Checking and Sketching Causes on Temporal Sequences

Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Julian Siber

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{raven.beutner,finkbeiner,hadar.frenkel,julian.siber}@cispa.de

Abstract. Temporal causality describes what concrete input behavior is responsible for some observed output behavior on a trace of a reactive system, and can be used to, e.g., generate explanations for counterexamples uncovered by a model checker. In this paper, we present `CATS`, the first tool that can automatically verify whether a given temporal property (specified in QPTL) is a cause for some observed ω-regular effect. In addition to *checking* whether a given property is a cause, `CATS` can *search* for potential causes by exhaustively exploring a cause sketch, i.e., a temporal formula in which some parts are left unspecified. Our experiments show that `CATS` can effectively check causes and search for causes in small reactive systems.

1 Introduction

Causality analysis plays an increasingly important role in computer science and has practical applications such as explaining the behavior of systems [3,5,17,15], establishing accountability in multi-agent systems [19], and as a reasoning tool for verification [35,36] and synthesis [2]. These approaches rely on the philosophical foundations of Lewis and Hume [39,33] that suggest *counterfactual reasoning* as a method of establishing a causal relationship between events. Following this reasoning, a property (or, in previous works, an event) is only a cause if, in case the cause does not occur, the effect is absent as well. Halpern and Pearl [29,30] formalized these notions into a rigorous system of structural equations over *finite* sets of events (variables). However, when naïvely applying it to reactive systems, i.e., systems that continuously interact with their environment, Halpern and Pearl's original definition fails as the behavior is characterized by *infinitely* many variables. Recently, Coenen et al. [18] lifted the ideas of Halpern and Pearl to the temporal domain and presented a framework in which (symbolic) temporal properties, expressed in temporal logic such as LTL or QPTL, constitute causes and effects.

Example 1. Consider the system of Figure 1 over inputs i_1, i_2 and output $\lightning$, which marks a failure. When checking whether the system satisfies $\square \neg \lightning$, a model checker might return $\pi = \{i_1, i_2\}\{i_1, i_2\}\{i_1\}(\{i_1, \lightning\})^\omega$ as a concrete counterexample. We are interested in explaining the effect $\varphi_E = \diamondsuit \lightning$ on the given (actual) error trace. Using the theory presented in [18] we can formally show that

$\varphi_C = i_1 \wedge \bigcirc\bigcirc\big((\neg i_2)\,\mathcal{U}(i_1 \wedge \neg i_2)\big)$ constitutes an actual cause for φ_E. Such a cause provides important information for debugging: It pinpoints that, in the first position, only i_1 is relevant; it does not refer to the second position on the trace as those inputs are irrelevant; and it precisely captures the information that, in order to reach the error state, i_1 must occur *strictly before* i_2. △

Coenen et al. [18] showed that checking if an ω-regular property constitutes an actual cause on a lasso-shaped trace is decidable. However, as their theory was not implemented, reasoning about causal relationships required manual computation. Given the intricate nature of causality (which encompasses complex features such as contingencies and interventions [29,30,18]), this manual reasoning is both time-consuming and error-prone.

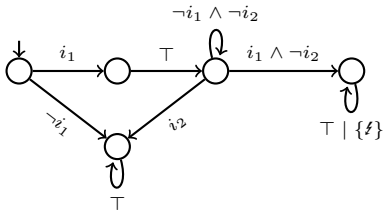


Fig. 1: An example system. Each edge has the form $\phi \mid o$ where ϕ is a Boolean formula over the inputs and $o \in 2^{\{\lightning\}}$ is a set of outputs. We write ϕ instead of $\phi \mid \emptyset$.

CATS. In this paper, we present `CATS` [11], short for *Causal Analysis on Temporal Sequences*, a fully-automatic implementation of the theory of [18]. `CATS` can check if a given temporal property (specified in QPTL [42]) qualifies as a cause on an actual lasso trace. Internally, our tool relies on encoding the cause-checking problem into the model-checking problem for *hyperproperties* [16,21], i.e., properties that relate multiple traces in a system.

Our tool serves two purposes: First and foremost, `CATS` allows for the automatic checking of symbolic causes (temporal formulas). This is a useful feature in many settings, perhaps most prominently in counterexample debugging, where we are interested in getting succinct yet informative summaries of what temporal input behavior triggered the violation of a property.

Secondly, `CATS` serves as a playground to experiment with temporal cause definitions. Causality definitions are inherently linked to human intuition, and coming up with a useful one is difficult (as, e.g., witnessed by the multiple updates of Halpern and Pearl's definitions [29,30,28]). A fully-automatic tool for cause checking allows us to experiment with more evolved causality definitions and see (within a few seconds; and with no manual computation) how small changes in the definition transfer to actual examples. This is particularly important in a temporal setting, where many parameters need to be fixed (e.g., what constitutes a "closer" trace as defined by Lewis [39,22,18]).

Cause Sketching. The main purpose of `CATS` is to check if a given temporal property qualifies as a cause. However, often it is also useful to *infer* a cause automatically. While general synthesis of temporal causes is not possible yet (cause synthesis corresponds to the search for an appropriate *set of traces*, a problem that is notoriously difficult [10]), we propose a very useful approximation in the

form of cause sketching. Inspired by program sketching [43] and query checking [12], CATS supports cause *sketches* – temporal properties in which some propositional holes are left unspecified. CATS then attempts to find an appropriate instantiation for all holes to generate an actual cause. On the theoretical side, we show that for time-bounded effects (i.e., properties whose satisfaction or violation can be determined by only looking at the first n steps), any potential cause only needs to refer to the first n steps. On the practical side, this implies that for time-bounded effects (which naturally occur in counterexample analysis where an error occurs after a fixed number of steps), there exists a cause sketch that encompasses all potential causes.

Related Work. We base our causality analysis on the theoretical foundations for temporal causality by Coenen et al. [18] (recapped in Section 3). For a comprehensive survey on applications of causality in formal methods and for providing explanations, see [3]. Leue et al. [37,13] propose a symbolic description of counterfactual causes in *Event Order Logic* (implemented into the tool SpinCause [38]), which can reason about the *ordering* of LTL-definable events. In particular, the logic cannot reason about the *absolute* timing as is, e.g., needed to specify that the input at the second position is part of the cause (cf. Example 1). Gössler and Métayer [23] define causality for component-based systems, and Gössler and Stefani [24] study theoretical foundations of causality based on counterfactual builders. Both works differ from our approach as we consider *actual* causality on the property level.[1]

Many existing works focus on explaining *finite* counterexamples [25,26,4,44]. Beer et al. [5] present a tool for causal analysis of finite traces with respect to LTL specifications by highlighting events that led to the violation. HyperVis [31] provides visualization of counterexamples for hyperproperties through highlighting. Coenen et al. [17] infer a cause for a hyperproperty violation, defined as a finite set of events (i.e., time points) on the trace. In contrast to all of the above, CATS provides *symbolic* causes (defined in QPTL) that can refer to infinitely many time points and – given their logical nature – are easier to understand. At the same time, the underlying theory provides strong guarantees for time-bounded effects as, e.g., encountered in error analysis (cf. Proposition 1).

Structure. In the next section, we provide preliminaries and recap the theory presented in [18]. Section 4 gives an overview of CATS. In Section 5, we evaluate the cause-checking ability of CATS on both hand-crafted examples and systems drawn from the SYNTCOMP competition [34]. In Section 6, we study CATS's cause-sketching functionality.

[1] The term "actual causality" was coined by Halpern and Pearl [29] and describes causes in a concrete (actual) instance (e.g., a trace) of a system. In contrast, *global* causality describes all of the system behavior that can cause an effect.

2 Preliminaries

Systems and Traces. We model a reactive system as a finite state transition system $\mathcal{T}$ over a set of atomic propositions $\text{AP} = I \cup O$, which is partitioned into inputs I and outputs O. A system then generates a set of traces $\mathit{Traces}(\mathcal{T}) \subseteq (2^{\text{AP}})^\omega$. For more details, see [18, § 5.1].

QPTL and HyperQPTL. HyperQPTL [41,9] extends linear-time temporal logic (LTL) [40] by adding quantification over atomic propositions, as well as explicit quantification over traces in a system. Given a set of trace variables $\mathcal{V}$, HyperQPTL formulas are defined by the following grammar

$$\varphi ::= \forall \pi.\, \varphi \mid \exists \pi.\, \varphi \mid \forall q.\, \varphi \mid \exists q.\, \varphi \mid \psi$$
$$\psi ::= a_\pi \mid q \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc \psi \mid \psi \,\mathcal{U}\, \psi$$

where $\pi \in \mathcal{V}$ is a trace variable, $a \in \text{AP}$ is an atomic proposition, and $q \notin \text{AP}$ is a fresh quantified proposition. We also consider the usual derived Boolean constants $(\top, \bot)$, Boolean connectives $(\vee, \rightarrow, \leftrightarrow, \not\leftrightarrow)$, and temporal operators *eventually* $(\diamondsuit \psi := \top \,\mathcal{U}\, \psi)$ and *globally* $(\square \psi := \neg(\diamondsuit \neg\psi))$.

In a HyperQPTL formula, atomic propositions are indexed by trace variables. For example, $\psi := \square(a_\pi \leftrightarrow b_\sigma)$ states that, on the trace that is bound to trace variable π, a holds iff b holds on the trace bound to trace variable σ. This allows us to compare multiple traces within a temporal formula which we use, e.g., to define a distance metric on traces. The trace variables in the formula are (existentially or universally) quantified at the top level. For example $\forall \pi. \exists \sigma.\, \psi$ states that for every trace π in the system, there exists a trace σ such that ψ holds on those two traces. In addition to trace quantification, HyperQPTL features propositional quantification (as found in QPTL [42]). This allows us to capture all ω-regular causes and effects, even those that are not LTL-definable. For more details on HyperQPTL and the full semantics, see [9] or [41].

3 Temporal Causality

Our tool is based on the theory of temporal causality as defined by Coenen et al. [18], extending Halpern and Pearl's foundational definition of actual causality to the setting of temporal causes and effects in reactive systems. In this section, we recall the key aspects from [18].

Interventions. Interventions define the counterfactual scenarios where the cause does not appear. Counterfactuals are the *closest* worlds in which the cause does not appear [39].

Example 2. Consider, for example, the LTL property $\varphi = \square a$ and the actual trace $\pi = \{a\}^\omega$. To have a meaningful definition of counterfactuals, that is, closest worlds in which the cause φ does not appear, it is not enough to negate the formula φ, as this would result in a set of traces that are not necessarily close enough to the original trace π. $\triangle$

Instead, Coenen et al. [18] adopt the idea of distance metrics known from Lewis [39]. Given a trace π, a *distance metric* $<_\pi$ is a *strict* partial order such that $\sigma <_\pi \sigma'$ if trace σ is closer to π than trace σ'. The *intervention set* $V(\varphi, <_\pi)$ then consists of the *closest* traces σ that do not satisfy φ:

$$V(\varphi, <_\pi) = \big\{\sigma \in \mathit{Traces}(\mathcal{T}) \mid \sigma \vDash \neg\varphi \wedge \neg\exists \sigma' \in \mathit{Traces}(\mathcal{T}).\, \sigma' \vDash \neg\varphi \wedge \sigma' <_\pi \sigma\big\}.$$

Distance Metrics in HyperQPTL. To handle distance metrics algorithmically, we consider them as being defined by a HyperQPTL formula. For example

$$\sigma <_\pi^{min} \sigma' \iff \square\big(\bigwedge_{i\in I}(i_\pi \not\leftrightarrow i_\sigma) \rightarrow (i_\pi \not\leftrightarrow i_{\sigma'})\big) \wedge \Diamond \bigvee_{i\in I}(i_\sigma \not\leftrightarrow i_{\sigma'}) \tag{1}$$

specifies that σ is closer to π than σ' iff whenever σ and π differ on some input, so should σ' and π, *and* σ' and σ differ at least in one position.

Example 3. In Example 2, $\{\}(\{a\})^\omega$ is closer (w.r.t. $<_\pi^{min}$) to π than $\{\}\{\}(\{a\})^\omega$. The intervention set $V(\varphi, <_\pi^{min})$ thus contains all traces in which a does not appear at *exactly* one position, i.e., traces of the form $(\{a\})^*\{\}(\{a\})^\omega$. △

Causality on Temporal Sequences. We are now ready to recall Coenen et al.'s [18] definition of temporal causality. Following Halpern and Pearl, Coenen et al. [18] use *contingencies* to deal with cases of preemption, i.e., scenarios where a possible cause gets nullified by another earlier cause for the same effect. Formally, they define the *counterfactual automaton* $\mathcal{C}_\pi^\mathcal{T}$ to account for the contingencies of a lasso trace π. See [18, § 5.2] for details.

Definition 1 ([18]). *Let $\mathcal{T}$ be a system over $AP = I \cup O$, $\pi \in \mathit{Traces}(\mathcal{T})$ a trace, $<_\pi$ a distance metric, and φ_C, φ_E two QPTL formulas over I and O, respectively. We say that φ_C is a* cause *of φ_E on π in $\mathcal{T}$ if the following three conditions hold:*

PC1: *$\pi \vDash \varphi_C$ and $\pi \vDash \varphi_E$.*
PC2: *For every counterfactual input sequence $\sigma \in V(\varphi_C, <_\pi)$, there is some trace $\pi' \in \mathcal{C}_\pi^\mathcal{T}$ s.t. $\pi' \vDash \neg\varphi_E$ and $\bigwedge_{i\in I}\square(i_\pi \leftrightarrow i_{\pi'})$.*
PC3: *There is no φ_C' s.t. $\varphi_C' \rightarrow \varphi_C$ is valid and φ_C' satisfies PC1 and PC2.*

The *counterfactual* condition (PC2) requires that for every closest input sequences in which the cause does *not* hold, we can use contingencies to avoid the effect. PC1 requires that cause and effect are satisfied by the *actual* trace at hand, and PC3 poses that the cause is *semantically minimal*.

Infinite Chains and Vacuity Condition. The above $<_\pi^{min}$ metric may admit infinite chains of ever smaller interventions, resulting in an empty intervention set $V(\varphi, <_\pi)$, which renders PC2 vacuously valid.

Example 4 ([18]). Let us consider the cause candidate $\varphi := \square\Diamond a$ and the trace $\pi = \{a\}^\omega$. Under the distance metric $<_\pi^{min}$ (1), there exists no closest traces that negate φ. For example, $\{\}^\omega >_\pi^{min} \{a\}(\{\})^\omega >_\pi^{min} \{a\}\{a\}(\{\})^\omega >_\pi^{min} \cdots$ forms an infinite chain with no minimal element. Formula φ thus qualifies as a cause for all effects that hold on π. △

To catch such situations, we add an additional *vacuity condition*, which, e.g., ensures that $\square\diamondsuit a$ (cf. Example 4) never constitutes a non-vacuous cause.

Definition 2 (Vacuity Condition). *Under the conditions of Definition 1, φ_C is a* non-vacuous *cause, if, in addition to PC1-PC3, the following holds:*

PC4: *The intervention set $V_\pi(\varphi_C, <_\pi)$ is non-empty.*

Some distance metrics are strong enough to always satisfy PC4. For example, Coenen et al. [18] propose an extension of $<_\pi^{min}$ which only orders traces that have the same *rejection structure*, that is, traces that falsify the cause formula at the same positions of the trace (see [18]). An alternative extension of $<_\pi^{min}$ we have discovered when experimenting with `CATS` is the following:

$$\sigma <_\pi^{full} \sigma' \iff (\sigma <_\pi^{min} \sigma') \wedge \bigwedge_{i \in I} \big(\square\diamondsuit(i_\pi \not\leftrightarrow i_\sigma)\big) \rightarrow \big(\square(i_\sigma \leftrightarrow i_{\sigma'})\big)$$

The metric $<_\pi^{full}$ only orders traces that have the same infinite interventions (with respect to individual atomic propositions).

Example 5. Recall Example 4. The traces $\sigma = \{\}^\omega$ and $\sigma' = \{a\}(\{\})^\omega$ are not ordered by $<_\pi^{full}$, as σ already intervenes on infinitely many positions against a in $\pi = \{a\}^\omega$ and σ' does not equal σ when both are projected to a. The infinite chain w.r.t. $<_\pi^{min}$ from Example 4 thus does not exist; all elements in the chain are minimal w.r.t. $<_\pi^{full}$. △

The modular design of `CATS` encourages experiments with *different* distance metrics. By default, `CATS` uses $<_\pi^{min}$ (1) ([18]) together with the vacuity condition PC4, as our experiments show that this performs best in practice.

4 CATS: Tool Overview

In this section, we discuss the input of `CATS` (Section 4.1) and provide a basic overview of the internal working (Section 4.2). All experiments in this paper were conducted on a Macbook Pro with an M1 Pro CPU and 32 GB of memory.

4.1 Input Specification

`CATS` supports arbitrary ω-regular properties specified in QPTL [42], an extension of LTL with explicit quantification over propositions. A *cause-checking* instance specifies the following: **(1)** The system – given as an arbitrary automaton in the `HANOI`-automaton format [1]; **(2)** a partition of the atomic propositions into inputs and outputs; **(3)** the cause and effect as QPTL formulas; and **(4)** a lasso-shaped trace. When given a cause-*checking* instance, `CATS` determines if the given cause candidate qualifies as an actual cause.

`CATS` can also be used in cause-*sketching* mode. In this mode, the cause is a QPTL formula that includes holes, and `CATS` tries to find a formula within

this sketch (i.e., a formula where all holes in the sketch are instantiated with propositional – non-temporal – formulas) that qualifies as a cause. In sketching mode, CATS either provides an actual cause or determines that no formula within the given sketch qualifies as a cause. See [12] for details on sketching in the context of *query checking*.

4.2 Algorithmic Core

Internally, CATS relies on a HyperQPTL-based encoding of the cause-checking problem. As observed in [18], the causality requirements PC1-PC3 can be expressed as a HyperQPTL model-checking problem. CATS decomposes this model checking problem as much as possible into 5 separate checks instead of one large one as used in [18]. Having multiple (but smaller) checks is crucial for the performance of CATS on larger cause formulas. By leveraging the HyperQPTL-encoding, CATS can discharge most of the heavy computation as HyperQPTL model-checking problems. For this, CATS relies on the automata-based model checker AutoHyper [8,9]; alternative hyperproperty verification approaches [6,7,32] can be substituted easily.

Handling Contingencies. If desired by the user, CATS adds the ability to reason about contingencies – a central feature of Halpern and Pearl's actual causality. For details on this so-called *contingency automaton*, see [18, Def. 8].[2]

Trace Checking for Cause Sketching. When invoked in sketching mode, most cause candidates within a sketch do not hold on the given trace and thus violate PC1. CATS can filter out these instances very effectively by employing an inexpensive trace-check of the candidate on the given lasso, which prunes the search space significantly. While there are exponentially many candidates within each cause sketch, in practice, only a few satisfy PC1 and thus progress to the algorithmically harder stages that require (proper) hyperproperty model checking.

5 Evaluation 1 - Cause Checking

To evaluate the cause-checking with CATS, we use both hand-crafted examples (Section 5.1) and instances from the annual SYNTCOMP competition (Section 5.2).

5.1 Hand-Crafted Examples

We collected a range of hand-crafted instances (consisting of system, lasso, cause, and effect). These systems are typically very small (so performance is not an issue) and serve as a test of the underlying causality definition. We depict the results in Table 1.

[2] Coenen et al. [18] use the assumption that every state of the transition system is labeled by a unique set of outputs. In practice, this assumption is unrealistic, so, in many cases, the contingency automaton leads to unintuitive results and prevents the discovery of causes. In CATS, we thus decided to allow the user to decide whether or not to use contingencies.

Table 1: Hand-crafted cause-checking instances. We display whether or not the cause candidate is a cause (✓ if it is a cause and ✗ if it is not) and the time taken by `CATS` in seconds. Example 1, mod changes the actual trace to $\pi = \{i_1, i_2\}\{i_1, i_2\}\{i_1\}(\{\lightning\})^\omega$ such that φ_C is no longer a cause.

Instance	Res	t	Instance	Res	t
Spurious Arbiter [18]	✗	0.6	Example 6, globally	✗	0.7
Arbiter simple [18]	✓	1.2	Example 7	✓	2.1
Arbiter [18]	✓	9.7	Example 8	✓	1.1
Example 1	✓	0.9	Example 8 mod	✗	1.2
Example 1, mod	✗	1.0	TP Left [18, Thm. 6]	✗	0.3
Example 6, odd	✓	1.4	TP Right [18, Thm. 6]	✓	1.1

Example 6. Consider the system in Figure 3a, which models a simple arbiter for two processes; each can issue a request ($I = \{r_0, r_1\}$) and might be answered by a grant ($O = \{g_0, g_1\}$). Importantly, the arbiter is biased towards process 0, i.e., prioritizes process 0 if both issue a request. Suppose we observe the trace $(\{r_0, r_1\}\{r_0, g_0\})^\omega$ and are interested in a cause for $\varphi_E = \square\neg g_1$, i.e., process 1 never gets a grant. `CATS` can automatically verify the cause $\varphi_C = \exists q.q \wedge \square(q \leftrightarrow \bigcirc\neg q) \wedge \square(q \rightarrow r_0)$ (instance Example 6, odd), stating that the cause is that process 0 issues requests at all *odd* positions. In particular, `CATS` can also infer that $\square r_0$ is *not* a cause (instance Example 6, globally); the requests at even positions are irrelevant for the effect. Such a precise cause cannot be expressed in LTL and requires the ω-regularity possible in QPTL. We stress that such an *automatic* analysis was not possible before, and each instance required manual (error-prone) checking. △

5.2 Syntcomp Evaluation

In the previous section, we considered hand-crafted examples that stress the underlying theory. In this section, we test the performance of `CATS` on a larger set of benchmarks. To obtain an interesting set of reactive systems, we use benchmarks from the reactive synthesis competition (`SYNTCOMP`) [34]. `SYNTCOMP` includes a collection of LTL formulas that specify requirements for a diverse collection of reactive systems. We use existing LTL synthesis tools (in our case `ltlsynt` [20]) to synthesize a strategy/system for each (realizable) LTL specification (within a timeout of 5 minutes). We obtain a collection of 204 systems of varying sizes. For each `SYNTCOMP` system, we randomly generate 10 different lasso traces and use `spot`'s `randltl` to generate random cause and effect formulas (over the inputs and outputs, respectively). This gives us a total of $204*10 = 2040$ cause-checking instances. In Figure 2, we depict the running time of `CATS` against the size of the underlying system. We observe that the vast majority of instances can be

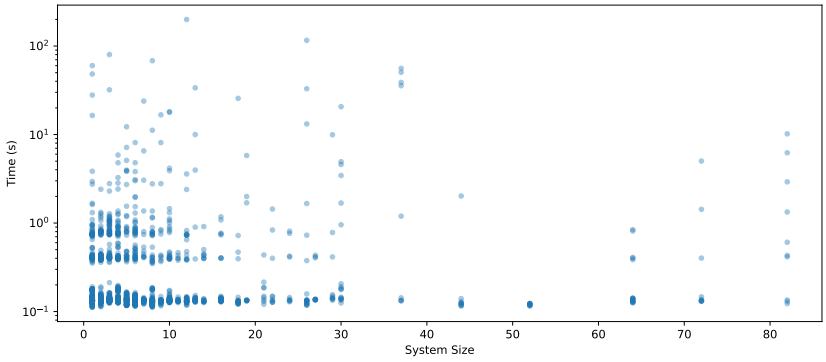


Fig. 2: We use CATS to check different cause-effect pairs in systems obtained from SYNTCOMP [34] benchmarks. Note that the time scale is logarithmic.

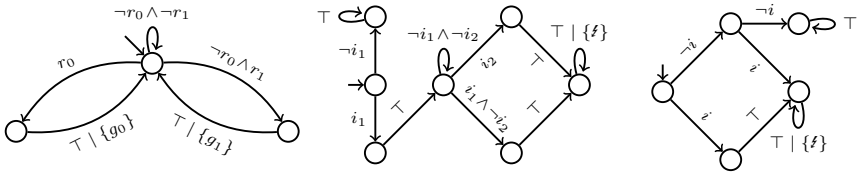
solved in less than 10 seconds. We can also see how the running time of each instance depends on the number of checks that are needed; due to the incremental checking by CATS (based on the decomposition of the cause-checking formula), instances that, e.g., already violate PC2 are not checked further. The overall running time thus depends on the number of stages that are passed.

6 Evaluation 2 - Cause Sketching

As already alluded to in the introduction, a typical use case of causal analysis is the analysis of a counterexample. The user can use such a cause to, e.g., extract a minimal error from the concrete error trace and effectively debug the system.

Example 7. Consider the simple system in Figure 3b in which output $\lightning$ marks an error. A model checker might return the concrete path $\pi = \{i_1, i_2\}^4(\{\lightning\})^\omega$. While the concrete path reaches the error state, it provides very little information about which inputs actually caused the error. Instead, we can use CATS to find a cause for the effect $\varphi_E = \bigcirc\bigcirc\bigcirc\bigcirc\lightning$. When given to CATS with an appropriate sketch, it will compute the cause $\varphi_C = i_1 \wedge \bigcirc\bigcirc(i_1 \vee i_2)$, which characterizes exactly the events on π that are of relevance: i_1 must hold in the first step, and either i_1 or i_2 must hold in the third (to avoid the self-loop). Note that this cause is tightly coupled with the concrete trace π. In particular, the cause does not describe *all* input events that lead to the error state, but only the minimal changes needed to *avoid* the error on the concrete example. The time for *checking* this causal relationship is depicted in Table 1 (instance Example 7). △

Example 8. As a second example, consider the system in Figure 3c and the concrete path $\pi = \{i\}^2(\{\lightning\})^\omega$. CATS can automatically verify the cause $\varphi_C = i \vee \bigcirc i$ for the effect $\varphi_E = \bigcirc\bigcirc\lightning$. Note that this cause is disjunctive as we need to intervene on i in the first *and* second step to avoid the effect. Symbolic causes (as supported by CATS) can describe such effects very succinctly. In contrast,



(a) (b) (c)

Fig. 3: Three example systems. Each edge has the form $\phi \mid o$ where ϕ is a boolean formula over I and $o \subseteq O$ a set of outputs. We write ϕ instead of $\phi \mid \emptyset$.

previous methods cannot handle such examples: they are either limited to a finite-variable setting and *conjunctive* causes [29,30] or can only reason about the order of events but *not* about the concrete time using $\bigcirc$'s [37,38]. The time for *checking* this causal relationship is given in Table 1 (instance Example 8). Table 1 also depicts a modified version using trace $\pi = \{i\}\{\}(\{\lightning\})^\omega$ (instance Example 8,mod). This results in $\varphi_C = i \vee \bigcirc i$ no longer being a cause. $\triangle$

6.1 Causes for Time-bounded Effects

When employing causality-based analysis on counterexamples, we often encounter effects of the form $\bigcirc^n \lightning$ for some $n \in \mathbb{N}$. We refer to such effects as *time-bounded effects*. We can formally prove that – within the causality framework of Coenen et al. [18] – an effect that is time-bounded by $n \in \mathbb{N}$ has a cause iff it has a time-bounded cause, i.e., a cause that only refers to the first n steps.

Proposition 1. *Let $\varphi_E = \bigcirc^n \psi$ be an effect, where ψ does not contain temporal operators, and let π be a trace. Then, there exists a cause for φ_E on π iff there exists a cause that uses at most n nested $\bigcirc$'s, and no other temporal operators.*

When looking for causes of the form $\bigcirc^n \lightning$ for some n, it thus suffices to check for causes that refer to the first n positions. It is easy to see that there exists a cause sketch that captures all such candidates (a simple DNF with atoms of the form $\bigcirc^j \psi$ with $j \leq n$).

6.2 Automatically Sketching Causes

To evaluate `CATS`'s cause-sketching ability, we use `spot`'s `randaut` [20] to generate 100 random systems of varying size (between 10 and 50 states) and randomly mark one state with a fresh $\lightning$ proposition. Using a model checker (in our case, a simple breath-first-search), we verify whether the error state is reachable, and if it is, compute a concrete (lasso) trace reaching the error in say n steps. We then use `CATS` to infer a cause for $\varphi_E = \bigcirc^n \lightning$, which, by Proposition 1, can be done by exploring an appropriate sketch.

Our results are displayed in Table 2. We find that although `CATS` explores many candidates, most of them can be pruned early using the inexpensive trace

Table 2: Evaluation of CATS's sketching for counterexample analysis. We depict the average length of the counterexample trace (avg. $|\pi|$), the average number of cause candidates checked (avg. $\#_{check}$), the average time (in seconds) spent on cause *checking* (avg. t_{check}), the average total time needed by CATS (avg. t), and the percentage of cases in which we could find a cause (avg. success).

avg. $\|\pi\|$	**avg.** $\#_{check}$	**avg.** t_{check}	**avg.** t	**avg. success**
6.28	169.1	2.08 s	2.19 s	86%

check for PC1 (cf. Section 4.2). The actual time t_{check} for checking (which takes up the vast majority of CATS's total computation time) is thus reasonable, as only a few of the (on average) 169.1 candidates progress to the expensive hyperproperty model-checking phase.

Limitations. We emphasize that CATS can, obviously, not compete with dedicated methods for counterexample analysis [27,5,14]. The big advantage of CATS stems from its reliance on an advanced theory that is *not limited* to counterexample analysis but applicable to arbitrary causal relationships. Despite the strong theoretical foundations (dating back to Halpern and Pearl's seminal definition [29,30]), CATS provides strong guarantees on the existence of causes (Proposition 1) and performs well in small systems.

7 Conclusion

Causal analysis has a long tradition in the analysis of systems. While most efforts on comprehensive causal definitions (mainly originating in philosophy) focused on finite settings, recent work discussed causality in reactive systems, where cause and effect reason about the infinite behavior of a system [18]. In this paper, we have presented CATS, the first tool that pushes causality in reactive systems towards automation. With CATS, causality definitions are no longer condemned to be purely theoretical endeavors but can be applied and tested fully automatically in actual systems. This allows for discovery and verification causal relationships and serves as a playground to experiment with more advanced causality definitions.

With CATS, we have shown that verifying causes based on an advanced causality theory is possible in practice and that sketching is a viable method to infer causes. For future work, it is interesting to attempt to synthesize a (ω-regular) cause directly. In such developments, CATS can serve as a useful baseline and debugger.

Acknowledgments. This work was supported by the European Research Council (ERC) Grant HYPER (No. 101055412) and by DFG grant 389792660 as part of TRR 248 – CPEC.

References

1. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kretínský, J., Müller, D., Parker, D., Strejcek, J.: The hanoi omega-automata format. In: International Conference on Computer Aided Verification, CAV 2015. Lecture Notes in Computer Science, vol. 9206. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_31
2. Baier, C., Coenen, N., Finkbeiner, B., Funke, F., Jantsch, S., Siber, J.: Causality-based game solving. In: International Conference on Computer Aided Verification, CAV 2021. Lecture Notes in Computer Science, vol. 12759, pp. 894–917. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_42
3. Baier, C., Dubslaff, C., Funke, F., Jantsch, S., Majumdar, R., Piribauer, J., Ziemek, R.: From verification to causality-based explications. In: International Colloquium on Automata, Languages, and Programming, ICALP 2021. LIPIcs, vol. 198. Dagstuhl (2021). https://doi.org/10.4230/LIPIcs.ICALP.2021.1
4. Ball, T., Naik, M., Rajamani, S.K.: From symptom to cause: localizing errors in counterexample traces. In: Symposium on Principles of Programming Languages, POPL 2003. ACM (2003). https://doi.org/10.1145/604131.604140
5. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J.: Explaining counterexamples using causality. Formal Methods Syst. Des. **40**(1) (2012). https://doi.org/10.1007/s10703-011-0132-2
6. Beutner, R., Finkbeiner, B.: Prophecy variables for hyperproperty verification. In: Computer Security Foundations Symposium, CSF 2022. pp. 471–485. IEEE (2022). https://doi.org/10.1109/CSF54842.2022.9919658
7. Beutner, R., Finkbeiner, B.: Software verification of hyperproperties beyond k-safety. In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13371, pp. 341–362. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_17
8. Beutner, R., Finkbeiner, B.: AutoHyper: Explicit-state model checking for HyperLTL. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2023. Lecture Notes in Computer Science, vol. 13993. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_8
9. Beutner, R., Finkbeiner, B.: Model checking omega-regular hyperproperties with AutoHyperQ. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023. EPiC Series in Computing, vol. 94, pp. 23–35. EasyChair (2023). https://doi.org/10.29007/1xjt
10. Beutner, R., Finkbeiner, B., Frenkel, H., Metzger, N.: Second-order hyperproperties. In: International Conference Computer Aided Verification. Lecture Notes in Computer Science, vol. 13965, pp. 309–332. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_15
11. Beutner, R., Siber, J.: CATS - causal analysis on temporal sequences. Zenodo (2023). https://doi.org/10.5281/zenodo.8192053
12. Bruns, G., Godefroid, P.: Temporal logic query checking. In: IEEE Symposium on Logic in Computer Science, LICS 2001. IEEE (2001). https://doi.org/10.1109/LICS.2001.932516
13. Caltais, G., Guetlein, S.L., Leue, S.: Causality for general LTL-definable properties. In: Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST 2018. EPTCS, vol. 286 (2018). https://doi.org/10.4204/EPTCS.286.1

14. Chaki, S., Groce, A., Strichman, O.: Explaining abstract counterexamples. In: International Symposium on Foundations of Software Engineering, FSE 2004. ACM (2004). https://doi.org/10.1145/1029894.1029908
15. Chockler, H., Halpern, J.Y., Kupferman, O.: What causes a system to satisfy a specification? ACM Trans. Comput. Log. **9**(3) (2008). https://doi.org/10.1145/1352582.1352588
16. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6) (2010), https://doi.org/10.3233/JCS-2009-0393
17. Coenen, N., Dachselt, R., Finkbeiner, B., Frenkel, H., Hahn, C., Horak, T., Metzger, N., Siber, J.: Explaining hyperproperty violations. In: International Conference on Computer Aided Verification, CAV 2022. vol. 13371. Springer (2022), https://doi.org/10.1007/978-3-031-13185-1_20
18. Coenen, N., Finkbeiner, B., Frenkel, H., Hahn, C., Metzger, N., Siber, J.: Temporal causality in reactive systems. In: International Symposium on Automated Technology for Verification and Analysis, ATVA 2022. Springer (2022), https://doi.org/10.1007/978-3-031-19992-9_13
19. Datta, A., Garg, D., Kaynar, D.K., Sharma, D., Sinha, A.: Program actions as actual causes: A building block for accountability. In: Computer Security Foundations Symposium, CSF 2015. IEEE (2015). https://doi.org/10.1109/CSF.2015.25
20. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13372. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_9
21. Finkbeiner, B.: Logics and algorithms for hyperproperties. ACM SIGLOG News **10**(2), 4–23 (2023). https://doi.org/10.1145/3610392.3610394
22. Finkbeiner, B., Siber, J.: Counterfactuals modulo temporal logics. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023. EPiC Series in Computing, vol. 94, pp. 181–204. EasyChair (2023). https://doi.org/10.29007/qtw7
23. Gössler, G., Métayer, D.L.: A general trace-based framework of logical causality. In: International Symposium on Formal Aspects of Component Software, FACS 2013. Lecture Notes in Computer Science, vol. 8348. Springer (2013). https://doi.org/10.1007/978-3-319-07602-7_11
24. Gössler, G., Stefani, J.: Causality analysis and fault ascription in component-based systems. Theor. Comput. Sci. **837** (2020). https://doi.org/10.1016/j.tcs.2020.06.010
25. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. Int. J. Softw. Tools Technol. Transf. **8**(3) (2006). https://doi.org/10.1007/s10009-005-0202-0
26. Groce, A., Kroening, D., Lerda, F.: Understanding counterexamples with explain. In: International Conference on Computer Aided Verification, CAV 2004. Lecture Notes in Computer Science, vol. 3114. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_35
27. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: International Workshop on Model Checking Software, SPIN 2003. Lecture Notes in Computer Science, vol. 2648. Springer (2003). https://doi.org/10.1007/3-540-44829-2_8
28. Halpern, J.Y.: A modification of the halpern-pearl definition of causality. In: International Joint Conference on Artificial Intelligence, IJCAI 2015. AAAI Press (2015)

29. Halpern, J.Y., Pearl, J.: Causes and explanations: A structural-model approach. Part I: Causes. The British Journal for the Philosophy of Science **56**(4) (2005)
30. Halpern, J.Y., Pearl, J.: Causes and explanations: A structural-model approach. Part II: Explanations. The British Journal for the Philosophy of Science **56**(4) (2005)
31. Horak, T., Coenen, N., Metzger, N., Hahn, C., Flemisch, T., Méndez, J., Dimov, D., Finkbeiner, B., Dachselt, R.: Visual analysis of hyperproperties for understanding model checking results. IEEE Trans. Vis. Comput. Graph. **28**(1) (2022). https://doi.org/10.1109/TVCG.2021.3114866
32. Hsu, T., Sánchez, C., Bonakdarpour, B.: Bounded model checking for hyperproperties. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2021. Lecture Notes in Computer Science, vol. 12651, pp. 94–112. Springer (2021). https://doi.org/10.1007/978-3-030-72016-2_6
33. Hume, D.: An Enquiry Concerning Human Understanding. London (1748)
34. Jacobs, S., Pérez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018-2021. CoRR **abs/2206.00251** (2022). https://doi.org/10.48550/arXiv.2206.00251
35. Kupriyanov, A., Finkbeiner, B.: Causality-based verification of multi-threaded programs. In: International Conference on Concurrency Theory, CONCUR 2013. Lecture Notes in Computer Science, vol. 8052, pp. 257–272. Springer (2013). https://doi.org/10.1007/978-3-642-40184-8_19
36. Kupriyanov, A., Finkbeiner, B.: Causal termination of multi-threaded programs. In: International Conference on Computer Aided Verification, CAV 2014. Lecture Notes in Computer Science, vol. 8559, pp. 814–830. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_54
37. Leitner-Fischer, F., Leue, S.: Causality checking for complex system models. In: International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2013. Lecture Notes in Computer Science, vol. 7737. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_16
38. Leitner-Fischer, F., Leue, S.: SpinCause: a tool for causality checking. In: International Symposium on Model Checking of Software, SPIN 2014. ACM (2014). https://doi.org/10.1145/2632362.2632371
39. Lewis, D.K.: Counterfactuals. Cambridge, MA, USA: Blackwell (1973)
40. Pnueli, A.: The temporal logic of programs. In: Annual Symposium on Foundations of Computer Science, FOCS 1977. IEEE (1977). https://doi.org/10.1109/SFCS.1977.32
41. Rabe, M.N.: A temporal logic approach to information-flow control. Ph.D. thesis, Saarland University (2016)
42. Sistla, A.P.: Theoretical issues in the design and verification of distributed systems. Ph.D. thesis, Harvard University (1983)
43. Solar-Lezama, A.: Program sketching. Int. J. Softw. Tools Technol. Transf. **15**(5-6) (2013). https://doi.org/10.1007/s10009-012-0249-7
44. Wang, C., Yang, Z., Ivancic, F., Gupta, A.: Whodunit? causal analysis for counterexamples. In: International Symposium on Automated Technology for Verification and Analysis, ATVA 2006. Lecture Notes in Computer Science, vol. 4218. Springer (2006). https://doi.org/10.1007/11901914_9