Counterfactual Explanations for MITL Violations

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Felix Jahn
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Julian Siber
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

MITL is a temporal logic that facilitates the verification of real-time systems by expressing the critical timing constraints placed on these systems. MITL specifications can be checked against system models expressed as networks of timed automata. A violation of an MITL specification is then witnessed by a *timed trace* of the network, i.e., an execution consisting of both discrete actions and real-valued delays between these actions. Finding and fixing the root cause of such a violation requires significant manual effort since both discrete actions and real-time delays have to be considered. In this paper, we present an automatic explanation method that eases this process by computing the root causes for the violation of an MITL specification on the execution of a network of timed automata. This method is based on newly developed definitions of counterfactual causality tailored to networks of timed automata in the style of Halpern and Pearl's actual causality. We present and evaluate a prototype implementation that demonstrates the efficacy of our method on several benchmarks from the literature.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Modal and temporal logics

Keywords and phrases Timed automata, actual causality, metric interval temporal logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.22

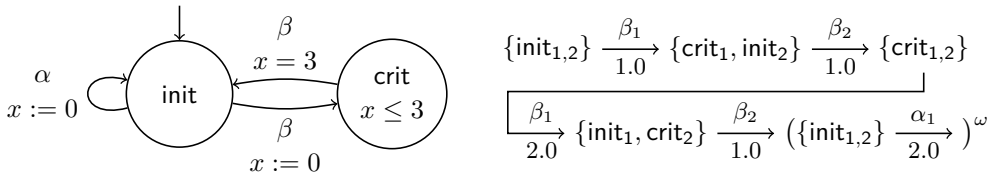
Supplementary Material
Software: `https://github.com/reactive-systems/rt-causality.git` [44]
archived at `swh:1:dir:de6b34eb1137d85c4257b5adac4b15646bd8ea3e`

Funding This work was partially supported by the DFG in project 389792660 (Center for Perspicuous Systems, TRR 248) and by the ERC Grant HYPER (No. 101055412).

1 Introduction

Networks of timed automata are a popular formalism to model a wide range of real-time systems such as automotive controllers [27, 50] and communication protocols [23, 39]. These models can be automatically checked against specifications in *Metric Interval Temporal Logic* (MITL) [4], a real-time extension of linear-time temporal logic that allows to constrain temporal operators to non-singleton intervals over the real numbers. In case a network of timed automata does not satisfy an MITL specification, a model-checking procedure will return an execution of the network as a counterexample. Such an execution is defined by discrete actions of the automata in the network and by real-valued delays that describe the time that passes between the discrete actions. Hence, fixing an erroneous system requires insight into both actions and delays that caused the violation on the given counterexample.

In this paper, we present an approach that facilitates this insight through counterfactual explanations for the observed violation. Previous approaches for explaining real-time violations only consider safety properties [52] or only real-time delays without discrete actions [46], and hence cannot provide a comprehensive insight for violations of unconstrained MITL



(a) A component automaton of network $\mathcal{A}_1 \,||\, \mathcal{A}_2$. **(b)** An execution of the network $\mathcal{A}_1 \,||\, \mathcal{A}_2$.

Figure 1 The network and its execution discussed as an illustrative example in Subsection 1.1.

properties. Like related efforts for discrete systems [10, 20], we ground our explanation method in the theory of *actual causality* as formalized by Halpern and Pearl [34, 35, 36] and identify the actions and delays that are actual causes for the violation of the specification on the observed counterexample. This approach faces several new challenges when confronted with real-time models expressed as networks of timed automata, instead of the previously considered structural equation models [35], finite-state machines [20], and traces [10].

The first challenge pertains to the concept of *interventions*, which describe how the observed counterexample is modified when hypothetical counterfactual executions are considered during the analysis. While previous results usually consider models where the set of counterfactual scenarios is finite, modifying delays in executions of timed automata gives rise to infinitely many counterfactual scenarios. Our main insight to solve this problem is based on constructing networks of timed automata that model all such counterfactual executions, such that checking a causal hypothesis or even synthesizing a cause from scratch can be realized through model checking of these newly constructed automata. Actual causality in models with infinitely many variables, each potentially having an infinite domain, is only starting to be understood [37] and our results suggest that known techniques from timed automata verification are partially transferable to this general theory, e.g., for cause computation.

A second challenge we face in networks of timed automata pertains to the concept of *contingencies*. When two or more potential causes preempt each other, contingencies allow to isolate the true, non-preempted cause from the others. In structural equations models [33] and Coenen et al.'s definition for finite-state machines [20], this is realized by extending the system dynamics with resets that set variables back to the value they had in the actual, original scenario. Networks of timed automata have both local variables, i.e., component locations, and global variables such as clocks. We account for this by defining two automata constructions that allow such resets through contingencies on the local level by single components, as well as on the network level for global clock variables.

1.1 Illustrative Example

We discuss our approach for causal analysis with the example of a small network of timed automata $\mathcal{A}_1 \,||\, \mathcal{A}_2$ consisting of two identical component automata as depicted in Figure 1a, which we will also use as a running example throughout the paper. The two automata can each switch between the two locations init and crit, but whenever they enter the location crit with action β, they are required to stay there for exactly three time units. This is realized through an initial reset of a global clock variable ($x := 0$) with the first β action and a location invariant ($x \leq 3$) in location crit, as well as a clock guard ($x = 3$) on the second β action. We want to check the mutual exclusion property $\square_{[0,\infty)}(\neg \mathsf{crit}_1 \vee \neg \mathsf{crit}_2)$ expressed in MITL, which states that the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are never both in location crit. It is easy to see that this property is violated, e.g., by the execution depicted in Figure 1b. This

Table 1 A contrastive overview of the four root causes on the execution in the illustrative example, inferred using but-for causality and actual causality.

Ref.	But-For Causes	Actual Causes	Intuitive Description
1	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_1)\}$	The first component did not wait.
2	$\{(2.0, 1, \mathcal{A}_2)\}$	$\{(2.0, 1, \mathcal{A}_2)\}$	The second component did not wait.
3	**a:** $\{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$ **b:** $\{(\beta, 1, \mathcal{A}_1), (3.0, 2, \mathcal{A}_1)\}$	$\{(\beta, 1, \mathcal{A}_1)\}$	The first component entered crit.
4	$\{(\beta, 1, \mathcal{A}_2)\}$	$\{(\beta, 1, \mathcal{A}_2)\}$	The second component entered crit.

(simplified) execution is an infinite sequence of location labels constructed from delays and discrete actions, where the ω-part is repeated infinitely often. For abbreviation, we place the delay values and actions on the same arrow, which means that the action above the arrow is performed after delaying for as long as specified under the arrow. Both action and location labels refer to a component automaton performing the action and being in a location, respectively, through their index. The execution depicted in Figure 1b respects the dynamics of the automata, e.g., exactly three time units pass between entering and leaving crit. As we can see, Automaton 2 uses a β action less than three time units after Automaton 1, while the latter needs to stay in crit for exactly three time units.

We generate explanations through counterfactual reasoning: For instance, we can infer that one cause of the violation above is that the second component waits only two time units before entering crit by considering hypothetical executions with alternative delays at this particular point, all else being the same. This relaxed model allows an execution where the second component waits with entering the crit location until after the first component has already left theirs, such that no violation occurs. Hence, we can infer the *but-for* cause $\{(2.0, 1, \mathcal{A}_2)\}$ which says that the first delay of 2.0 time units by component $\mathcal{A}_2$ is a root cause for the violation. We measure delays locally on the component level and hence need to add all global delays between actions of the second component as defined in the execution above. Table 1 lists this cause (Cause 2) along with the other root causes inferred through *but-for* causal analysis in the second column. Cause 1 expresses that the delay of component $\mathcal{A}_1$ can similarly be set high enough that no violation occurs.

Cause 3 shows that the *but-for* counterfactual analysis is not always enough: With this naïve criterion we cannot infer that the first β action of $\mathcal{A}_1$ is a cause for the violation of the property on this execution, since changing it alone to, e.g., α, does not suffice to avoid the violation. The second β then steps in to produce the same effect, which means we are dealing with a *preemption* of potential causes. In the but-for causal analysis, we consequently have to additionally intervene on the preempted causes to obtain executions to avoid the effect. In this case, we can either additionally change the second β (Cause 3a), or the second delay (Cause 3b – this way we can set the entering of crit to after component $\mathcal{A}_2$ has already left). These larger causes are not desirable, because they do not only point to the root of the issue. As a solution in such cases of preemption, Halpern and Pearl [35] suggest *contingencies*, and Coenen et al. [20] have recently lifted this to finite-state machines with infinite executions. Inspired by these efforts, we propose a contingency mechanism for networks of timed automata that similarly allows us to infer the first β as the true cause in the given scenario (Cause 3). This mechanism extends the network with contingency edges that, e.g., allow the second β to move to the same location as in the original execution, i.e., to init. This then produces a witnessing counterfactual run that avoids the effect.

1.2 Outline and Contributions

After recalling preliminaries in Section 2, we develop our definitions of counterfactual causality in networks of timed automata (Section 3). We follow Halpern's approach [34] in first defining a notion of *minimal but-for* causality. Counterfactual reasoning is realized through an automaton construction that allows to search for a witnessing intervention in the infinite set of counterfactual runs through model checking. Inspired by Coenen et al. [20, 21], we extend but-for causality through a construction of contingency automata, which model contingencies on the local level of components as well as on the network level for global variables such as clocks (Subsection 3.3), yielding a main building block for our definition of *actual* causality. In Section 4, we present algorithms for computing and checking but-for and actual causes. These algorithms exploit a property of both notions of causality that we term cause *monotonicity*, which allows us to reduce the potential causes we need to consider during computation. We have implemented a prototype of this algorithm and report on its experimental evaluation in Section 5. We show that causes can be computed in reasonable time and help in narrowing down the behavior responsible for an MITL violation. To summarize, we make the following contributions:

- We define and study the notions of but-for causality and actual causality in networks of timed automata, for effects described by arbitrary MITL properties;
- We propose an algorithm for computing these causes and study its theoretical complexity;
- We report the results of a prototype implementation of this algorithm for automated explanations of counterexamples in real-time model checking.

2 Preliminaries

We recall background on actual causality, timed automata as models of real-time systems, and MITL as a temporal logic for specifying real-time properties.

2.1 Actual Causality

We recall Halpern's modified version [33] of actual causality [35], which uses *structural equation models* to define the causal dependencies of a system. Formally, a *causal model* is a tuple $M = (\mathcal{S}, \mathcal{F})$ that consists of a *signature* $S = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ and *structural equations* $\mathcal{F} = \{F_X \mid X \in \mathcal{V}\}$. The sets $\mathcal{U}$ and $\mathcal{V}$ define *exogenous variables* and *endogenous variables*, respectively. The *range* $\mathcal{R}(Y)$ specifies the possible values of each variable $Y \in \mathcal{Y} = \mathcal{U} \cup \mathcal{V}$. A structural equation $F_X \in \mathcal{F}$ defines the value of an endogenous variable $X \in \mathcal{V}$ as a function $F_X : (\times_{Y \in \mathcal{Y} \setminus \{X\}} \mathcal{R}(Y)) \to \mathcal{R}(X)$ of the values of all other variables in $\mathcal{U} \cup \mathcal{V}$, without creating cyclic dependencies in $\mathcal{F}$. Therefore, the structural equations have a unique solution for a given *context* $\vec{u} \in (\times_{U \in \mathcal{U}} \mathcal{R}(U))$, i.e., a valuation for the variables in $\mathcal{U}$. Actual causality then defines whether a value assignment $\vec{X} = \vec{x}$ causes φ, a conjunction of *primitive events* $Y = y$ for $Y \in \mathcal{V}$, in a given context.

▶ **Definition 1** (Halpern's Version of Actual Causality [33]). *$\vec{X} = \vec{x}$ is an actual cause of φ in $(M, \vec{u})$, if the following three conditions hold:*

AC1. $(M, \vec{u}) \models \vec{X} = \vec{x}$ *and* $(M, \vec{u}) \models \varphi$.

AC2. *There is a* contingency $\vec{W} \subseteq \mathcal{V}$ *with* $(M, \vec{u}) \models \vec{W} = \vec{w}$ *and a setting* $\vec{x}'$ *for the variables in* $\vec{X}$ *s.t.* $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}]\neg\varphi$.

AC3. $\vec{X}$ *is minimal, i.e., no strict subset of* $\vec{X}$ *satisfies* ***AC1*** *and* ***AC2***.

AC1 simply states that both the cause and the effect have to be satisfied in the given context $\vec{u}$ and causal model $\mathcal{M}$. AC2 appeals to an *intervention* $\vec{X} \leftarrow \vec{x}'$ that overrides the structural equations for all $\vec{X}_i \in \vec{X}$ such that $F_{\vec{X}_i} = \vec{x}'_i$. While the witness $\vec{x}'$ can be chosen arbitrarily, the valuation $\vec{w}$ for the *contingency* variables $\vec{W}$ has to be the same as in the original context. The contingency is applied after the intervention, and in this way allows to reset certain variables to their original values, with the aim to infer more precise causes in certain scenarios. Hence, AC2 requires that some intervention together with a contingency avoids the effect, i.e., the resulting solution to the modified structural equations falsifies at least one primitive event in φ. AC3 ensures that $\vec{X} = \vec{x}$ is a concise description of causal behavior by enforcing minimality. In particular, this ensures that for no variable the valuation in $\vec{x}'$ (AC2) coincides with its original valuation in $\vec{x}$.

▶ **Example 2.** We recall a classic example of Suzy and Billy throwing rocks at a bottle [35]. We have the endogenous variables BT, ST for Billy and Suzy throwing their rock, respectively. BH, SH signify that they hit, and BB encodes that the bottle breaks from a hit. BT and ST directly depend on some nondeterministic exogenous variables, while the other structural equations are $BH = BT \wedge \neg ST$, $SH = ST$ and $BB = BH \vee SH$, i.e., Suzy's throw is always faster than Billy's. Hence, in the context where both throw their rock, we have $BT = ST = SH = BB = 1$ and $BH = 0$. The *intervention* $ST = 0$ does not suffice to avoid the effect, because the structural equations still evaluate to $BB = 1$ due to Billy's throw. We say Billy's throw was *preempted*. We can pick the contingency $BH = 0$ from the original evaluation as a *contingency*. This means we set both $ST = 0$ and $BH = 0$, the latter of which "blocks" the influence of Billy's throw, and obtain an evaluation where the effect disappears, i.e., with $BB = 0$. Finally, only the event $ST = 1$ is in the cause.

2.2 Networks of Timed Automata

We use networks of timed automata [1] to model real-time systems. We fix a finite set AP of *atomic propositions* and a finite set of *actions* Act. Given a set of real-valued *clocks* X, a *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ with $x, y \in X$, $\sim \in \{<, \leq, =, \geq, >\}$, and $n \in \mathbb{N}$. The set of clock constraints over a clock set X is denoted $\mathsf{C}(X)$. Then, a *timed automaton* is a tuple $\mathcal{A} = (Q, q_0, X, E, I, L)$, where Q is a finite set of *locations*, $q_0 \in Q$ is the *initial location*, X is a finite set of *clocks*, $E \subseteq (Q \times \mathsf{C}(X) \times Act \times \mathsf{U}(X) \times Q)$ is the *edge relation*, $I : Q \to \mathsf{C}(X)$ is an *invariant assignment*, and $L : Q \to 2^{AP}$ is a *labeling function*. We consider a version of *updatable* timed automata that can reset clocks to constants [16]. Hence, the set of *clock updates* $\mathsf{U}(X)$ is the set of partial functions mapping clocks to natural numbers: $\mathsf{U}(X) = \{U : X \rightharpoonup \mathbb{N}\}$. A *clock assignment* for a set of clocks X is a function $u : X \to \mathbb{R}_{\geq 0}$. u_0 denotes the assignment where all clocks are mapped to zero. We write $u \models g$ if u satisfies a clock constraint $g \in \mathsf{C}(C)$, $u + \delta$ for the clock assignment that results from u after $\delta \in \mathbb{R}_{\geq 0}$ time units have passed, i.e., $(u + \delta)(x) = u(x) + \delta$, and $u \leftarrow U$ for the assignment that updates u in accordance with U, i.e., $(u \leftarrow U)(x) = U(x)$ if $x \in dom(U)$ else $(u \leftarrow U)(x) = u(x)$.

▶ **Definition 3** (Semantics of Timed Automata). *The semantics of a timed automaton* $\mathcal{A} = (Q, q_0, X, E, I, L)$ *is defined by a transition system* $(Q \times \mathbb{R}_{\geq 0}^{|X|}), (q_0, u_0), \to)$, *where* $\to$ *contains:*

delays: $(q, u) \xrightarrow{\delta} (q, u + \delta)$ *iff* $\delta \in \mathbb{R}_{\geq 0}$ *and* $(u + \delta') \models I(q)$ *for all* $0 \leq \delta' \leq \delta$, *and*

actions: $(q, u) \xrightarrow{\alpha} (q', u \leftarrow U)$ *iff* $(q, g, \alpha, U, q') \in E$, $u \models g$, *and* $(u \leftarrow U) \models I(q')$.

A *run* $\rho = (q_0, u_0) \xrightarrow{\delta_1}\xrightarrow{\alpha_1} (q_1, u_1) \xrightarrow{\delta_2}\xrightarrow{\alpha_2} \dots$ of $\mathcal{A}$ is a sequence of alternating delay and action transitions. The set $\Pi(\mathcal{A})$ is the set of all runs of $\mathcal{A}$. The *trace* $\pi(\rho) = \langle \delta_1^\rho, \alpha_1^\rho \rangle \langle \delta_2^\rho, \alpha_2^\rho \rangle \dots$ of a run ρ is the sequence of delay and action transitions. We sometimes denote the elements of some run ρ or trace π at index i with q_i^ρ, δ_i^ρ etc. We define the accumulated delay as $\delta(i,j) = \sum_{k=i,\dots,j} \delta_k$ and $\delta_0 = 0$. The *signal* σ^ρ of the run ρ maps time points to location labels: $\sigma^\rho(t) = \{a \mid \exists i.\ a \in L(q_i) \wedge \delta(0,i) \leq t < \delta(0,i+1)\}$. The language $\mathcal{L}(\mathcal{A})$ is the set of all signals with a corresponding run of $\mathcal{A}$. We use this *left-closed right-open* interpretation of signals due to Maler et al. [51] because of its simplicity. It is straightforward to extend our counterfactual analysis technique to other semantics, e.g., continuous time and point wise [4], or even to other logics with linear-time semantics, as long as their model checking problem is decidable. Note that we make use of an intersection operation $\cap$ for timed automata which intersects the *actions*, i.e., the edge label of the automata. You may assume that the operation unifies the labels of the locations, but we apply it such that only one operand automaton has location labels. This means that the result of $\mathcal{A}_1 \cap \mathcal{A}_2$ is not (singal-based) language intersection in the classical sense, i.e., we do not have $\mathcal{L}(\mathcal{A}_1 \cap \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

▶ **Definition 4** (Network of Timed Automata). *A network of timed automata $\mathcal{A}_1 \,||\, \dots \,||\, \mathcal{A}_n$ is constructed through parallel composition. Let $\mathcal{A}_i = (Q^i, l_0^i, X, E^i, I^i, L^i)$ for all $1 \leq i \leq n$ with a common set of global clocks X. The network $\mathcal{A}_1 \,||\, \dots \,||\, \mathcal{A}_n$ is defined by the automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$, where the locations are the Cartesian product $Q = Q^1 \times \dots \times Q^n$, with the initial state $q_0^n = (q_0^1, \dots, q_0^n)$, the invariants are combined as $I(\vec{q}) = \bigwedge_{1 \leq i \leq n} I^i(q_i)$, and the labels are unified as $L(q) = \bigcup_{1 \leq i \leq n} L^i(q_i)$. The edge relation E contains two types:*
internal: $\big(\vec{q}, g, \langle \mathcal{A}_i, \mathcal{A}_i, \alpha \rangle, U, \vec{q}\,[q_i'/q_i]\big)$ *iff* $(q_i, g, \alpha, U, q_i') \in E^i$, *and*
synchronized: $\big(\vec{q}, g_i \wedge g_j, \langle \mathcal{A}_i, \mathcal{A}_j, \alpha \rangle, U_i \cup U_j, \vec{q}\,[q_i'/q_i, q_j'/q_j]\big)$ *iff* $i \neq j$, $(q_i, g_i, \alpha, U, q_i') \in E^i$, *and* $(q_j, g_j, \bar{\alpha}, U, q_j') \in E^j$.

Hence, we do explicitly identify the component automata participating in action transitions by constructing tuples containing actions and automata handles. This is for technical convenience in later constructions, and we define a predicate to check whether an automaton participates in an action transition as $\mathit{participates}(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) := (i = j) \vee (i = k)$, as well as a partial function for accessing the original action as $\mathit{action}(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) = \alpha$ iff $i = j$ and $\mathit{action}(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) = \bar{\alpha}$ iff $i = k$.

2.3 Metric Interval Temporal Logic

We use *Metric Interval Temporal Logic* (MITL) [4] for defining real-time properties such as system specifications and effects. The syntax of MITL formulas over a set of atomic propositions AP is defined by $\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathcal{U}_I\, \phi$, where $p \in AP$ and I is a non-singleton interval of the form $[a,b]$, $(a,b]$, $[a,b)$, (a,b), (a,∞), or $[a,\infty)$ with $a, b \in \mathbb{N}$ and $a < b$. We also consider the usual derived Boolean and temporal operators ($\Diamond_I\ \phi := \top\, \mathcal{U}_I\ \phi$, $\Box_I\ \phi := \neg \Diamond_I \neg\phi$, $\phi\, \mathcal{U}\, \psi := \phi\, \mathcal{U}_{[0,\infty)}\, \psi$, $\Diamond\ \phi := \Diamond_{[0,\infty)}\ \phi$, and $\Box\ \phi := \Box_{[0,\infty)}\ \phi$).
The semantics of MITL is defined inductively with respect to a signal $\sigma : \mathbb{R}_{\geq 0} \to 2^{AP}$ and a timepoint $t \in \mathbb{R}_{\geq 0}$.

$\sigma, t \models p$	iff	$p \in \sigma(t)$
$\sigma, t \models \neg\phi$	iff	$\sigma, t \not\models \phi$
$\sigma, t \models \phi \wedge \psi$	iff	$\sigma, t \models \phi$ and $\sigma, t \models \psi$
$\sigma, t \models \phi\, \mathcal{U}_I\ \psi$	iff	$\exists t' > t.\ t' - t \in I,\ \sigma, t' \models \psi$ and $\forall t'' \in (t, t').\, \sigma, t'' \models \phi$

A run ρ *satisfies* an MITL formula ϕ, iff $\sigma(\rho), 0 \models \phi$. A timed automaton $\mathcal{A}$ *satisfies* ϕ, iff all of its runs satisfy ϕ. We write $\rho \models \phi$ and $\mathcal{A} \models \phi$, respectively.

3 Counterfactual Causality in Real-Time Systems

In this section, we develop two notions of counterfactual causality in real-time systems. We first define our language for describing causes and how they define interventions on timed traces (Subsection 3.1). We then start with a simple notion of *but-for* causality in networks of timed automata based on interventions without contingencies (Subsection 3.2). Afterward, we outline how to model contingencies in a timed automaton (Subsection 3.3) and use them to define *actual* causes, in the sense of Halpern and Pearl (cf. Subsection 2.1). Note that the proofs of all nontrivial statements of this section are in Appendix A.

3.1 Interventions on Timed Traces

We describe actual causes as finite sets of *events*. Events have two distinct types such that they either refer to an action or a delay transition in a given run.

▶ **Definition 5** (Event). *A delay event is a tuple* $(\delta, i) \in \mathbb{R}_{\geq 0} \times \mathbb{N}$ *and an action event is a tuple* $(\alpha, i) \in Act \times \mathbb{N}$*. The sets of all delay and action events are denoted as* $\mathcal{DE}$ *and* $\mathcal{AE}$*, respectively. The set of all events is* $\mathcal{E} = \mathcal{DE} \dot{\cup} \mathcal{AE}$*. For a trace* π*, the set of events on* π *is defined as* $\mathcal{E}_\pi = \{(\alpha_i^\pi, i) \mid i \in \mathbb{N}_{>0}\} \cup \{(\delta_i^\pi, i) \mid i \in \mathbb{N}_{>0}\}$.

When we describe a cause as a set of events, we are mainly interested in the counterfactual runs obtained by modifying the events contained in the cause. In the style of Halpern and Pearl, we call such modifications *interventions*. If the actual run is given in a finite, lasso-shaped form and the cause is a finite set of events, these interventions can be described by a timed automaton that follows the dynamics of the actual trace, except for events that appear in the cause. For these events, the behavior is relaxed to allow arbitrary alternative actions or delays. We call a run ρ *lasso-shaped* if it can be composed of a (possibly empty) prefix and an infinitely occurring loop, i.e., if it is of the form

$$\rho = (q_0, u_0) \dots \big((q_n, u_n) \dots (q_{p-1}, u_{p-1}) \xrightarrow{\delta_p} \xrightarrow{\alpha_p} \big)^\omega \ ,$$

where the ω-part is repeated infinitely often. Note that strictly speaking, a lasso-shaped trace as defined here does not exists for all models that violate an MITL property, because clock valuations are not guaranteed to stabilize in some infinitely-repeating loop $u_n \dots u_{p-1}$. We use the valuations to reset clock values in our contingency construction that will be introduced in Subsection 3.3. This construction may be generalized by considering clock *regions* or *zones* [14] instead of the valuations. This requires defining the resets in the contingency automaton accordingly.

In this paper, we simplify by assuming the existence of a lasso-shaped run as defined above. In general, we can further assume the clocks to be assigned to values in $\mathbb{Q}$, as timed automata do not distinguish between the real and rational numbers [3]. For a lasso-shaped run ρ as described above, we define a function to access the successor index of an action as $dst_\rho : \{1, \dots, p\} \mapsto \{0, \dots, p-1\}$ with $dst_\rho(k) = k$ if $k \neq p$ and $dst_\rho(p) = n$ else. We define the length of the run ρ as $|\rho| = p$. The functions dst_π and $|\pi|$ are defined analogously for the trace of a lasso-shaped run. We are now ready to define the automaton modelling traces with interventions.

$$\frac{(\delta_i^\pi, i) \notin \mathcal{C} \qquad (\alpha_i^\pi, i) \notin \mathcal{C}}{\big(i-1, d = \delta_i^\pi, \alpha_i^\pi, d := 0, dst_\pi(i)\big) \in E} \qquad \frac{(\delta_i^\pi, i) \in \mathcal{C} \qquad (\alpha_i^\pi, i) \notin \mathcal{C}}{\big(i-1, \top, \alpha_i^\pi, d := 0, dst_\pi(i)\big) \in E}$$

$$\frac{(\delta_i^\pi, i) \notin \mathcal{C} \qquad (\alpha_i^\pi, i) \in \mathcal{C} \qquad \beta \in Act}{\big(i-1, d = \delta_i^\pi, \beta, d := 0, dst_\pi(i)\big) \in E} \qquad \frac{(\delta_i^\pi, i) \in \mathcal{C} \qquad (\alpha_i^\pi, i) \in \mathcal{C} \qquad \beta \in Act}{\big(i-1, \top, \beta, d := 0, dst_\pi(i)\big) \in E}$$

Figure 2 Rules defining the edge relation E of the counterfactual trace automaton $\mathcal{A}_\pi^{\mathcal{C}}$.

▶ **Definition 6** (Counterfactual Trace Automaton). *Let π be a lasso-shaped trace over the set of actions Act and let $\mathcal{C} \subseteq \mathcal{E}_\pi$ be a finite set of events. The counterfactual trace automaton of trace π for the set of events $\mathcal{C}$ is defined as $\mathcal{A}_\pi^{\mathcal{C}} := (Q, q_0, X, E, I, L)$ with $Q := \{0, \dots, |\pi|-1\}$, $q_0 := 0$, $X := \{d\}$. The transition relation E is defined by the following rules depicted in Figure 2, we have $L(q) = \emptyset$ for all $q \in Q$, and*

$$I(q) := \begin{cases} d \leq \delta_{q+1}^\pi, & \text{if } (\delta_{q+1}^\pi, q+1) \notin \mathcal{C} \\ \top, & \text{otherwise.} \end{cases}$$

The main idea of the counterfactual automaton $\mathcal{A}_\pi^{\mathcal{C}}$ is to follow the actions and delays of the original run for all events that are not in the event set $\mathcal{C}$, and allow arbitrary action and delays for events in $\mathcal{C}$. Hence, $\mathcal{A}_\pi^{\mathcal{C}}$ modifies the *trace* of ρ, i.e., the sequence of action and delay events. Subsequently, we will combine a local $\mathcal{A}_\pi^{\mathcal{C}}$ with the dynamics of the original components to obtain full counterfactual runs of a network of timed automata. The interventions on actions and delays are captured by the rules that define the transition relation and are listed in Figure 2, which treat the different combination of events that may or may not be in the cause at a specific index i. Crucially, the automaton $\mathcal{A}_\pi^{\mathcal{C}}$ then captures not just a single concrete intervention on the events in $\mathcal{C}$ with respect to the run ρ, such as a modified trace with a specific alternative delay deviating from the actual trace, but *all* (possibly infinitely many) interventions on the events, i.e., it contains all traces with possibly varying actions and delays at specific indices.

▶ **Example 7.** For the trace $\pi = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$ and the set of events $\mathcal{C} = \{(\beta, 1), (3.0, 2)\}$, we depict the counterfactual trace automaton $\mathcal{A}_\pi^{\mathcal{C}}$ in Figure 3. For the first action and the second delay, arbitrary interventions are allowed, all other action and delay events are enforced to be as in π.

3.2 But-For Causality in Networks of Timed Automata

We now use the construction from the previous section to define counterfactual causes for MITL-expressible effects on runs of networks of timed automata. In practice, an effect ϕ may be the violation of a specification ψ, such that the effect corresponds to the negation of the specification: $\phi \equiv \neg\psi$. The main idea of our definition is to isolate the local traces of the

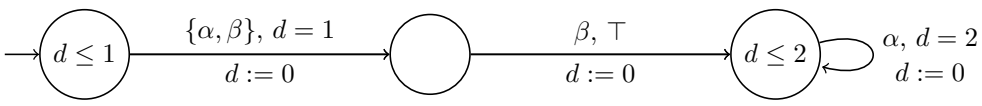


Figure 3 Counterfactual trace automaton $\mathcal{A}_\pi^{\mathcal{C}}$.

component automata, and then construct a counterfactual trace automaton (cf. Definition 6) for each component, where the former intervenes on the events in a given cause that refer to the specific component. Afterward, each counterfactual trace automaton is intersected with its corresponding component automaton, and the network of all these intersections describes the counterfactual runs after intervention. To apply interventions locally, we start by defining the local projections of a run in a network of timed automata.

▶ **Definition 8** (Local Projection). *For a network $\mathcal{A}^n = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ and one of its runs ρ, we denote $\{j_1, \ldots, j_l\} := \{\, j \in \mathbb{N} \mid \mathit{participates}(\mathcal{A}_i, \alpha^\rho_j)\}$ as the the event points of some component automaton $\mathcal{A}_i$, whereby we let $j_1 < \ldots < j_l$. Then the local projection $\rho(\mathcal{A}_i)$ of the component automaton $\mathcal{A}_i$ is defined as the trace $\rho(\mathcal{A}_i) := \langle \delta_1, \alpha_1 \rangle \langle \delta_2, \alpha_2 \rangle \ldots$, in which*

- *$\alpha_k^{\rho(\mathcal{A}_i)} := \mathit{action}(\mathcal{A}_i, \alpha^\rho_{j_k})$ for all $k = 1, 2, \ldots$, i.e., the identity of the actions is preserved;*
- *$\delta_k^{\rho(\mathcal{A}_i)} = \Sigma_{x=j_{k-1}+1,\ldots,j_k}\, \delta^\rho_x$ for all $k = 1, 2, \ldots$ and with $j_0 := 1$, i.e., the delays in the local projection are the cumulative delays between two actions of the automaton in the global run of the network.*

Furthermore, we denote with $\mathit{locations}(\rho, \mathcal{A}_i) := (q^\rho_0)_i, (q^\rho_{j_1})_i, (q^\rho_{j_2})_i, \ldots$ the sequence of local locations, i.e., the projection to the i-th component of the network location. We define the localization function as $\mathit{localize}(\rho, \mathcal{A}^n) := (\rho(\mathcal{A}_1), \ldots, \rho(\mathcal{A}_n))$.

Note that the local projection as defined here differs fundamentally from local runs as defined for the local time semantics of timed automata [12], as the clocks still advance globally at the same speed. However, by conducting counterfactual interventions on the delays in a local projection of a run, we are able to change the order of transitions, which is not possible by interacting with delays in the global run of the network. It should also be noted that even if the global run ρ is infinite, the local projections may still turn out to be finite because the transitions occurring infinitely often may stem from a subset of the automata.

▶ **Example 9.** For the run ρ from Subsection 1.1, the first local projection $\rho(\mathcal{A}_1)$ is exactly the trace π considered in Example 7 and $\mathit{locations}(\rho, \mathcal{A}_1) = \mathsf{init}, \mathsf{crit}, (\mathsf{init})^\omega$ as the sequence of local locations. The second local projection $\rho(\mathcal{A}_2)$ is the finite trace $\rho(\mathcal{A}_2) = \langle 2.0, \beta \rangle \langle 3.0, \beta \rangle$.

It is worth pointing out that every global run induces well-defined local projections, however, the tuple $\mathit{localize}(\rho, \mathcal{A})$ of local traces may have multiple associated global runs. This stems from nondeterminism in the order of actions happening at the same timepoint. In essence, we treat the scheduler's decisions in such a situation as nondeterministic, and allow different resolution of this nondeterminism in counterfactual runs of the network.

▶ **Proposition 10.** *The localization function is not injective: There exists a network $\mathcal{A}$ and two runs $\rho \neq \rho'$ such that $\mathit{localize}(\rho, \mathcal{A}) = \mathit{localize}(\rho', \mathcal{A})$.*

Since we want to apply the construction of the counterfactual trace automaton locally to every component, we lift the definition of events from traces to (network) runs, such that the events of a network run are the union of events on local projections of the run.

▶ **Definition 11** (Events of a Network Run). *Given a run ρ of a network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, we define the set of associated events as*

$$\mathcal{E}_\rho := \{\, (e, i, \mathcal{A}_k) \mid (e, i) \in \mathcal{E}_{\rho(\mathcal{A}_k)} \text{ for some component } 1 \leq k \leq n \,\} \ .$$

We lift the set of all events to a network $\mathcal{A}^n$ and define it as $\mathcal{E}(\mathcal{A}^n) = \{(e, i, \mathcal{A}_k) \mid (e, i) \in \mathcal{E} \wedge 1 \leq k \leq n\}$ and say a run ρ satisfies a set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$, denoted $\rho \models \mathcal{C}$, if $\mathcal{C}$ is a subset of the events on ρ, i.e., if $\mathcal{C} \subseteq \mathcal{E}_\rho$. We further define an operator to filter for events of a specific component k: $\mathcal{C}|_k := \{(e, i) \mid (e, i, \mathcal{A}_k) \in \mathcal{C}\}$.

Note that $\mathcal{E}_{\rho(\mathcal{A}_k)}$ contains both action events as well as *locally projected* delay events. Hence, when we speak about the events on a network run we talk about the actions of the respective component automata (identified through the third position in the event tuple), as well as about the time between these actions of a component automaton (i.e., the cumulative delays between two actions of a component). These events are the atomic building blocks of our counterfactual expalantions. With this at hand we can define our first notion of counterfactual causality based on allowing arbitrary alternatives for all the events appearing in a hypothetical cause. The corresponding notion for structural equation models was termed *but-for* causality by Halpern [34], so we adopt the same name here. Crucially, in our setting with networks of timed automata, the alternative delays and events are realized with respect to the local projections of the network run, such that an alternative delay can change the order of actions emerging in different component automata.

▶ **Definition 12** (But-For Causality in Real-Time Systems). *Let $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ be a network of timed automata and ρ a run of the network. A set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$ is a but-for cause for ϕ in ρ of $\mathcal{A}$, if the following three conditions hold:*

SAT *$\rho \models \mathcal{C}$ and $\rho \models \phi$, i.e., cause and effect are satisfied by the actual run.*

$\mathbf{CF_{BF}}$ *There is an intervention on the events in $\mathcal{C}$ s.t. the resulting run avoids the effect ϕ, i.e., we have*

$$(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi \ .$$

MIN *$\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies* **SAT** *and* $\mathbf{CF_{BF}}$.

▶ **Example 13.** Consider again the system and run from Subsection 1.1, and the cause $\mathcal{C} = \{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$, i.e., the two β-actions of the first component (Cause 3a). **SAT** is satisfied, since the effect $\neg\Box_{[0,\infty)}(\neg \mathsf{crit}_1 \vee \neg \mathsf{crit}_2)$, i.e., the negation of the MITL specification is satisfied and the local projection of $\mathcal{A}_1$ is $\rho(\mathcal{A}_1) = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$. For $\mathbf{CF_{BF}}$, consider that the network run emerging from setting $\mathcal{A}_1$'s local projection to $\langle 1.0, \alpha\rangle\langle 3.0, \alpha\rangle(\langle 2.0, \alpha\rangle)^\omega$ does not violate the specification since the first component never enters crit_1. To see that $\mathcal{C}$ also satisfies **MIN** consider its two singleton subsets. Setting the alternative α for either of the actions alone does not suffice to avoid the effect due to the temporal ordering of the β-actions, e.g., when intervening only on the first β, then the second β enters crit_1 while the second component is also in its critical section, hence the effect is still present. Similarly, we can show $\{2.0, 1, \mathcal{A}_2\}$, i.e., the first delay of the second component (Cause 2), as well as all the other but-for causes from Table 1 to be but-for causes for ϕ in ρ.

Besides this intuitive example, we can prove several sanity properties about but-for causality. These properties concern the existence and identity of causes in certain distinctive cases. First up, we show that the existence of a but-for cause is guaranteed as long as a system run avoiding the effect exists.

▶ **Proposition 14.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, then for every run ρ of the network in which ϕ appears, there is a but-for cause for ϕ in ρ of $\mathcal{A}^n$, if and only if there exists a run ρ' of the network with $\rho' \not\models \phi$.*

Next, we consider the case where there is nondeterminism on the actual run, i.e., when there is another run with the same trace, that does no satisfy the effect. In this case, our definition returns the empty set as a unique actual cause.

▶ **Proposition 15.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n$, $\emptyset$ is the (unique) but-for cause for an effect ϕ on a run ρ of $\mathcal{A}^n$, if and only if there exists a run η of $\mathcal{A}^n$ with $localize(\rho, \mathcal{A}^n) = localize(\eta, \mathcal{A}^n)$ and $\eta \not\models \phi$, i.e., a run with the same local traces as the actual run, that does, however, not satisfy the effect.*

From a philosophical point of view, the empty set is a desirable verdict: It conveys that the smallest change necessary to avoid the effect does not consist of any changes of delay or action events, instead simply an alternative resolution of the underlying nondeterminism of this trace suffices to obtain a witnessing counterfactual run. Also from a practical perspective, it is helpful to know that the empty set gets returned only in this distinguishable scenario.

3.3 Contingencies in Networks of Timed Automata

Actual causality employs a *contingency* mechanism to isolate the true cause in the case of preemption. The key idea of contingencies to overcome this preemption is to reset certain propositions in counterfactual executions to their value as it is in the actual world. Coenen et al. [20] have outlined how to model contingencies for lasso-shaped traces of a Moore machine. We now describe a construction that applies this idea to networks of timed automata. The central idea is that the state resets resulting from applying a contingency now do not only reset the discrete machine state, but the clock assignment and the location of the timed automaton, i.e., the full underlying state. However, a central issue in networks of timed automata is that clocks are global variables shared by all component automata of the network, while the location is a local attribute of single components. We respect this dichotomy by allowing location contingencies only by actions of the corresponding component automaton and clock contingencies by any action in the global network. This is realized by two automata constructions, i.e., a local one applied to all component automata (for resetting locations) and a global one applied to the full network (for resetting clocks). In both cases, we model the behavior as an updatable timed automaton, as we outline in the following.

▶ **Definition 16** (Location Contingency Automaton). *Let ρ be a lasso-shaped run of a network and the timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$ a component of this network. The location contingency automaton of $\mathcal{A}$ and ρ is defined as $\mathcal{A}^{\mathsf{loc}(\rho)} := (Q', q_0', X, E', I', L')$ with $Q' := Q \times \{0, \dots, |\mathit{locations}(\rho, \mathcal{A})| - 1\}$, $q_0' := \langle q_0, 0\rangle$, $I'(\langle q, i\rangle) := I(q)$, $L'(\langle q, i\rangle) := L(q)$, and E' is defined as follows, where $\pi = \mathit{localize}(\rho, \mathcal{A})$.*

$$\frac{(q, g, \alpha, U, q') \in E \quad i = 1, \dots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q', \mathit{dst}_\pi(i)\rangle) \in E'} \qquad \frac{(q, g, \alpha, U, q') \in E \quad i = 1, \dots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q_j^\pi, \mathit{dst}_\pi(i)\rangle) \in E'}$$

The location contingency automaton $\mathcal{A}_\rho^{\mathsf{loc}}$ hence consists of copies of the original system, one for each position in the lasso-shaped local projection π. With an action transition, it moves from one copy into the next, either following the edge (q, g, α, U, q') of the original system (left rule in Definition 16) or moving to the same location q_j^π as present in π at the respective position $\mathit{dst}_\pi(i)$ by applying a contingency (right rule in Definition 16). Note that after the end of the loop in the lasso-shaped projection, the transitions are redirected to the copy corresponding to the initial position of the loop by the definition of the function dst_π. The same principle can now also be applied to global variables. In our setting, this only concerns clocks, but the following definition of the clock contingency automaton can be generalized to all global variables such as integers, if these are included in the system model.

▶ **Definition 17** (Clock Contingency Automaton). *Let ρ be a lasso-shaped run of a timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$. The clock contingency automaton of $\mathcal{A}$ and ρ is defined as $\mathcal{A}^{\mathsf{clk}(\rho)} := (Q', q_0', X, E', I', L')$ with $Q' := Q \times \{0, \dots, |\rho| - 1\}$, $q_0' := \langle q_0, 0\rangle$, $I'(\langle q, i\rangle) := I(q)$, $L'(\langle q, i\rangle) := L(q)$, and E' is defined as follows.*

$$\frac{(q, g, \alpha, U, q') \in E \quad i = 1, \dots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q', \mathit{dst}_\rho(i)\rangle) \in E'} \qquad \frac{(q, g, \alpha, U, q') \in E \quad i = 1, \dots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, u_j^\rho, \langle q', \mathit{dst}_\rho(i)\rangle) \in E'}$$

Note that strictly speaking we have defined clock updates to values in $\mathbb{Q}$, instead of $\mathbb{N}$ as considered in classic decidability results. It is, however, straightforward to scale these values to the natural numbers [3]. Clearly, the signals modeled by the contingency automata subsume the ones by the original automaton, because it is possible to simply never invoke a contingency and, hence, always follow the dynamics of the original system.

▶ **Proposition 18.** *For all timed automata $\mathcal{A}$ and runs ρ of $\mathcal{A}$, we have that the languages of the contingency automata subsume the language of the original automaton: $\mathcal{L}(\mathcal{A}_\rho^{\mathsf{loc}}) \supseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}_\rho^{\mathsf{clk}}) \supseteq \mathcal{L}(\mathcal{A})$.*

▶ **Definition 19** (Actual Causality in Real-Time Systems)**.** *Let $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ be a network of timed automata and ρ a run of the network. A set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$ is an actual cause for ϕ in ρ of $\mathcal{A}$, if the following three conditions hold:*

SAT *$\rho \models \mathcal{C}$ and $\rho \models \phi$, i.e., cause and effect are satisfied by the actual run.*

$\mathbf{CF}_{\mathsf{Act}}$ *There is an intervention on the events in $\mathcal{C}$ and a location and clock contingency (denoted by $\mathsf{loc}(\rho)$ and $\mathsf{clk}(\rho)$ resp.) s.t. the resulting run avoids the effect ϕ, i.e., we have*

$$\left((\mathcal{A}_1^{\mathsf{loc}(\rho)} \cap \mathcal{A}_{\rho(\mathcal{A}_1)}^{\mathcal{C}|_1}) \,||\, \ldots \,||\, (\mathcal{A}_n^{\mathsf{loc}(\rho)} \cap \mathcal{A}_{\rho(\mathcal{A}_n)}^{\mathcal{C}|_n})\right)^{\mathsf{clk}(\rho)} \not\models \phi \ .$$

MIN *$\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies* **SAT** *and* $\boldsymbol{CF}_{\mathsf{Act}}$.

▶ **Example 20.** Consider the but-for cause $\mathcal{C} = \{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$ from Example 13. This $\mathcal{C}$ is not an *actual* cause because it does not satisfy the **MIN** condition of Definition 19: The subset $\mathcal{C}' = \{(\beta, 1, \mathcal{A}_1)\}$ satisfies **SAT** and $\mathbf{CF}_{\mathsf{Act}}$. For $\mathbf{CF}_{\mathsf{Act}}$ we can use contingencies to neutralize the effect of the second β in the local projection $\rho(\mathcal{A}_1) = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$. Since this action moves to init in the original run (cf. Subsection 1.1), it can also move to this location in the contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$. Hence we find an intervention (setting $\langle 1.0, \boldsymbol{\beta}\rangle$ to $\langle 1.0, \boldsymbol{\alpha}\rangle$) and a contingency (setting the target location of $\langle 3.0, \beta\rangle$ to init) that avoid the effect together. A more detailed construction of the contingency automaton is given in Appendix C. In fact, $\mathcal{C}' = \{(\beta, 1, \mathcal{A}_1)\}$ is an actual cause (Cause 3) since additionally to **SAT** and $\mathbf{CF}_{\mathsf{Act}}$ it also satisfies **MIN** – the empty set does not satisfy $\mathbf{CF}_{\mathsf{Act}}$. Again, also all the other actual causes from Table 1 can be shown to fulfill our definition.

▶ Remark 21. Note that as a consequence of Proposition 18, the statements regarding the existence and identity of causes as proven in Propositions 14 and 15 can be lifted to actual causality, but require replacing the original networks in the equivalence statements by the contingency automata construction used in $\mathbf{CF}_{\mathsf{Act}}$ (cf. Definition 19).

4 Computing Counterfactual Causes

In this section, we develop algorithms for computing but-for and actual causes for any MITL effect. Proofs and further details related to this section can be found in Appendix B. We only explicitly present the algorithm for but-for causes; for actual causes the central model checking query needs to be substituted (cf. Definition 19, Lines 4 and 10 in Algorithm 1).

In principle, the algorithms are based on enumerating all candidate causes. However, we can speed up this process significantly by utilizing what we term the monotonicity of causes.

▶ **Lemma 22** (Cause Monotonicity)**.** *For every network of timed automaton $\mathcal{A}$, run ρ, and effect ϕ, we have that*

1. *if a set of events $\mathcal{C}$ fulfills* **SAT** *also every subset $\mathcal{C}' \subseteq \mathcal{C}$ fulfills* **SAT**.
2. *if a set of events $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$) also every superset $\mathcal{C}' \supseteq \mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$).*

Algorithm 1 Compute But-For Causes.

Input: network $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, run ρ of $\mathcal{A}$ satisfying effect ϕ, i.e. $\rho \models \phi$
Output: set of all but-for causes for ϕ in ρ of $\mathcal{A}$

$Res_s := \{\}, Res_l := \{\}, Power := \mathcal{P}(\mathcal{E}_\rho)$
for $i = 0, 1, 2, \ldots, \frac{|\mathcal{E}_\rho|}{2}$ **do**
  for $\mathcal{C} \in Power$ *with* $|\mathcal{C}| = i$*:* **do**
    if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ **then** `// cause found?`
      $Res_s := Res_s \cup \mathcal{C}$
      $Power := \{\mathcal{C}' \in Power \,|\, \mathcal{C} \not\subseteq \mathcal{C}'\}$; `// remove all supersets`
    end
  end
  for $\mathcal{C} \in Power$ *with* $|\mathcal{C}| = \frac{|\mathcal{E}_\rho|}{2} - i$*:* **do**
    if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ **then** `// cause found?`
      $Res_l := Res_l \cup \mathcal{C}$
    else
      $Power := \{\mathcal{C}' \in Power \,|\, \mathcal{C}' \not\subseteq \mathcal{C}\}$; `// remove all subsets`
    end
  end
end
return $Res_s \cup \{\mathcal{C} \in Res_l \,|\, \neg\exists\, \mathcal{C}' \subsetneq \mathcal{C}.\, \mathcal{C}' \in Res_s \cup Res_l\}$; `// filter` Res_l `for MIN`

The second monotonicity property enables efficient checking of the **MIN** condition, as it suffices to check only the subsets with one element less instead of checking all subsets of a potential cause. For the computation of causes on a given run ρ, a naive approach could now simply enumerate all elements of $\mathcal{P}(\mathcal{E}_\rho)$, that is, all subsets of the all events $\mathcal{E}_\rho$ on ρ, and check whether they form a cause. By further exploiting monotonicity properties, we can find a more efficient enumeration significantly accelerating the computation of causes. The key idea is to enumerate through the powerset $\mathcal{P}(\mathcal{E}_\rho)$ simultaneously from below (starting with the empty cause and then causes of increasing size) and above (starting with the full cause and then causes of decreasing size). This then allows to exclude certain parts of the powerset from the computation in two ways: First, when finding a set of events $\mathcal{C}$ fulfilling $\mathbf{CF}_{\mathsf{BF}}$, we can exclude all of its supersets as we know that they cannot satisfy **MIN**. Second, when finding a set of events $\mathcal{C}$ not fulfilling $\mathbf{CF}_{\mathsf{BF}}$, we can exclude all of its subsets as the monotonicity of $\mathbf{CF}_{\mathsf{BF}}$ implies that $\mathcal{C}' \subseteq \mathcal{C}$ will neither fulfill $\mathbf{CF}_{\mathsf{BF}}$. This idea of the simultaneous enumeration of $\mathcal{P}(\mathcal{E}_\rho)$ is implemented in Algorithm 1.

▶ **Theorem 23.** *Algorithm 1 is sound and complete, i.e., it terminates with*

$$\textit{Compute But-For Causes}(\mathcal{A}, \rho, \phi) = \{\, \mathcal{C} \,|\, \mathcal{C} \text{ is a but-for cause for } \phi \text{ in } \rho \text{ of } \mathcal{A} \,\}\ ,$$

for all networks $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ and runs ρ of $\mathcal{A}$ satisfying an effect ϕ.

While it is clear that our algorithm requires to solve several model checking problems for the effect ϕ, we can show that we cannot do better: Model checking some formula ϕ can be encoded as a cause checking problem. Hence, asymptotically, cause checking and computation scale similar to MITL model checking for the formula φ.

▶ **Theorem 24.** *Checking and computing causes for an effect ϕ on the run ρ in a network of timed automata $\mathcal{A}$ is* EXPSPACE(ϕ)*-complete.*

Table 2 Experimental results. n: number of automata in the network; $|\rho|$: run length; $|\mathcal{E}_\rho|$: number of events on the run; $\#\mathcal{C}$: number of but-for/actual causes; $|\mathcal{C}|$: *average* but-for/actual cause size; t: runtime for computing but-for/actual causes.

Instance	n	$\lvert\rho\rvert$	$\lvert\mathcal{E}_\rho\rvert$	$\#\mathcal{C}_{BF}$	$\#\mathcal{C}_{Act}$	$\lvert\mathcal{C}_{BF}\rvert$	$\lvert\mathcal{C}_{Act}\rvert$	t_{BF}	t_{Act}
Run. Ex.	2	5	11	6	5	1.83	1.2	5.67s	5.42s
		6	16	10	7	3.2	2	88.2s	128.8s
Run. Ex.	3	5	11	6	5	1.83	1.2	5.70s	5.53s
		7	16	6	6	1.5	1.17	55.0s	78.6s
Run. Ex.	4	9	19	8	7	1.625	1.14	279.3s	331.8s
Fischer	2	4	12	2	2	1	1	2.37s	9.73s
		7	20	5	5	1.2	1.2	273.1s	1499s
Fischer	3	5	14	2	2	1	1	3.40s	16.9s
		7	20	5	5	1.2	1.2	283.6s	1516s
Fischer	4	6	16	2	2	1	1	4.58s	28.8s
		7	20	5	5	1.2	1.2	295.3s	1535s

Note that this discussion on the complexity with respect to the size of the effect abstracts away from the, e.g., the length of the counterexample, which contributes polynomially to cause checking and exponentially to cause computation since we need to check all subsets of events. In practice, we have observed that the bidirectional enumeration of the powerset realized in Algorithm 1 significantly speeds up the compuation of causes.

5 Experimental Evaluation

We have implemented a prototype in Python.[1] For model checking networks of timed automata, we use Uppaal [13] and the library PyUppaal [56]. Our tool can check and compute causes for effects in the fragment of MITL that is supported by the Uppaal verification suite. We conducted experiments on the running example of this paper, as well as on Fischer's protocol, a popular benchmark for real-time model checking. The experiments were run on a MacBook Pro with an Apple M3 Max and 64GB of memory. The results can be found in Table 2. For the running example, the tool found exactly the causes depicted in Table 1; for Fischer's protocol, we report details in Appendix D. The computed causes narrow down the responsible behavior on a given execution, with the average size of the causes between 1 and 3.2 on execution with a large number of events ($|\mathcal{E}_\rho|$). Using contingencies does result in smaller causes (cf. $\boldsymbol{Avg.}\ |\mathcal{C}_{BF}|$ vs. $|\mathcal{C}_{Act}|$) on the running example. This is not the case for Fischer's protocol, where but-for and actual causes are identical. These findings suggest some directions for optimization, since computing but-for causes is more efficient than computing actual causes. Since the latter are always subsets of the former, it may be sensible to first compute but-for causes and then refine them by taking into account contingencies. Further, the times in Table 2 refer to the time to compute *all* causes. Hence, the performance in practical applications may be improved by iteratively presenting the user with (but-for or actual) causes that have already been found during the execution.

[1] Our prototype and benchmarks are available on GitHub [43].

6 Related Work

Providing explanatory insight into *why* a system does not satisfy a specification has been of growing interest in the verification community: Baier et al. [6] provide a recent and detailed survey. Most works focus on discrete systems and perform error localization in executions [9, 31, 59, 45] or by identifying responsible components [58, 29, 28, 60, 32, 5]. There are also several works on program slicing for analyzing dependencies between different parts of a program [61, 41, 38]. The concepts of vacuity and coverage can be used to gain causal insight also in the case of a successful verification [11, 8, 42, 18]. There are several recent works that take a state-based view of responsibility allocation in transition systems [7, 53], but they do not consider infinite state systems where such an approach is not directly applicable. There are several works [54, 21, 26] that use a notion of distance defined by similarity relations in the counterfactual tradition of Lewis [49]. These are more closely related to our work since the minimality criterion in our definitions of but-for and actual causality can be interpreted as a similarity relation [26]. Like this paper, a range of works has been inspired by Halpern and Pearl's actual causality for generating explanations [10, 22, 32, 20, 48, 57, 15]. Our contingency automata constructions are particularly inspired by Coenen et al. [20, 21]. There is a growing interest in counterfactual causality in models with infinitely many variables or infinite domains [26, 37]. In the latter work, Halpern and Peters provide an axiomatic account for counterfactual causes in such (structural equation) models, where variables are further allowed to have infinite ranges. Our results suggest that fragments of structural equation models related to networks of timed automata as studied here may be particularly amenable to cause computation. A correspondence between these modeling formalisms has already been pointed out by the same authors [55], albeit to the even more expressive hybrid automata [2] that subsume timed automata. For real-time systems, Dierks et al. develop an automated abstraction refinement technique [19] for timed automata based on considering causal relationships [24]. Wang et al. introduce a framework for the causal analysis of component-based real-time systems [60]. Kölbl et al. follow a similar direction and propose a repairing technique of timed systems focusing on static clock bounds [47]. In a further contribution, they consider the delay values of timed systems to compute causal delay values and ranges in traces violating reachability properties [46]. Mari et al. propose an explanation technique for the violation of safety properties in real-time systems [52], their approach is based on their corresponding work on explaining discrete systems [30]. Hence, in the domain of real-time systems ours is the first technique to consider arbitrary MITL properties, i.e., safety *and* liveness, as effects, together with both actions *and* real-time delays as causes.

7 Conclusion

Based on the seminal works of Halpern and Pearl, we have proposed notions of but-for and actual causality for networks of timed automata, which define counterfactual explanations for violations of MITL specifications. Our definitions rely on the idea of counterfactual automata that represent infinitely many possible counterfactual executions. We then leveraged results on real-time model checking for algorithms that check and compute but-for and counterfactual causes, demonstrating with a prototype that our explanations significantly narrow down the root causes in counterexamples of MITL properties. Interesting directions of future work are to study symbolic causes [48, 21, 25] in real-time system, i.e., to consider real-time properties specified in MITL or an event-based logic as causes [48, 17], and to develop tools for visualizing [40] counterfactual explanations in networks of timed automata.

References

1 Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999. doi:10.1007/3-540-48683-6_3.

2 Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992. doi:10.1007/3-540-57318-6_30.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

4 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, January 1996. doi:10.1145/227595.227602.

5 Uwe Aßmann, Christel Baier, Clemens Dubslaff, Dominik Grzelak, Simon Hanisch, Ardhi Putra Pratama Hartono, Stefan Köpsell, Tianfang Lin, and Thorsten Strufe. *Tactile computing: Essential building blocks for the Tactile Internet*, pages 293–317. Academic Press, 2021. 46.23.01; LK 01. doi:10.1016/B978-0-12-821343-8.00025-3.

6 Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Rupak Majumdar, Jakob Piribauer, and Robin Ziemek. From verification to causality-based explications (invited talk). In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.1.

7 Christel Baier, Roxane van den Bossche, Sascha Klüppelholz, Johannes Lehmann, and Jakob Piribauer. Backward responsibility in transition systems using general power indices. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 20320–20327. AAAI Press, 2024. doi:10.1609/AAAI.V38I18.30013.

8 Thomas Ball and Orna Kupferman. Vacuity in testing. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs*, pages 4–17, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-79124-9_2.

9 Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, pages 97–105, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/604131.604140.

10 Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard Trefler. Explaining counterexamples using causality. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 94–108, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-02658-4_11.

11 Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in actl formulas. In Orna Grumberg, editor, *Computer Aided Verification*, pages 279–290, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. doi:10.1007/3-540-63166-6_28.

12 Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998. doi:10.1007/BFB0055643.

13 Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL – A tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995. doi:10.1007/BFB0020949.

14 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3.

15 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Julian Siber. Checking and sketching causes on temporal sequences. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II*, volume 14216 of *Lecture Notes in Computer Science*, pages 314–327. Springer, 2023. doi:10.1007/978-3-031-45332-8_18.

16 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004. doi:10.1016/J.TCS.2004.04.003.

17 Georgiana Caltais, Sophie Linnea Guetlein, and Stefan Leue. Causality for general LTL-definable properties. *Electronic Proceedings in Theoretical Computer Science*, 286:1–15, January 2019. doi:10.4204/eptcs.286.1.

18 Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Logic*, 9(3), June 2008. doi:10.1145/1352582.1352588.

19 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/10722167_15.

20 Norine Coenen, Raimund Dachselt, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Tom Horak, Niklas Metzger, and Julian Siber. Explaining hyperproperty violations. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 407–429. Springer, 2022. doi:10.1007/978-3-031-13185-1_20.

21 Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger, and Julian Siber. Temporal causality in reactive systems. In Ahmed Bouajjani, Lukás Holík, and Zhilin Wu, editors, *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings*, volume 13505 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2022. doi:10.1007/978-3-031-19992-9_13.

22 Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. Program actions as actual causes: A building block for accountability. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 261–275, 2015. doi:10.1109/CSF.2015.25.

23 Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *12th Euromicro Conference on Real-Time Systems (ECRTS 2000), 19-21 June 2000, Stockholm, Sweden, Proceedings*, pages 165–172. IEEE Computer Society, 2000. doi:10.1109/EMRTS.2000.854004.

24 Henning Dierks, Sebastian Kupferschmid, and Kim Guldstrand Larsen. Automatic abstraction refinement for timed automata. In Jean-François Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, volume 4763 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2007. doi:10.1007/978-3-540-75454-1_10.

25 Bernd Finkbeiner, Hadar Frenkel, Niklas Metzger, and Julian Siber. Synthesis of temporal causality. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III*, volume 14683 of *Lecture Notes in Computer Science*, pages 87–111. Springer, 2024. doi:10.1007/978-3-031-65633-0_5.

26 Bernd Finkbeiner and Julian Siber. Counterfactuals modulo temporal logics. In Ruzica Piskac and Andrei Voronkov, editors, *LPAR 2023: 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, June 4-9, 2023*, volume 94 of *EPiC Series in Computing*, pages 181–204. EasyChair, 2023. doi:10.29007/qtw7.

27 Michael Gerke, Rüdiger Ehlers, Bernd Finkbeiner, and Hans-Jörg Peter. Model checking the flexray physical layer protocol. In Stefan Kowalewski and Marco Roveri, editors, *Formal Methods for Industrial Critical Systems - 15th International Workshop, FMICS 2010, Antwerp, Belgium, September 20-21, 2010. Proceedings*, volume 6371 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2010. doi:10.1007/978-3-642-15898-8_9.

28 Gregor Gössler and Daniel Le Métayer. A General Trace-Based Framework of Logical Causality. Research Report RR-8378, INRIA, October 2013. URL: https://inria.hal.science/hal-00873665.

29 Gregor Gössler, Daniel Le Métayer, and Jean-Baptiste Raclet. Causality analysis in contract violation. In Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Roşu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification*, pages 270–284, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

30 Gregor Gössler, Thomas Mari, Yannick Pencolé, and Louise Travé-Massuyès. Towards Causal Explanations of Property Violations in Discrete Event Systems. In *DX'19 - 30th International Workshop on Principles of Diagnosis*, pages 1–8, Klagenfurt, Austria, November 2019. URL: https://inria.hal.science/hal-02369014.

31 Alex Groce. Error explanation with distance metrics. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 108–122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-24730-2_8.

32 Gregor Gössler and Jean-Bernard Stefani. Causality analysis and fault ascription in component-based systems. *Theoretical Computer Science*, 837:158–180, 2020. doi:10.1016/j.tcs.2020.06.010.

33 Joseph Y. Halpern. A modification of the halpern-pearl definition of causality. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3022–3033. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/427.

34 Joseph Y. Halpern. *Actual Causality*. MIT Press, 2016.

35 Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British Journal for the Philosophy of Science*, 2005.

36 Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British Journal for the Philosophy of Science*, 2005.

37 Joseph Y. Halpern and Spencer Peters. Reasoning about causal models with infinitely many variables. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5668–5675, June 2022. doi:10.1609/aaai.v36i5.20508.

38 Mark Harman and Robert M. Hierons. An overview of program slicing. *Softw. Focus*, 2(3):85–92, 2001. doi:10.1002/SWF.41.

39 K. Havelund, A. Skou, K.G. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. In *Proceedings Real-Time Systems Symposium*, pages 2–13, 1997. doi:10.1109/REAL.1997.641264.

40 Tom Horak, Norine Coenen, Niklas Metzger, Christopher Hahn, Tamara Flemisch, Julián Méndez, Dennis Dimov, Bernd Finkbeiner, and Raimund Dachselt. Visual analysis of hyperproperties for understanding model checking results. *IEEE Trans. Vis. Comput. Graph.*, 28(1):357–367, 2022. doi:10.1109/TVCG.2021.3114866.

41 Susan B. Horwitz, Thomas Reps, and Dave Binkley. Interprocedural slicing using dependence graphs. *SIGPLAN Not.*, 23(7):35–46, June 1988. doi:10.1145/960116.53994.

42 Yatin Vasant Hoskote, Timothy Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage estimation for symbolic model checking. In Mary Jane Irwin, editor, *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999*, pages 300–305. ACM Press, 1999. doi:10.1145/309847.309936.

43 Felix Jahn. Prototype tool of our approach. URL: https://github.com/FelixJahnFJ/Real-Time-Causality-Tool.git.

44 Felix Jahn. Real Time Causality Analysis Tool. Software, swhId: swh:1:dir:de6b34eb1137d85c4257b5adac4b15646bd8ea3e (visited on 2024-10-11). URL: https://github.com/reactive-systems/rt-causality.git.

45 Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In Mary W. Hall and David A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 437–446. ACM, 2011. doi:10.1145/1993498.1993550.

46 Martin Kölbl, Stefan Leue, and Robert Schmid. Dynamic causes for the violation of timed reachability properties. In Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*, volume 12288 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2020. doi:10.1007/978-3-030-57628-8_8.

47 Martin Kölbl, Stefan Leue, and Thomas Wies. Clock bound repair for timed systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2019. doi:10.1007/978-3-030-25540-4_5.

48 Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013. doi:10.1007/978-3-642-35873-9_16.

49 David K. Lewis. *Counterfactuals*. Cambridge, MA, USA: Blackwell, 1973.

50 Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. In Bernhard Steffen, editor, *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1384 of *Lecture Notes in Computer Science*, pages 281–297. Springer, 1998. doi:10.1007/BFB0054178.

51 Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2006. doi:10.1007/11867340_20.

52 Thomas Marix, Thao Dang, and Gregor Gössler. Explaining safety violations in real-time systems. In Catalin Dima and Mahsa Shirmohammadi, editors, *Formal Modeling and Analysis of Timed Systems - 19th International Conference, FORMATS 2021, Paris, France, August 24-26, 2021, Proceedings*, volume 12860 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2021. doi:10.1007/978-3-030-85037-1_7.

53 Corto Mascle, Christel Baier, Florian Funke, Simon Jantsch, and Stefan Kiefer. Responsibility and verification: Importance value in temporal logics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470597.

54 Julie Parreaux, Jakob Piribauer, and Christel Baier. Counterfactual causality for reachability and safety based on distance functions. In Antonis Achilleos and Dario Della Monica, editors, *Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023*, volume 390 of *EPTCS*, pages 132–149, 2023. doi:10.4204/EPTCS.390.9.

55 Spencer Peters and Joseph Y. Halpern. Causal modeling with infinitely many variables. *CoRR*, abs/2112.09171, 2021. arXiv:2112.09171.

56 Pyuppaal library webpage. URL: https://pypi.org/project/pyuppaal/1.0.0/.

57 Arshia Rafieioskouei and Borzoo Bonakdarpour. Efficient discovery of actual causality using abstraction-refinement. *CoRR*, abs/2407.16629, 2024. doi:10.48550/arXiv.2407.16629.

58 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

59 Chao Wang, Zijiang Yang, Franjo Ivančić, and Aarti Gupta. Whodunit? causal analysis for counterexamples. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis*, pages 82–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

60 Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor Gössler, Oleg Sokolsky, and Insup Lee. A causality analysis framework for component-based real-time systems. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, pages 285–303, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40787-1_17.

61 Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984. doi:10.1109/TSE.1984.5010248.

A Proofs of Section 3

▶ **Proposition 14.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n = \mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$, then for every run ρ of the network in which ϕ appears, there is a but-for cause for ϕ in ρ of $\mathcal{A}^n$, if and only if there exists a run ρ' of the network with $\rho' \not\models \phi$.*

Proof. Let ρ be a run of such a network with $\rho \models \phi$. We show both direction of the equivalence separately.

"⇒": Assume there is a but-for cause $\mathcal{C}$ for ϕ in ρ of $\mathcal{A}$. From $\mathbf{CF}_{\mathsf{BF}}$, we know that there exists a run $\rho' \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})\big)$ such that $\rho' \not\models \phi$. Since the components of the network are built from (trace) intersections, is easy to see that $\Pi(\mathcal{A}_i \cap \mathcal{A}^{\mathcal{C}}_{\rho(\mathcal{A}_i)}) \subseteq \Pi(\mathcal{A}_i)$ for all components $1 \leq i \leq n$. From the semantics of the network based on parallel composition, it follows that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})\big) \subseteq \Pi(\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n)$, from which this direction of the claim immediately follows.

"⇐": Let ρ' be a run of the network $\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$ with $\rho' \not\models \phi$. We show that the set of events $\mathcal{E}_\rho$, i.e., the set of *all* events appearing on the path ρ, fulfills the **SAT** and the $\mathbf{CF}_{\mathsf{BF}}$ condition: From our initial assumption, it follows that $\rho \models \phi$ and from the definition of $\mathcal{E}_\rho$ we have $\rho \models \mathcal{E}_\rho$, hence the **SAT** condition is fulfilled. From the definition of the counterfactual trace automaton, it follows that the language $\mathcal{A}^{\mathcal{E}_\rho|_i}_{\rho(\mathcal{A}_i)}$ of every component i describes all possible traces, i.e., arbitrary orderings of actions, with arbitrary delays, over the alphabet of actions Act. From this we can deduce that the runs of the network under arbitrary interventions are in fact the runs of the original network, i.e., we have $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{E}_\rho|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{E}_\rho|_n}_{\rho(\mathcal{A}_n)})\big) = \Pi(\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n)$. Since by our initial assumption there exists a $\rho' \not\models \phi$ in $\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$, we can deduce that $\mathbf{CF}_{\mathsf{BF}}$ is fulfilled. Finally, since $\mathcal{E}_\rho$ is finite, it either has a minimal subset that satisfies the two criteria and hence witnesses this direction of our claim, or $\mathcal{E}_\rho$ itself is the desired witness. ◀

▶ **Proposition 15.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n$, $\emptyset$ is the (unique) but-for cause for an effect ϕ on a run ρ of $\mathcal{A}^n$, if and only if there exists a run η of $\mathcal{A}^n$ with $localize(\rho, \mathcal{A}^n) = localize(\eta, \mathcal{A}^n)$ and $\eta \not\models \phi$, i.e., a run with the same local traces as the actual run, that does, however, not satisfy the effect.*

Proof. Let a network $\mathcal{A}^n = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ be given. First up, it is easy to see that whenever $\emptyset$ is a but-for cause, it is unique: No other set $\mathcal{C} \neq \emptyset$ can satisfy **MIN**, since $\emptyset \subset \mathcal{C}$ and $\emptyset$ satisfies **SAT** and $\mathbf{CF}_{\mathsf{BF}}$ by assumption. We proceed with proving the equivalence:

"⇒": Assume that $\emptyset$ is a but-for cause on some run ρ, then from $\mathbf{CF}_{\mathsf{BF}}$ it follows that there exists a run $\rho' \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ such that $\rho' \not\models \phi$. From the definition of the counterfactual trace automaton $\mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)}$ it follows that for all components i and for all $\rho_i \in \Pi(\mathcal{A}_i \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)})$ we have that $\pi^{\rho_i} = \rho(\mathcal{A}_i)$. From the definition of the localization function it then follows that for all $\zeta \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ we have that $localize(\rho, \mathcal{A}) = localize(\zeta, \mathcal{A})$, so in particular for ρ', which shows this direction of the claim.

"⇐": Assume there is such an η with $localize(\rho, \mathcal{A}) = localize(\eta, \mathcal{A})$ and $\eta \not\models \phi$. It is easy to see that $\emptyset$ trivially satisfies **SAT** and **MIN**. Hence, we only need to show that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ includes η (and indeed all runs with the same local traces as ρ). This follows from the fact that $\Pi(\mathcal{A}_i \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)})$ includes all runs η_i that have the same trace as the local projection of ρ with respect to this component, i.e., all $\eta_i = \rho(\mathcal{A}_i)$, due to the definition of $\mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)}$ and of trace intersection. By the definition of parallel composition, we can conclude that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ includes all ρ' with $localize(\rho, \mathcal{A}) = localize(\rho', \mathcal{A})$, hence it also includes η, which can then serve as a witness for $\emptyset$ satisfying $\mathbf{CF}_{\mathsf{BF}}$, which closes this direction of the equivalence. ◀

B Algorithms and Proofs of Section 4

In this section, we give the algorithm for checking causality and detailed proofs of the statements from Section 4. We start by proving the monotonicity properties.

▶ **Lemma 22** (Cause Monotonicity). *For every network of timed automaton $\mathcal{A}_1 \,||\, \ldots \,||\, TA_n$, run ρ, and effect ϕ, we have that*

1. *if a set of events $\mathcal{C}$ fulfills* **SAT** *also every subset $\mathcal{C}' \subseteq \mathcal{C}$ fulfills* **SAT**.
2. *if a set of events $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$) also every superset $\mathcal{C}' \supseteq \mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$).*

Proof. We show the two statements separately:

1. Follows by the transitivity of set inclusions: If $\mathcal{C}$ fulfills **SAT**, we have that $\rho \models \mathcal{C}$ and $\rho \models E$. Hence, $\mathcal{C}' \subseteq \mathcal{C} \subseteq \mathcal{E}_\rho$ and therefore $\rho \models \mathcal{C}'$ such that also $\mathcal{C}'$ fulfills **SAT**.
2. Let $\mathcal{C}$ fulfill $\mathbf{CF}_{\mathsf{BF}}$, that is, there is a counterfactual run ρ' of $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})$ with $\rho \not\models \phi$. Now notice that for $\mathcal{C}' \supseteq \mathcal{C}$, also the transition relation of each counterfactual trace automaton of $\mathcal{C}'$ is a superset of the one of $\mathcal{C}$ such that we also have $\Pi(\mathcal{A}^{\mathcal{C}'|_i}_{\rho(\mathcal{A}_i)}) \supseteq \Pi(\mathcal{A}^{\mathcal{C}|_i}_{\rho(\mathcal{A}_i)})$. Therefore, ρ' is also a run of $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}'|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}'|_n}_{\rho(\mathcal{A}_n)})$ such that $\mathcal{C}'$ fulfills $\mathbf{CF}_{\mathsf{BF}}$. The proof for $\mathbf{CF}_{\mathsf{Act}}$ works analogously for the run in intersection of the contingency and counterfactual trace automata. ◀

Algorithm 2 decides whether a given set of events forms a but-for cause. It is a straightforward implementation of Definition 12 of but-for causality under the use of monotonicity for accelerating the verification of the **MIN** condition. Hence, we do not give a detailed proof of correctness for Algorithm 2 and continue directly with cause computation.

Algorithm 2 Checking But-For Cause.

Input: network $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, run ρ, effect ϕ, set of events $\mathcal{C}$
Output: "Is $\mathcal{C}$ a but-for cause for ϕ in ρ of $\mathcal{A}$?"

1 **if** $\rho \not\models \phi$ ***or*** $\mathcal{C} \not\subseteq \mathcal{E}_\rho$ **then** `// checking SAT`
2 | **return false**
3 **end**
4 **if** $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \models \phi$ **then** `// checking` $\mathbf{CF}_{\mathsf{BF}}$
5 | **return false**
6 **end**
7 **for** *event* $e \in \mathcal{C}$ **do** `// checking MIN`
8 | $C' := \mathcal{C} \setminus \{e\}$
9 | **if** $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}'|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}'|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ **then**
10 | | **return false**
11 | **end**
12 **end**
13 **return true**

▶ **Theorem 23.** *Algorithm 1 is sound and complete, i.e., it terminates with*

$$\textit{Compute But-For Causes}(\mathcal{A}, \rho, \phi) = \{\mathcal{C} \mid \mathcal{C} \textit{ is a but-for cause for } \phi \textit{ in } \rho \textit{ of } \mathcal{A}\},$$

for all networks $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ and runs ρ of $\mathcal{A}$ satisfying an effect ϕ.

Proof. We argue for soundness ($\subseteq$) and completeness ($\supseteq$) separately:

"$\supseteq$": Let $\mathcal{C}$ be a but-for cause for ϕ in ρ of $\mathcal{A}$, i.e. fulfilling **SAT**, $\mathbf{CF}_{\mathsf{BF}}$, **MIN**. We first notice that the algorithm does then not remove $\mathcal{C}$ from *Power* (until it may be added to Res_s): $\mathcal{C}$ is not removed by Line 6 since the minimality of $\mathcal{C}$ implies that it has no subset fulfilling $\mathbf{CF}_{\mathsf{BF}}$; and $\mathcal{C}$ is not removed by Line 13 since the monotonicity of $\mathbf{CF}_{\mathsf{BF}}$ implies that it has no superset not fulfilling $\mathbf{CF}_{\mathsf{BF}}$. Now since $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$, if $|\mathcal{C}| \leq \frac{\mathcal{E}_\rho}{2}$, $\mathcal{C}$ is added to Res_s in Line 5, if $|\mathcal{C}| > \frac{\mathcal{E}_\rho}{2}$ it is added to Res_l in Line 11 and is, in addition, not removed in the last line as $\mathcal{C}$ fulfills the **MIN** condition. Therefore, $\mathcal{C}$ is returned by $\mathsf{Compute\ But\text{-}For\ Causes}(\mathcal{A}, \rho, \phi)$.

"$\subseteq$": Let $\mathcal{C} \in \mathsf{Compute\ But\text{-}For\ Causes}(\mathcal{A}, \rho, \phi)$. As for all set of events considered by the algorithm, we have $\mathcal{C} \in \mathcal{P}(\mathcal{E}_\rho)$ and, hence, $\mathcal{C} \subseteq \mathcal{E}_\rho$ such that $\mathcal{C}$ fulfills **SAT**. By definition of the algorithm, $\mathcal{C}$ is only returned as a result when it was added to Res_s or Res_l. This, in turn, is only the case, if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$. Therefore, $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$. Lastly to establish the **MIN** condition, we have to show that there are no proper subsets of $\mathcal{C}$ that fulfill **SAT** and $\mathbf{CF}_{\mathsf{BF}}$. Towards a contradiction, lets assume there are such subsets and let $\mathcal{C}' \subsetneq \mathcal{C}$ be the minimal one. Then, $\mathcal{C}'$ is but-for cause and by the first inclusion $\mathcal{C}' \in \mathsf{Compute\ But\text{-}For\ Causes}(\mathcal{A}, \rho, \phi)$. Now, if $\mathcal{C}$ was returned by the algorithm since $\mathcal{C} \in Res_s$, then $|\mathcal{C}'| < |\mathcal{C}|$ implies that the algorithm has considered $\mathcal{C}'$ earlier. From this point, however, $\mathcal{C} \notin Power$, a contradiction. If $\mathcal{C}$ was returned since $\mathcal{C} \in Res_l$, the filtering in Line 17 results in a contradiction. Therefore, $\mathcal{C}$ is a but-for cause for ϕ in ρ of $\mathcal{A}$. ◀

▶ **Theorem 24.** *Checking and computing causes for an effect ϕ on the run ρ in a network of timed automata $\mathcal{A}$ is* $\mathsf{EXPSPACE}(\phi)$*-complete.*

Proof. Analyzing the computational compexity of Algorithms 1 and 2 shows the two problems of cause checking and computations to be solvalbe in $\mathsf{EXPSPACE}(\phi)$. For showing $\mathsf{EXPSPACE}(\phi)$-hardness, we present a reduction from the model checking problem, that is $\mathsf{EXPSPACE}$-complete [4]. We construct for a timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$ an extended reduction automaton $\mathcal{A}_{\mathsf{red}} := (Q \dot\cup \{s_{\mathsf{new}}, q_{\mathsf{new}}\}, s_{\mathsf{new}}, X \dot\cup \{x_{\mathsf{new}}\}, E', I', L')$ over an extended set of actions $Act \dot\cup \{\alpha_{\mathsf{new}}, \beta_{\mathsf{new}}\}$ and labels $AP \dot\cup \{p_{\mathsf{new}}\}$ whereby s_{new} and q_{new} are fresh locations, α_{new} and β_{new} are fresh actions, x_{new} is a fresh clock, p_{new} is a fresh atomic proposition, and we have

$$E' := E \cup \{(s_{\mathsf{new}}, \top, \alpha_{\mathsf{new}}, \{x_{\mathsf{new}} := 1\}, q_{\mathsf{new}}), (q_{\mathsf{new}}, \top, \alpha_{\mathsf{new}}, \emptyset, q_{\mathsf{new}}), (s_{\mathsf{new}}, \top, \beta_{\mathsf{new}}, \epsilon, q_0)\},$$

$$I'(q) := \begin{cases} I(q), & q \in Q, \\ x_{\mathsf{new}} \leq 0, & q = s_{\mathsf{new}}, \\ x_{\mathsf{new}} \leq 1, & q = q_{\mathsf{new}}, \end{cases} \quad \text{and} \quad L'(q) := \begin{cases} L(q), & q \in Q, \\ \{\,\}, & q = s_{\mathsf{new}}, \\ \{p_{\mathsf{new}}\}, & q = q_{\mathsf{new}}. \end{cases}$$

That is, $\mathcal{A}_{\mathsf{red}}$ is an extension of $\mathcal{A}$ that has a new initial location s_{new} from which a direct transition (delay of 0) to either a second new location q_{new} or to the initial state of the original automaton $\mathcal{A}$ is enforced. This new automaton has a new run, namely $\rho_{\mathsf{red}} := (s_{\mathsf{new}}, u_0) \xrightarrow{0.0} \xrightarrow{\alpha_{\mathsf{new}}} (q_{\mathsf{new}}, u_0) \xrightarrow{1.0} \xrightarrow{\alpha_{\mathsf{new}}})^\omega$ fulfilling the effect $\phi_{\mathsf{red}} := \phi \vee \Diamond\, p_{\mathsf{new}}$.

Instances $(\mathcal{A}, \phi)$ of the model checking problem are now mapped to instances of the cause checking problem via the reduction $r : (\mathcal{A}, \phi) \mapsto (\mathcal{A}_{\mathsf{red}},\ \rho_{\mathsf{red}},\ \phi_{\mathsf{red}},\ \mathcal{C}_{\mathsf{red}})$ with $\mathcal{C}_{\mathsf{red}} := \{((\alpha_{\mathsf{new}}, 1, \rho(\mathcal{A}_{\mathsf{red}})\}$. Now, we have that $\mathcal{A} \not\models \phi$ iff $\mathcal{C}_{\mathsf{red}}$ is a cause for ϕ_{red} in ρ_{red} of $\mathcal{A}_{\mathsf{red}}$. ◀

C Contingency Automaton

In this section, we illustrate the contingency automaton construction from Example 20. For automaton $\mathcal{A}_1$, run ρ from Subsection 1.1 and its sequence of local locations $\mathsf{loc}(\rho, \mathcal{A}_1) = \mathsf{init}, \mathsf{crit}, \mathsf{init}$, the location contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ is depicted in Figure 4. Following Definiton 16, the contingency automaton is constructed in the following way:

- we copy the automaton $|\rho(\mathcal{A}_1)|$ times, to encode the current step in the states (second component of the tuple);
- we redirect the transitions from the original automata (black transitions) to their target location in the next copy;
- in each step, we add contingency transitions (red transitions), allowing the location to be reset to what it had been in the corresponding step of the original run ρ.

We can now find a counterfactual run in $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ avoiding the critical section by taking the contingency from $(\mathsf{init}, 1) \xrightarrow{\beta} (\mathsf{init}, 2)$. That is, the location after the second transition is reset to what it had been in the original run, namely to location init. The construction of the clock contingency automaton works in a similar way: We allow additional transitions to reset the clocks as they had been in the original run at the respective positions.

D Experimental Setup and Results for Fischer's Protocol

We report on the details in the experimental evaluation for Fischer's protocol and the causes identified by the tool. Fischer's protocol is a popular real-time mutual exclusion protocol, we depict one component $\mathcal{A}_i$ in Figure 5. We then test the effect $\phi := \neg \square_{[0,\infty)} \neg \mathsf{crit}_1$ on the network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, i.e. that the first component reaches its critical section.

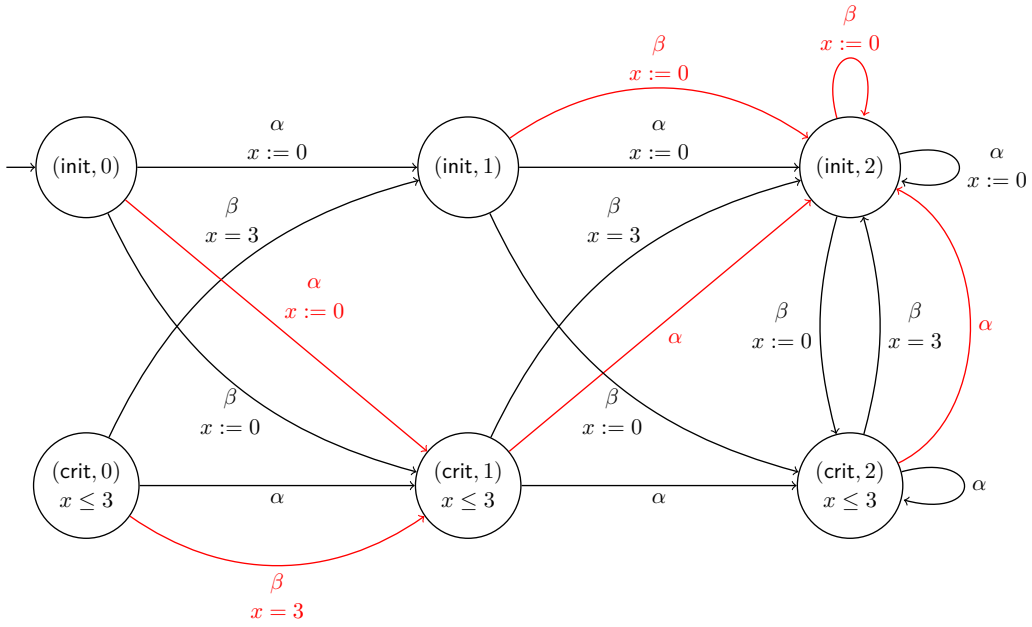


Figure 4 The contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ from Example 20.

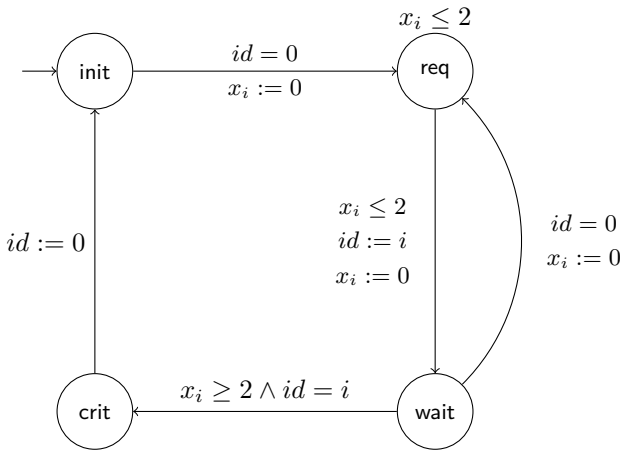


Figure 5 A single component automaton $\mathcal{A}_i$ of Fischer's protocol network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$.

We state the tested runs and results exemplary for $n = 2$:

$$\rho_1 := \big(\{\mathsf{init}_{1,2}\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{init}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{init}_2\} \xrightarrow[4.0]{\tau_1} \{\mathsf{wait}_1, \mathsf{init}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{crit}_1, \mathsf{init}_2\} \xrightarrow[2.0]{\tau_1} \big)^\omega$$

$$\rho_2 := \{\mathsf{init}_{1,2}\} \xrightarrow[1.0]{\tau_2} \Big(\{\mathsf{init}_1, \mathsf{req}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{req}_2\} \xrightarrow[1.0]{\tau_2} \{\mathsf{req}_1, \mathsf{wait}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{wait}_1, \mathsf{wait}_2\} \xrightarrow[3.0]{\tau_1} \{\mathsf{crit}_1, \mathsf{wait}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{init}_1, \mathsf{wait}_2\} \xrightarrow[1.0]{\tau_2} \Big)^\omega$$

The detected causes for those two runs are reported in Table 3. As in Fischer's protocol only internal actions are used, the detected root causes only consist out of delay actions. Further, since we do not encounter cases of preemption, but-for and actual causes agree on this example. For $n = 3, 4$ we tested runs with the same lasso-part.

Table 3 Overview of the root causes found in the experiments for Fischer's protocol.

Ref.	ρ_1: BF Causes	ρ_1 : Actual Causes	ρ_2: BF Causes	ρ_2 : Actual Causes
1	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_2)\}$	$\{(1.0, 1, \mathcal{A}_2)\}$
2	$\{(4.0, 3, \mathcal{A}_1)\}$	$\{(4.0, 3, \mathcal{A}_1)\}$	$\{(2.0, 1, \mathcal{A}_1)\}$	$\{(2.0, 1, \mathcal{A}_1)\}$
3			$\{(2.0, 2, \mathcal{A}_2)\}$	$\{(2.0, 2, \mathcal{A}_2)\}$
4			$\{(2.0, 2, \mathcal{A}_1)\}$	$\{(2.0, 2, \mathcal{A}_1)\}$
5			$\{(3.0, 3, \mathcal{A}_1),$ $(6.0, 3, \mathcal{A}_2)\}$	$\{(3.0, 3, \mathcal{A}_1),$ $(6.0, 3, \mathcal{A}_2)\}$