{maxcp}(\text{tr}_{i_k})$; however the behavior after $\text{maxcp}(\text{tr}_{i_k})$ is considered to be tainted. This is because after $\text{maxcp}(\text{tr}_{i_k})$ 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 $\text{maxcp}(\text{tr}_{i_k})$. Moreover, before $\text{maxcp}(\text{tr}_{i_k})$, no component is in error, and all are behaving according to their specifications; thus, we need not consider alternate traces before $\text{maxcp}(\text{tr}_{i_k})$.

* Feasible$^A_1(C_i)$ is defined as: $\text{Feasible}^A_1(C_i) = \text{Prefs}(\text{tr}_{i_k}) \cup \text{Repair}_{i}(C_i)$. This tracset is obtained by adding all the prefixes of the observed (possibly incorrect) trace $\text{tr}_{i_k}$ to the set $\text{Repair}_{i}(C_i)$. Thus, this tracset 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 $\text{maxcp}(\text{tr})$.

This set models the faulty behavior of $C_i$ under fault model $F_i$.

In Feasible$^A_2(C_i)$, 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 Feasible$^A_2(C_i)$, because (i) although we want Feasible$^A_2(C_i)$ 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.

* Feasible$^A_2(C_i)$ is defined as: $\text{Feasible}^A_2(C_i) = \text{Prefs}(\text{tr}_{i_k} \bowtie \text{maxcp}(\text{tr})) \cup \text{Repair}_{i}(C_i)$. This set is used to model the faulty behavior of $C_i$ under fault model $F_i$. We first obtain the incorrect traces for $C_i$ under $F_i$. Let $w_{\text{map}}$ be the maximal correct prefix (with respect to $\phi_i$) of $\text{tr}_{i_k}$. Let $L_{\text{map}}(C_i) \subseteq \Sigma^*$ be the language such that $u \in L_{\text{map}}(C_i)$ if $u = w_{\text{map}} \cdot v$ for some $v \in \Sigma^*$ such that $\text{len}(u) = \text{len}(\text{tr}_{i_k})$ and $d_{\text{map}}(u) = \text{tr}_{i_k}(u_{\text{map}})$. Thus, to obtain $L_{\text{map}}(C_i)$, we cement the maximal correct prefix locally $w_{\text{map}}$, and for the positions after that we keep the same output as in $\text{tr}_{i_k}$ and we allow for all possible inputs. The set Feasible$^A_2(C_i)$ is defined to be:

$\text{Feasible}^A_2(C_i) = \text{Prefs}(L_{\text{map}}(C_i)) \cup \text{Repair}_{i}(C_i)$.

Observe that since $L_{\text{map}}(C_i)$ contains $\text{tr}_{i_k}$, we have Feasible$^A_2(C_i)$ to be a subset of Feasible$^A_1(C_i)$.

**Example 5 (Counterfactual Sets).** Consider the error trace $\text{tr}$ from Example 4 on component $C_1$. - $\text{Repair}_{i}(C_i)$ consists of all traces in $\phi_i$ that agree with $\text{tr}_{i_k}$ up to and including position 2 (recall that 3 was the first position at which $\phi_i$ was violated).
- Feasible$^A_1(C_i)$ extends $\text{Repair}_{i}(C_i)$ with all prefixes of $\text{tr}_{i_k}$.
- Feasible$^A_2(C_i)$ extends $\text{Repair}_{i}(C_i)$ by including all traces in $w \in \Sigma^*$ such that: (i) $w$ has length 4; and (ii) $\text{tr}_{i_k}[1...2]$ is a substring of $w$; and (iii) $w$ agrees with $\text{tr}_{i_k}$ on the values of the variables out($X$).

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

In addition to the two counterfactual tracsets above, we have the most expansive tracsets:

1. $\phi$, which is a superset of $\text{Repair}_{i}(C_i)$.
2. $\phi \bowtie \text{Feasible}^A_1(C_i)$.
3. $\phi \bowtie \text{Feasible}^A_2(C_i)$.

**Notation.** For a set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of components we denote $\text{Repair}_{i}(\mathcal{T}) = \text{Repair}_{i}(D_1) \parallel \cdots \parallel \text{Repair}_{i}(D_n)$, and similarly for the functions Feasible$^A_1$ and Feasible$^A_2$.

**Global Counterfactual Tracsets.** The global counterfactual tracsets of a system with respect to the component collection $\mathcal{E}$ are obtained by composing appropriately chosen local counterfactual tracsets, for components both in $\mathcal{E}$ and in $\mathcal{\overline{E}}$. That is, for each component $C_i$, we pick a counterfactual tracset $T_i$, e.g., $T_i = \text{Repair}_{i}(C_i)$, or $T_i = \text{Feasible}^A_1(C_i)$, or $T_i = \text{Feasible}^A_2(C_i)$. The global counterfactual tracset is then $T_1 \parallel \cdots \parallel T_n$. 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 tracsets 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 $\mathcal{E}$ be enough to mitigate the faults of all components (including those of components that are not in $\mathcal{E}$), by ensuring that the global property $\psi$ holds?
2. **Fault Manifestation:** Is the observed faulty behavior of the component set $\mathcal{E}$ enough to manifest a global fault (i.e., does it lead to global behaviors that violate $\psi$), even if the components in $\mathcal{E}$ were to behave correctly?

If the answer to the first question above is affirmative, we classify the component set $\mathcal{E}$ as fault mitigation-capable. If the answer to the second question is affirmative, we classify $\mathcal{E}$ as fault manifesting. In [7, 6], 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. In Subsection 3.3 we analyze fault mitigation-capable component sets. Fault manifestation analysis is presented in Subsection 3.4.

**Reminder.** Before we formally define the sets, we remind the reader that correcting an individual component does not always make things better with respect to the global requirement $\psi$, i.e., two wrongs can make a right.

### 3.3 Causality Analysis: Fault Mitigation

**Fault Mitigation-Capable Sets.** Intuitively, a component set $\mathcal{E}$ is fault mitigation-capable if it can, were it to behave correctly, mask the faults of $\mathcal{E}$ in the observed trace $\text{tr}$ with respect to $\psi$ by ensuring that every trace in the counterfactual tracset belongs to $\psi$.

Here we present the definitions of two possible such sets that arise from two natural choices of counterfactual tracsets.

**MitigCbl-1.** Component set $\mathcal{E}$ is fault mitigation-capable if

$$\text{Repair}_{i}(\mathcal{E}) \parallel \text{Feasible}^A_1(\mathcal{E}) \subseteq \psi$$

Thus, we correct the behavior of the components in $\mathcal{E}$; and take the incorrect together with the correct behaviors of components in $\mathcal{E}$.

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

**MitigCbl-2.** For 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".

Note that in case the observed behavior $\text{tr}_{i_k}$ satisfies the local specification $\phi_i$, we have Feasible$^A_1(C_i)$ and Feasible$^A_2(C_i)$ both to be equal to $\text{Repair}_{i}(C_i)$.
of the system. Under this assumption, a

As mentioned previ-

Approximation introduced by the analysis.

In the case of a

when we change inputs to faulty components), the “Yes” answer in

mentioned before, there is no mechanism to predict what happens

will react correctly (that is, satisfying their local specifications) to

changes the input to the faulty components in

changes the input to the faulty components in

thus

but

moreover is not a subset of

That is, including correct behaviors of some components in \( \mathcal{C} \) can help us in finding out that correcting

the behaviors of the components in \( \mathcal{C} \) does not suffice to

ensuring the satisfaction of the global property.

Approximation introduced by the analysis. As mentioned previ-

ously, the counterfactual analysis procedure can only rely on a

single observed trace \( \mathbf{tr} \) and the expected behavior component specifi-

ications. It assumes that all global traces resulting from composing

local projections of \( \mathbf{tr} \) and correct local traces are possible execu-

tions of the system. Under this assumption, a conservative result is

one that relies on the existence of an execution (with certain prop-

erties) in this set (the set being Feasible \( \mathcal{I}(\mathcal{C}) \)).

In particular, the answer “No” in Figure 3, that is, when Equa-

tion 1 is not satisfied is conservative. A negative answer is given when one of the following two cases arises:

– After correcting the components in \( \mathcal{C} \) the violation of \( \varphi \) will

remain under the original observed faulty behaviors of the compo-

nents in \( \mathcal{C} \). That is, we have Repair\(_{\mathcal{C}}\)\( \mathbf{tr} \) \( \in \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \subseteq \varphi \).

– After correcting the components in \( \mathcal{C} \) the violation of \( \varphi \) will

remain in the case when some components in \( \mathcal{C} \) are correct. That

is, when we have that Repair\(_{\mathcal{C}}\)\( \mathbf{tr} \) \( \in \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \subseteq \varphi \) but it

holds that Repair\(_{\mathcal{C}}\)\( \mathbf{tr} \) \( \in \) Feasible\(_{\mathcal{I}}\)\( \mathcal{C} \) \( \notin \varphi \).

In both cases there exists a global counterfactual trace in the set

of possible executions discussed above that violates \( \varphi \), and thus

Equation 1 is conservative when it gives a “No” answer.

Since Equation 1 is based on the fault model F1 (see Subsec-

tion 3.1), it is based on the assumption that in case the repair of \( \mathcal{C} \)

changes the input to the faulty components in \( \mathcal{C} \), 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 compo-

nents do not satisfy the fault model F1 a positive answer need

not imply that correcting the components in \( \mathcal{C} \) will result only in

executions that satisfy \( \varphi \) (this may or may not be the case, since

changing the input to \( \mathcal{C} \) may lead to new faulty behaviors).

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 Repair\(_{\mathcal{C}}\)\( \mathbf{tr} \) \( \in \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) is non-empty, then it means

that the behavior of \( \mathcal{C} \) can be corrected in such a way that the

original faulty behavior of \( \mathcal{C} \) is compatible with the new behavior

and ask if all the resultant traces are in \( \varphi \). An obvious question is

why the correct behaviors of \( \mathcal{C} \) need to be taken – since Repair\(_{\mathcal{C}}\)\( \mathbf{tr} \) contains only correct behaviors of \( \mathcal{C} \), composing these correct be-

haviors with correct behaviors from \( \mathcal{C} \) would automatically result in

the satisfaction of \( \varphi \). The reason for this is the following subletty.

Let \( \mathcal{C} = \{ C_f \} \) and \( \mathcal{C} = \{ C_2, C_3 \} \). Suppose all components are faulty in

the observed trace. If we correct \( C_1 \), then the situation can arise

where Repair\(_{\mathcal{C}}\)\( C_1 \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) is the empty set
due to inconsistencies between Repair\(_{\mathcal{C}}\)\( C_1 \) and Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) (and

thus Repair\(_{\mathcal{C}}\)\( C_1 \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \subseteq \varphi \) vacuously),

but Repair\(_{\mathcal{C}}\)\( C_1 \) \( \cup \) Prefs\((\mathbf{tr}^*_{\mathcal{C}}) \) \( \cap \) Feasible\(_{\mathcal{I}}\)\( C_1 \) is not empty; and more-

over is not a subset of \( \varphi \). That is, including correct behaviors

of some components in \( \mathcal{C} \) can help us in finding out that correcting

the behaviors of the components in \( \mathcal{C} \) does not suffice to

ensuring the satisfaction of the global property.

Quantifying the confidence in the approximation. In the case of a

“When” answer in this case is determined in a way similar to

before. The more interesting case is that of a “No” answer.
Recall that Feasible$^{\phi_1}_{\mathbb{F}}(\mathbb{C}) \subseteq$ Feasible$^{\phi_2}_{\mathbb{F}}(\mathbb{C})$, and thus if $\mathbb{C}$ 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, the lower our confidence in the “No” answer for Equation 2 should be.

**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 $\phi$. Composing the set $\text{Repair}_{\phi}(C_2)$ with Feasible$^{\phi}_{\mathbb{F}}(C_2) = \text{Repair}_{\phi}(C_1)$ and Feasible$^{\phi}_{\mathbb{F}}(C_1)$ yields traces in $\phi$, since by correcting $C_1$ we also eliminate the observed violation of $\phi$ by $C_2$ (recall that in $tr$ the first violation of $\phi$ occurs at position 2 and the first violation of $\phi_i$ at 3).

In this example, even under the fault model $F_2$, where Feasible$^{\phi}_{\mathbb{F}}(C_2)$ also includes local traces where the output of $C_2$ remains as in $tr$, the violation of $\phi$ and $\phi_i$ gets ruled out.

- $(C_1)$ 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 $C_1$ in step 3, where the violation of $\phi_i$ occurs. Thus composing any trace in $\text{Repair}_{\phi}(C_1)$ with traces from Feasible$^{\phi}_{\mathbb{F}}(C_1) = \text{Repair}_{\phi}(C_1)$ and Feasible$^{\phi}_{\mathbb{F}}(C_2)$ results in a trace on which $\phi$ is satisfied (even if $\phi_i$ 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 $F_2$.

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 $C_2$ reacting to the changed input from $C_3$. In our next two examples we consider situations in which this is not the case, and demonstrate scenarios in which fault mitigation capability classification depends on the fault mode used.

**Example 7 (Fault Mitigation Capability under $F2$).** Consider the system of Example 4. Suppose we restrict the correct behaviors of $C_1$ further by adding the following condition $\phi_1$ to $\phi_i$:

$$ \phi_1 \equiv \exists j < \ln(\mathbb{w}), \text{ there exist } w_1(x_1^{j+1}) + w_1(x_2^{j+1}) + w_1(x_3^{j+1}) \geq 29. $$

That is, component $C_1$ tries to optimize the resource allocation so that the combined utilization (usage by $C_3$) is at least 29 in each step. All components are faulty in the following trace:

\[
\begin{array}{cccccccc}
\mathbf{j} & \mathbf{\varphi} & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{\sigma}_1 & \mathbf{\sigma}_2 & \mathbf{\sigma}_3 \\
1 & 1 & 11 & 12 & 12 & 12 & 12 & 12 \\
2 & 2 & 11 & 12 & 12 & 12 & 12 & 12 \\
3 & 3 & 0 & 8 & 16 & 16 & 16 & 16 \\
4 & 4 & 0 & 7 & 5 & 5 & 5 & 5 \\
\end{array}
\]

In the above trace, both $C_1$ and $C_2$ exceed their allocations by 4 units each at the end of 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.

Analysis under fault model $F1$ (Equation 1) classifies $(C_2)$ 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 $\sigma_1', \sigma_2', \sigma_3'$. 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 $\sigma_1'(x_2^3) + \sigma_2'(x_3^3)$ by 2 implies that $\geq 27 + \sigma_3'(x_3^3) + \sigma_2'(x_3^3)$.

Thus, the values of at least one of $x_2^3, x_3^3$ 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 $(C_2)$ as fault mitigation-capable.

**Example 8 (Fault Mitigation Capability under $F1$).** Consider the system of Example 4 (with the additional requirement $\phi_1$ added to $\phi_i$).

In addition, let component $C_2$ have another output variable $x_3'$ denoting the requested amount for the next to next time step; and let this variable be read by $C_3$. Let us add to $\phi_1$ the requirement $\phi_1^*$ that the value requested $x_3'$ at step $j$ is at least as much as the depleted amount $x_2^j$ by $C_2$ in step $j + 2$. Finally, let us add another requirement $\phi_3^*$ to $\phi_1$, saying that the value of the allocation to $C_2$, i.e. $x_2^j$, is equal to its requested resource $x_3'$ in the previous time step. Thus, $C_3$ trusts the estimate of $C_2$. Consider the following observed trace $\sigma_1, \sigma_2, \sigma_3, \sigma_4$, in which $C_2$ are faulty, $C_1$ is not.

\[
\begin{array}{cccccccc}
\mathbf{j} & \mathbf{\varphi} & \mathbf{\sigma}_1 & \mathbf{\sigma}_2 & \mathbf{\sigma}_3 & \mathbf{\sigma}_4 \\
1 & 1 & 13 & 25 & 7 & 7 \\
2 & 2 & 13 & 5 & 23 & 23 \\
3 & 3 & 0 & 10 & 9 & 7 \\
4 & 4 & 0 & 16 & 23 & 23 \\
5 & 5 & 5 & 6 & 8 & 6 \\
6 & 6 & 5 & 6 & 8 & 6 \\
\end{array}
\]

Thus, $C_2$ is violated at step two because $\sigma_1(x_2^3) + \sigma_2(x_3^3) + \sigma_3(x_3^3) < 29$ violating the added optimization criteria $\phi_1^*$. 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 $\varphi$ to be prefix-closed).

Intuitively, looking at the example, the problem with $C_1$ is that it blindly trusts $C_2$’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 $x_3'$ 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 $x_3^3$, then $C_1$ 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 $x_3'$ by component $C_2$), the behavior of $C_1$ will be assumed to be the same as observed, with the same old output values of $x_2^3, x_3^3$. However, under $F1$, with the changed inputs, the behavior of $C_1$ is assumed to be different, and correct; and will correctly set the changed $x_2^3, x_3^3$ 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.
In this section we analyze fault manifesting sets; which are the groups of components, whose erroneous behaviors may cause a violation of the global specification. Of these, groups ε 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. Fixing the fault mitigation-capable component sets ε is also important, as this provides insurance against unfair negative reputation impact due to specification violations by other components which are composed with ε out in the field.

Example 9 (Fault Manifestation). We consider the system and the error trace from Example 4 and determine that ∑ε is fault manifesting. The set Pref(ε|repairε(ε)) contains the projection of ε on the alphabet ∑ε ∪ ∑F. Since Cε behaves correctly in tr, the observed error trace is actually an element of the set of counterfactual traces which implies that the set (Cε, Cε) is fault manifesting.

3.5 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 ε 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 ε is a fault mitigation-capable set according to Equation (1), then ε is not a fault manifesting set according to Equation (3).
2. If ε is a fault manifesting set according to Equation (3), then ε is not fault mitigation-capable under Equation (1).

Note that the set of all components (C1, . . . , CN) 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 ε satisfies Equation (1), then any superset ε ⊆ ∑ε also satisfies Equation (1).
2. If ε satisfies Equation (3) any superset ε ⊇ ∑ε also satisfies it.
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 $S = \{C_1, \ldots, C_n\}$ is a tuple $f = (f_{i \mapsto C})$ of functions where each $f_i : \Sigma^* \rightarrow 2^{\Sigma^*}$ maps system traces to a set of local traces for $C_i$, such that for every trace $tr$, we have $tr_{\mapsto C_i} \in f_i(tr)$, and $f_i(tr)$ is prefix closed. Intuitively, $f_i$ describes the fault model for component $C_i$, and given an observed (error) trace $tr \in \Sigma^*$, the set $f_i(tr)$ is the set of possible local counterfactual traces for $C_i$ (which includes $tr_{\mapsto C_i}$ since it was observed). In this generalized setting, the sets $\text{Feasible}^{f_1}_i$ and $\text{Feasible}^{f_2}_i$ define two specific functions: $f^{(1)}_i(tr) = \text{Feasible}^{f_1}_i(C_i)$ and $f^{(2)}_i(tr) = \text{Feasible}^{f_2}_i(C_i)$. Let $\text{CFac}(tr) = f_1(tr) \parallel f_2(tr) \parallel \cdots \parallel f_i(tr)$, where $\Sigma = \{C_1, C_2, \ldots, C_n\}$ (the set $\text{CFac}(tr)$ is defined similarly).

Similarly, the general definition of fault manifestation based causality, generalizing Equations 1 and 2, as follows: the component set $\mathcal{C}$ is fault manifestation capable under the fault-model profile $f$ if

$$\text{Repair}_\mathcal{C}(tr) \parallel \text{CFac}(tr) \subseteq \varphi$$

(4)

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 [9] 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 $f$, and its utilization in causality definitions 4 and 5 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 $\text{Dep}_i$, which others (possibly in the future), $x$, has an associated variable dependency $\text{Dep}_i$. This variable dependency can be utilized in the fault model profile $f_i$ for $C_i$: the set $f_i(tr)$ 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 $C_j$, 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 Language-Theoretic Algorithm Complexity

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 $\Sigma$ and a word $w$, let

• $\text{Cones}_\mathcal{S}(w) = \{|w| \Sigma^*\}$, i.e. the word $w$ followed by all possible strings in $\Sigma^*$;

• for $\Sigma(X) = \Sigma(X') \parallel \Sigma(X'')$, for some alphabets $\Sigma(X')$ and $\Sigma(X'')$, and for $w_p$ a prefix of $w$, let

$$\text{AlterRest}(w_p, w, \Sigma) = \{w_p | u \in \Sigma^* \setminus \{w_p\} \text{ and } (w_p, u) \in \Sigma^*\}.$$

The set $\text{AlterRest}(w_p, w, \Sigma)$ contains words of length $|w|$ obtained from $w_p$ by keeping the first $|w_p|$ letters unchanged, and then changing all letters not in $\Sigma$ to all possible values (this corresponds to changing values 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 follows:

$$\text{Repair}_\mathcal{C}(tr) = \varphi \cap \text{Cones}_\mathcal{S}.$$  \hfill (5)

$$\text{Feasible}^{f_1}_i(C_i) = \text{Prefs}(\text{tr}_{\mapsto C_i}) \cup \text{Repair}_\mathcal{C}(C_i).$$  \hfill (6)

$$\text{Feasible}^{f_2}_i(C_i) = \text{Prefs}(\text{AlterRest}_\mathcal{S}(\text{tr}_{\mapsto C_i}, \text{wmap}, \Sigma)) \cup \text{Repair}_\mathcal{C}(C_i),$$

where where $\text{wmap}$ is the maximal correct prefix (with respect to $\varphi_i$) of $\text{tr}_{\mapsto C_i}$. Specific algorithms for the case of finite automaton to obtain the basic sets used in Equation 6 are as follows. Recall that a non-deterministic finite automaton (NFA) over an alphabet $\Sigma$ is a tuple $A = (Q, \Sigma, \rho, Q_i)$, where $Q$ is a finite set of states, $q_0 \in Q$ is an initial state, $\Sigma$ is the input alphabet, $\rho \subseteq Q \times \Sigma$ is a transition relation, and $Q_i \subseteq Q$ is a set of accepting states. A deterministic automaton (DFA) is one where for any $q \in \Sigma$ and $\sigma \in \Sigma$, there is at most one $q'$ such that $(q, \sigma, q') \in \rho$. We denote $L(A)$ as the language of words in $\Sigma$ accepted by $A$. Define $|\mathcal{A}| = |Q| + |\rho|$ (thus $|\mathcal{A}| \leq |Q| \times |\Sigma|$). Let the local and global specifications $\varphi_1, \ldots, \varphi_n$ and $\varphi$ be given as DFAs or NFAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ and $\mathcal{A}$ respectively. Note that since the specifications are prefix closed, we can assume that all reachable states are final states $[11]$. The various entities in Equation 6 are obtained as follows.

- The string $\text{wmap}$ can be obtained from $\text{tr}$ and $\varphi_i$ in time $O(|\text{tr}| \cdot \Sigma^*)$ by running the automaton $\mathcal{A}_i$ on $\text{tr}_{\mapsto C_i}$ (if $\mathcal{A}_i$ is a DFA, this can be done in $O(|\text{tr}|)$ time). Similarly, string $\text{maxcpx}(\text{tr})$ can be obtained in $O(|\text{tr}| \cdot \Sigma^* \cdot \Sigma^*) \leq O(|\text{tr}|)$ time ($O(|\text{tr}|)$ in the DFA case).
- A DFA $D$, with $|\mathcal{D}|$ states (and size $|\mathcal{D}| + 1$) can be constructed such that $L(D) = \text{Cones}_\mathcal{S}(\text{tr}_{\mapsto C_i}).$
- We can construct a DFA $D_\mathcal{D}$ with $|\mathcal{D}|$ states and size such that $L(D_\mathcal{D}) = \text{Prefs}(\text{tr}_{\mapsto C_i})$ using the standard prefix construction.
- We can construct a DFA for $\text{AlterRest}_\mathcal{S}(\mathcal{D}_w, \Sigma)$ with $|\mathcal{D}_w| \Sigma$ states (and size $|\mathcal{D}_w| \Sigma$). It can be modified to accept $\text{Prefs}(\text{AlterRest}_\mathcal{S}(\mathcal{D}_w, \Sigma))$ by making all states final. Union and intersection, used in Equation 6 are standard operations on NFAs/DFAs. Thus, the sets $\text{Repair}_\mathcal{C}(C_i)$, and $\text{Feasible}^{f_1}_i(C_i)$ and $\text{Feasible}^{f_2}_i(C_i)$ 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 $f = (f_{i \mapsto C})$ such that $f_i(tr) = \text{Feasible}^{f_1}_i(C_i)$ or $f_i(tr) = \text{Feasible}^{f_2}_i(C_i)$. Recall Equations 4 and 5. The equations involved take the parallel composition of the $f_i(tr)$ sets. As we just showed, each $f_i(tr)$ set can be represented as the language of a NFA (or DFA) of polynomial size. The parallel composition of the $f_i(tr)$ 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 $L(\mathcal{B}_1) \subseteq \cdots \subseteq L(\mathcal{B}_n) \subseteq L(\mathcal{A})$, where $B_1$ are (polynomial sized) automata derived as above for either the Repair or Feasible sets. This check can be performed by checking $L(\mathcal{B}_1) \subseteq \cdots \subseteq L(\mathcal{B}_n) \subseteq L(\mathcal{A})$ where $\mathcal{A}$ is the deterministic automaton for $\mathcal{A}$. Putting everything together, we get the following.
Theorem 1. Let \( \hat{f} = (f_i)_{i=1}^{n} \) be a fault-model profile such that \( f_i(tr) = \text{Feasible}_{C_i}^{f_i}(C) \) or \( f_i(tr) = \text{Feasible}_{C_i}^{2}(C) \). Let the local and global specifications \( \phi_1, \ldots, \phi_n \) (such that \( \phi_1 \equiv \ldots \equiv \phi_n \leq \phi \) be given as NFAs (or DFAs) \( A_1, \ldots, A_n \) and \( A \) respectively. Given an observed trace \( tr \notin \phi \), for the fault profile \( \hat{f} \), a component set \( F \) can be determined to be: fault mitigation capable (Equation 4), or fault manifesting (Equation 5) in time (i) polynomial in the sizes of \( A_1, \ldots, A_n \), and \( tr \); and (ii) exponential in \( |A| \) in case \( A \) is an NFA, or polynomial in \( |A| \) in case \( A \) is a DFA.

A careful analysis shows that for fault-model profiles with \( f_i(tr) = \text{Feasible}_{C_i}^{f_i}(C) \) or \( f_i(tr) = \text{Feasible}_{C_i}^{2}(C) \), the fault mitigation capability problem is in co-NP. A similar argument shows that for \( \phi \) consistent counterfactual traces, leading to erroneous results. A component set \( F \) can be determined to be: fault mitigation capable (Equation 4), or fault manifesting (Equation 5) in time (i) polynomial in the sizes of \( A_1, \ldots, A_n \), and \( tr \); and (ii) exponential in \( |A| \) in case \( A \) is an NFA, or polynomial in \( |A| \) in case \( A \) is a DFA.

5. DISCUSSION

In this section we discuss the most prominent definitions of causes from the literature and highlight connections to our definitions of causal sets from Equations 1 through 5.

Causality for Structural Equations. The paper [9] 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 dependencies and requires that some resultant behavior is faulty. Our definition of fault manifesting sets (counterfactual as well as on contingency dependency) is exactly one root cause, while there can be multiple mitigation-capable sets, which will not be discovered by the algorithm in [8].

Counterexample analysis. In [16] 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 [12, 14] 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.

6. REFERENCES


APPENDIX

A. A WEAKER NOTION OF FAULT MITIGATION

We now present a definition of fault mitigation-capable sets which is weakening of the one based on Equation 1. It involves counterfactual sets that are considerably more complicated than the previous ones. In these intricate counterfactual tracesets, we only take traces from the \( \text{Feasible}_\ell \) sets that are as close as possible to the observed trace \( \text{tr} \). In the ideal case that would be the projections of the trace \( \text{tr} \) itself (i.e. the actual observed trace without any hypothetical corrections), but as we already discussed, correctness of the behavior of some components can have the side effect of making the observed local traces of other components infeasible. To this end, we need to include in the set \( \text{Feasible}_\ell \), also correct behaviors of the respective component. The construction, which we now describe, does so only when this is unavoidable.

First, we define for a language \( L \) and some set of components \( \mathcal{C} \) the language \( \text{MinModSel}^L_\ell(\mathcal{C}) \) that consists of those words in \( L \) that are obtained from \( \text{tr} \) by minimal modifications to \( \text{tr}_{\mathcal{C}_\ell} \). Formally, \( \text{MinModSel}^L_\ell(\mathcal{C}) \) contains all the words \( v \in L \) such that for all words \( v \neq u \in L \), either

1. for some \( C_i \in \mathcal{C} \), we have \( \text{lcommpref}(v|\ell,\text{tr}_C) \) to be a strict prefix of \( \text{lcommpref}(u|\ell,\text{tr}_C) \); or
2. for every \( C_i \in \mathcal{C} \), we have \( \text{lcommpref}(v|\ell,\text{tr}_C) \) to be equal to \( \text{lcommpref}(u|\ell,\text{tr}_C) \).

Thus, \( \text{MinModSel}^L_\ell(\mathcal{C}) \) consists of words \( u \) from \( L \) such that there is no other \( v \neq u \in L \), which is a better match to \( \text{tr}_{\mathcal{C}_\ell} \).

With the help of this definition the following set can be defined.

**MitigCbI-MM.** Component set \( \mathcal{C} \) is fault mitigation-capable if

\[
\text{MinModSel}^L_\ell(\text{Repair}_\ell(\mathcal{C})) \supseteq \text{Feasible}^L_\ell(\mathcal{C}) \implies \varphi \tag{7}
\]

Thus, the definition above only requires the new smaller counterfactual set on the left-hand side to be a subset of \( \varphi \), rather than the set \( \text{Repair}_\ell(\mathcal{C}) \supseteq \text{Feasible}^L_\ell(\mathcal{C}) \) as was originally required in Equation 1. Hence, the notion defined here is weaker.

**Proposition 4** (Fault Mitigation-Capable Sets). If set \( \mathcal{C} \) is fault mitigation-capable under Equation 1, then it is also fault mitigation-capable under Equation 7.

**Proposition 5** (Monotonicity). If a set \( \mathcal{C} \) satisfies Equation 2, then any superset \( \mathcal{C}' \supseteq \mathcal{C} \) also satisfies Equation 2.

Remark While this definition is in the flavor of the definition in [5] that requires that for each component the difference between the local counterfactual traces and the observed local trace is minimal, our definition minimizes the difference for the component set as a whole, thus overcoming the deficiency of [5] (the construction there can result in local traces that are not composable).

B. A STRONGER NOTION OF FAULT MANIFESTATION

Now we present a stronger definition of fault manifesting sets, in which the possible counterfactual behaviors of the components in \( \mathcal{C} \) are restricted to prefixes of the observed behavior, and the requirement for the existence of traces violating \( \varphi \) is stronger.

**Strong Fault Manifesting Sets.** Intuitively, a component set \( \mathcal{C} \) is strong-fault manifesting if its observed faulty behavior alone is enough to manifest in a global error (with respect to \( \varphi \)) in some resultant trace, whether the components in \( \mathcal{C} \) were to behave correctly or incorrectly. Recall that \( \mathcal{E} = \{C_1, \ldots, C_n\} \), thus \( \mathcal{E} \) has \( n_\mathcal{E} = n - n_{\mathcal{C}_\ell} \) elements. For each component, consider a function

\[
G^i : \{0, 1\} \rightarrow (\text{Prefs}(\text{tr}_{\mathcal{C}_\ell}), \text{Repair}(\mathcal{C})) \text{ defined by: }
\]

\[
G^i(0) = \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) ;
\]

\[
G^i(1) = \text{Repair}(\mathcal{C}).
\]

Now consider the natural extension to \( \mathcal{E} \), where \( G^\mathcal{E} : \{0, 1\}^\mathcal{E} \rightarrow \text{Feasible}^\mathcal{E} \) defined by:

\[
G^\mathcal{E}(b_1, b_2, \ldots, b_{n_{\mathcal{C}_\ell}}) = G^1(b_1) \parallel G^2(b_2) \parallel \ldots \parallel G^{n_{\mathcal{C}_\ell}}(b_{n_{\mathcal{C}_\ell}})
\]

That is, the boolean vector \((b_1, b_2, \ldots, b_{n_{\mathcal{C}_\ell}})\) tells us whether to choose \( \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) \), or \( \text{Repair}(\mathcal{C}) \) for each component of \( \mathcal{E} \) in the composition.

**Manifest-Strong.** Set \( \mathcal{E} \) is strongly-fault manifesting if

\[
\forall (b_1, \ldots, b_{n_{\mathcal{C}_\ell}}) \in \{0, 1\}^{n_{\mathcal{C}_\ell}} \left[ \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) \parallel G^\mathcal{E}(b_1, b_2, \ldots, b_{n_{\mathcal{C}_\ell}}) \not\subseteq \varphi \right]
\]

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

We claim that Equation 8 enables us to make a bullet-proof argument that \( \mathcal{E} \) is for sure to blame for the violation of \( \varphi \). The defining criterion ensures that no matter which subset of components of \( \mathcal{E} \) were to be corrected, some resulting trace in the composition with the observed \( \mathcal{C} \) behavior would have resulted in a violation of \( \varphi \). However, since Equation 8 is a strong condition, it classifies fewer sets as fault manifesting.

**Example 10** (Fault Manifestation). We consider the system and the error trace from Example 4 and determine that \( \{C_2, C_4\} \) is strongly fault manifesting.

Since the set \( \mathcal{C} \) contains the single component \( C_1 \), Equation 8 amounts to the two conditions \( \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) \parallel \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) \not\subseteq \varphi \) and \( \text{Prefs}(\text{tr}_{\mathcal{C}_\ell}) \parallel \text{Repair}(\mathcal{C}) \not\subseteq \varphi \). Since \( \text{tr}_{\mathcal{C}_\ell} \in \text{Repair}(\mathcal{C}) \), reasoning as in Example 9 we conclude that \( \{C_2, C_4\} \) is also strongly fault manifesting.

**Proposition 6** (Fault Manifesting Sets). If \( \mathcal{E} \) is fault manifesting set under Equation 8, it is also fault manifesting under Equation 3.

**Proposition 7** (Interrelationships). 1. If \( \mathcal{E} \) is a fault mitigation-capable set according to Equation 1, then \( \mathcal{E} \) is not a fault manifesting set according to Equation 8.

2. If \( \mathcal{E} \) is a fault manifesting set according to Equation 8, then \( \mathcal{E} \) is not fault mitigation-capable under Equation 1.

**Proposition 8** (Monotonicity). If a set \( \mathcal{E} \) satisfies Equation 8, any superset \( \mathcal{E}' \supseteq \mathcal{E} \) also satisfies Equation 8.

Application. In liability resolution, e.g., in a legal context, we need to prove, beyond reasonable doubt, that a component set \( \mathcal{E} \) is “responsible” for an erroneous behavior. This is extremely difficult to do in most cases for concurrent reactive systems where different components are owned by different companies, and where opportunities for passing blame abound. The condition defining the set Manifest-Strong identifies the case when it indeed is possible to prove, beyond reasonable doubt, that \( \mathcal{E} \) was responsible for the global violation. Only the incorrect behaviors of \( \mathcal{E} \) are considered, and the condition states that no matter what \( \mathcal{E} \) would have done, correct, or (the observed) incorrect, a violation would have occurred in some resultant trace. On the flip side, when arguing that a component set \( \mathcal{E} \) is not liable, while showing that \( \mathcal{E} \) is not a Manifest-Strong set is the first step, also proving that it is (not some of) the other sets makes for a stronger defense.
C. COMPUTATIONAL COMPLEXITY OF CAUSALITY PROBLEMS

Let \( \mathcal{A}, \mathcal{B} \) be NFAs over an alphabet \( \Xi \), such that \( L(\mathcal{A}) \) and \( L(\mathcal{B}) \) 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 [11]. We will now show how to reduce the question \( L(\mathcal{B}) \subseteq L(\mathcal{A}) \) 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 \( x \) be a variable with domain \( \Xi \).

We define the component \( C_1 = (X_1, \text{inp}(X_1), \text{out}(X_1), \Sigma_1, \varphi_1) \), where \( X_1 = \{x \in x_1, x_2\} \), \( \text{inp}(X_1) = \{x_2\} \), \( \text{out}(X_1) = \{x_1, x_2\} \) and

\[
\varphi_1 = (L(\mathcal{A}) \parallel ((a_1) \cdot O_1) \parallel ((a_2) \cdot O_2)) \cup \nabla (L(\mathcal{B}) \parallel ((a_1) \cdot O_1) \parallel ((b_2) \cdot O_2^i) \parallel \{e\}).
\]

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

For the other component, let \( C_2 = (X_2, \text{inp}(X_2), \text{out}(X_2), \Sigma_2, \varphi_2) \), where \( X_2 = \{x \in x_2\} \), \( \text{inp}(X_2) = \emptyset \), \( \text{out}(X_2) = \{x_2\} \) and \( \varphi_2 = (a_2) \cdot O_2^i \cup \{e\} \).

Finally, we define the global specification as

\[
\varphi = L(\mathcal{A}) \parallel ((a_1) \cdot O_1) \parallel O_2^i \cup \{e\}.
\]

Thus, we clearly have that \( \varphi_1 \parallel \varphi_2 \subseteq \varphi \), regardless of \( L(\mathcal{B}) \). Fix the fault profile \( f : f_1(\text{tr}) = \Xi^* \parallel O_1^i \parallel O_2^i \) and \( f_2(\text{tr}) = O_2^i \).

Consider the trace \( \text{tr} \) of length 1, where for the first letter we have \( x_1 = b_1 \), \( x_2 = b_2 \) and \( x_2 = x \) for some letter \( \xi \in \Xi \). Clearly \( \text{tr} \notin \varphi_1, \varphi_2, \varphi \). Repair\( \mathcal{A} \) \( \mathcal{B} \) \( C_1 \) and CF\( \mathcal{A} \) \( \mathcal{B} \) \( C_2 \) are \( b_2 \cdot O_2^i \cup \{e\} \).

Thus, the set \( \xi = \{C_1\} \) is fault mitigation-capable w.r.t. \( f \) iff

\[
\varphi_1 \parallel (b_2) \cdot O_2^i \subseteq L(\mathcal{A}) \parallel (a_1) \cdot O_1^i \parallel O_2^i \quad \text{iff} \quad \nabla (L(\mathcal{B}) \parallel (a_1) \cdot O_1^i \parallel (b_2) \cdot O_2^i \subseteq L(\mathcal{B}) \parallel (a_1) \cdot O_1^i \parallel O_2^i)
\]

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

Given the automata \( \mathcal{A} \) and \( \mathcal{B} \), in time polynomial in their size we can construct NFAs for \( \varphi_1 \) and \( \varphi \) by extending their alphabet by \( O_1 \parallel O_2 \). The NFAs for \( \varphi_2 \), \( f_1(\cdot) \) and \( f_2(\cdot) \) are of constant size. Thus, we can reduce \( L(\mathcal{B}) \subseteq L(\mathcal{A}) \) to checking fault mitigation/fault manifestation for a suitable fault model.