MEng Individual Project

Imperial College London

Department of Computing

Revisiting Causality of Violations to LTL Formulae in Counterexamples

Author:
Herong Meng

Supervisor:
Dr. Dalal Alrajeh

Second Marker:
Prof. Francesca Toni

Abstract

The application of causality on temporal logics is an active research area, with the potential of enhancing state-of-the-art artificial intelligence paradigms which lacks the ability for temporal reasoning. One field that could benefit from such application is the repair of unrealizability in high-level system specifications, commonly formalized in a subset of linear temporal logic known as generalized reactivity of rank 1. Existing approaches rely on counterexample executions, extracted by automated procedures from these specifications, as guidance for the generation of repairs. By revisiting the work by Beer *et al.* [1] based on the pivotal Halpern and Pearl definition of actual causality, this report proposes a new definition of causality for linear temporal property violations on counterexamples. This new definition is based on an updated version of the Halpern and Pearl definition, and captures causes in a more informative and intuitive manner. Follow this definition, this report describes a new algorithm which computes all linear temporal violation causes in a counterexample with a upper bound on the size of the causes. The algorithm is shown to be sound, complete under bound, and having polynomial time complexity for finite temporal traces. Evaluation on the new algorithm against the baseline causes approximated by Beer *et al.* reveals that, on an assorted dataset of specifications and counterexamples, the causes computed by the new algorithm is able to cover the baseline in most cases, is stronger than the baseline in some cases, and in the few remaining cases unable to fully cover the baseline for well-justified reasons.

Acknowledgements

I, Herong Meng, hereby declare that the contents of this report, as well as the project and its software components on which this report is based, is to the best of my knowledge entirely my own work. I also declare that the intellectual contents of this report, as well as the project and its software components, is the product of my own work, except to the extent that all the assistance received in undertaking this project and sources have been acknowledged.

However, in a counterfactual world without the support of the various people around me, this report would not have been possible either. (Thanks, Lewis!)

First and foremost, I dedicate this work to my supervisor, Dr. Dalal Alrajeh, for her continual academic guidance throughout this project. I deeply appreciate and respect her ability to arrange frequent meeting with me despite her busy schedule. Such strong support is invaluable, especially for someone like me without any prior research experience. I would also like to thank Daniel Ozcan for his assistance on the theoretical component of this project.

In addition, I dedicate this work to the faculty at Imperial College London for the unforgettable experience I've had during my studies. I would like to thank Prof. Francesca Toni and all members of faculty on my marking team for their time on reviewing this project. I express my gratitude to the many friends I've had the honour to acquaint, both within and beyond this college.

I shall particularly dedicate this work to Jiaying Lin for her affectionate care and support. Her encouragement was my greatest support during the most intimidating initial stages of this project.

Finally, I dedicate this work to my parents, Xue Zhou and Jie Meng, for their love, discipline and support. I simply cannot imagine being where I am now without them.

Contents

Chapter 1

Introduction

A major aspiration in computer science has been the automatic synthesis of error-free systems, such as software programs and protocols, from formal specifications [2]. The aim of *reactive synthesis* (or *controller synthesis*) is the automated generation of correct-by-construction systems, *i.e.* controllers, from specifications formalized in temporal logic. Such problem had been shown to be doubly exponential for generic linear-time temporal logic (LTL) [3], and thus has long been deemed impractical.

Nonetheless, recent advances [2] suggests a polynomial-time algorithm for synthesizing specifications in *general reactivity of rank 1* (GR(1)), a subset of LTL with the ability to formalize many patterns of LTL specifications used in industry [4]. Specifications in GR(1) consists of environmental assumptions and controller guarantees. Reactive synthesis tries to build a controller that, when facing any assumptions-compliant environment, also complies with guarantees. However, synthesis fails for *unrealizable* specifications, for which no controller exists.

Unrealizability in GR(1) specifications are often contributed to overly weak assumptions [5] which make the environment overpowered in a game against the controller. This can be due to the inherently challenging nature of engineering an error-free specification at design time [6, 7], which is targeted by the research problems of assumptions refinement or assumptions repair. Numerous counterstrategy-guided approaches has been proposed, including initially template-based algorithms [6, 7], and later interpolation-based [8] as well as the purely symbolic procedures [9].

Alternatively, for systems designed with adaptive capabilities [10, 11], unrealizability can occur to their specifications when they overly weaken the assumptions to adjust for unforeseen scenarios during run time, which has sparked interest in recent academic research [12].

In both cases, a natural and logical approach to fixing unrealizable specifications is to search for sets of sufficiently strong assumptions that is satisfiable, and restricts the environment so that a controller can be re-established. However, such a search problem is defined over a doubly exponential space of possible assumptions [8]. In search of a tractable solution, a paradigm of current academic interest attempts to employ the power of logic-based learning. The paradigm, known as *counterexample-guided inductive synthesis* (CEGIS), consists of (1) an *oracle* (or *teacher*) which generates a *counterexample* at each time which violates the current specification, and (2) a *learner* which tries to understand the source of error in the counterexample, and uses it to strengthen the specification assumption. The oracle and the learner interacts with each other in an iterative fashion, until a realizable specification is found [13, 14]. Most of the works mentioned above regarding unrealizability repairs exhibit this pattern.

So arguably, for a CEGIS-based procedure to converge to a succinctly realizable specification efficiently, it needs to ensure that (1) the oracle always provides concise and insightful counterexamples, while (2) the learner is able to pinpoint the exact *cause(s)* of violation in each counterexample. For the latter point, existing approaches such as [8] chose to represent causes in an indirect manner, in the particular case *i.e.* interpolants.

Curiously, there exists many academic works that directly captures causality, albeit each in their own alternative interpretations. Among them, Halpern and Pearl's formalization of actual causality provides a modern approach to causal reasoning, allowing for the automated generation of diagnostic explanations [15], opening up the gateway for the introduction of causal reasoning in computer science and artificial intelligence.

A subsequent work by Beer *et al.* [1] adapts this notion of causality for explaining trace violation of system properties expressed in LTL. Their work's successful implementation strongly suggests that directly computing the causes of a violation in a counterexample is possible for an optimized learner within the CEGIS architecture, potentially leading to faster convergence and thus greater efficiency.

This project aims to provide a foundation on which causal computation can be integrated with inductive synthesis approaches. It is based on the work of Beer *et al.* [1], with the following main contributions:

1. A new definition of *causes for LTL property violation on a counterexample*, using the modified Halpern definition of actual causality [16], and a new causal representation.

2. An algorithm based on the new definition, FINDVIOLATIONCAUSES($\sigma, \phi, bound$), for computing all causes for violation of LTL formula ϕ on counterexample σ of sizes no greater than *bound*. FINDVIOLATIONCAUSES guarantees soundness, bounded completeness, as well as polynomial time complexity when assuming finite input counterexample traces.

3. Implementation of the algorithm as a new causality checking tool `Caupybara`.

4. Evaluation of the new algorithm, by drawing comparisons between the new algorithm and the baseline approximation algorithm in [1]. For each comparison, causality computations are performed, using both algorithms, on the same dataset of GR(1) specifications and their respective counterexamples taken from [12].

Chapter 2

Background

This chapter will clarify various concepts, introduce formal definitions, and discuss existing work in the academic literature that are closely relevant to this project.

Section 2.1 introduces the formal definitions of LTL syntax and semantics, the LTL$_f$ semantics variant which forms the basis of this project, as well as the negation normal form for LTL. Section 2.2 first defines GR(1) specifications and formulae, then illustrates the concepts of GR(1) games, realizability, as well as unrealizability and its related concepts. Finally, Section 2.3 focuses on the notion of causal reasoning. It first formalizes the HP definition for actual causality, then discusses the Beer *et al.* [1] definition of causality for trace violations.

2.1 Linear-Time Temporal Logic

Linear-time temporal logic (LTL) is an significant branch of temporal logics, the family of logics constructed on top of propositional and predicate logic to formalize the progression of systems throughout time. LTL models system behaviour over time as a linear, potentially infinite sequence of states.

2.1.1 LTL Syntax

As per one of the widely accepted standard definitions [3, 17, 2], a *linear-time temporal logic (LTL) formula* ϕ follows the syntax below, in Backus-Naur form,

$$\phi ::= \ p \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid \ \mathbf{X}\,\phi \mid \phi\,\mathbf{U}\,\phi \mid \ \mathbf{G}\,\phi \mid \ \mathbf{F}\,\phi,$$

where $\top$, $\neg$ and $\wedge$ are Boolean connectives as in propositional logic, p is a propositional atom (*i.e.* variable), and $\mathbf{X}$ (next), $\mathbf{U}$ (until), $\mathbf{G}$ (globally / always) and $\mathbf{F}$ (finally / eventually) are temporal operators. For simplicity, the connectives $\bot$, $\vee$, $\rightarrow$, $\leftrightarrow$ in propositional logic are also used throughout this report, which are derivable from connectives $\top$, $\neg$ and $\wedge$.

2.1.2 Transition Systems

In preparation for presenting the formal definition of LTL semantics, the concept of transition systems is introduced below.

Definition 1 (Transition system). A *labelled transition system* (or simply *transition system*) can be considered as the tuple $T = \langle Q, A, \delta, q_0, \mathcal{V}, \lambda \rangle$ where

- Q is a set of states, where $|Q|$ is finite, and all states in Q are implicit accepting states,
- A is a set of actions,
- $\delta \subseteq Q \times A \times Q$ is a relation defining transition between states, where $(q, \alpha, q') \in \delta$ is written as $q \rightarrow_{\alpha} q'$,
- $q_0 \in Q$ is the initial state,
- $\mathcal{V}$ is a non-empty set of propositional atoms (*i.e.* Boolean variables),
- $\lambda : Q \rightarrow 2^{\mathcal{V}}$ is a labelling function, which associates each state $q \in Q$ with a subset of the variables $\mathcal{P} \subseteq \mathcal{V}$.

An *execution* (or *trace*, *path*) σ in a transition system T has the form $\sigma = q_0\alpha_1 q_1 \alpha_2 q_2 ...$, where for all $i \geq 0$, $q_i \rightarrow_{\alpha_{i+1}} q_{i+1}$ holds. An either finite or an infinite number of states can be visited in any execution σ. For simplicity, a section of any execution σ can be represented as

- $\sigma[i..]$, standing for $q_i \alpha_{i+1} q_{i+1} ...$, and
- $\sigma[i..j]$, standing for $q_i \alpha_{i+1} ... \alpha_j q_j$.

Observation 1. Given an execution σ in any transition system $T = \langle Q, A, \delta, q_0, \mathcal{V}, \lambda \rangle$, σ itself can be considered a subsystem of T, *i.e.* a transition system containing only the execution itself.

For the sake of clarity, this report will denote the transition system representation of σ as $T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$, where Q_σ, δ_σ and λ_σ are the restrictions of their counterparts in T on the execution σ.

2.1.3 LTL Semantics over Transition Systems

Now the semantics of LTL can be formally established.

Definition 2 (LTL semantics). Given a transition system $T = \langle Q, A, \delta, q_0, \mathcal{V}, \lambda \rangle$, an execution $\sigma = q_0 \alpha_1 q_1 \alpha_2 q_2 ...$ in T, and any LTL formulae ϕ, ϕ_1, ϕ_2, the satisfaction relation, denoted by $\vDash$, is defined as:

$$\begin{aligned}
&\sigma \vDash p \text{ iff } p \in \lambda(q_0)\\
&\sigma \vDash \top\\
&\sigma \vDash \neg\phi \text{ iff } \sigma \nvDash \phi\\
&\sigma \vDash \phi_1 \wedge \phi_2 \text{ iff } \sigma \vDash \phi_1 \text{ and } \sigma \vDash \phi_2\\
&\sigma \vDash \mathbf{X}\,\phi \text{ iff } \sigma[1..] \vDash \phi\\
&\sigma \vDash \phi_1\,\mathbf{U}\,\phi_2 \text{ iff } \exists j \geq 0.\ \sigma[j..] \vDash \phi_2 \text{ and } \forall 0 \leq i \leq j.\ \sigma[i..] \vDash \phi_1\\
&\sigma \vDash \mathbf{G}\,\phi \text{ iff } \forall i \geq 0.\ \sigma[i..] \vDash \phi\\
&\sigma \vDash \mathbf{F}\,\phi \text{ iff } \exists i \geq 0.\ \sigma[i..] \vDash \phi.
\end{aligned}$$

And for the transition system T, the satisfaction relation is defined as:

$$T \vDash \phi \text{ iff for every execution } \sigma \text{ in } T, \sigma \vDash \phi.$$

2.1.4 Finite LTL Semantics

One limitation of Definition 2 is that many LTL executions studied in academic literature, especially when as counterexamples, are finite in length [1, 12]. To be able to use them for evaluation, as well as to make more comprehensible analyses, it seems appropriate to introduce a slightly modified set of semantics for *finite LTL*, written as LTL$_f$.

Definition 3 (LTL$_f$ semantics [18, 12])**.** Given a transition system $T = \langle Q, A, \delta, q_0, \mathcal{V}, \lambda\rangle$, a finite execution $\sigma = q_0\alpha_1q_1\alpha_2q_2...\alpha_nq_n$ in T for some $n \in \mathbb{N}$, and any LTL formulae ϕ, ϕ_1, ϕ_2, the finite satisfiability relation, also denoted by $\vDash$, is defined as:

$$\begin{aligned}
&\sigma \vDash p \text{ iff } |\sigma| = 0 \text{ or } p \in \lambda(q_0)\\
&\sigma \vDash \top\\
&\sigma \vDash \neg\phi \text{ iff } \sigma \nvDash \phi\\
&\sigma \vDash \phi_1 \wedge \phi_2 \text{ iff } \sigma \vDash \phi_1 \text{ and } \sigma \vDash \phi_2\\
&\sigma \vDash \mathbf{X}\,\phi \text{ iff } n > 0 \text{ implies } \sigma[1..] \vDash \phi\\
&\sigma \vDash \phi_1\,\mathbf{U}\,\phi_2 \text{ iff } \exists 0 \leq j \leq n.\ \sigma[j..] \vDash \phi_2 \text{ and } \forall 0 \leq i \leq j.\ \sigma[i..] \vDash \phi_1\\
&\sigma \vDash \mathbf{G}\,\phi \text{ iff } \forall 0 \leq i \leq n.\ \sigma[i..] \vDash \phi\\
&\sigma \vDash \mathbf{F}\,\phi \text{ iff } \exists 0 \leq i \leq n.\ \sigma[i..] \vDash \phi.
\end{aligned}$$

In particular, operator $\mathbf{X}$ in this version of LTL$_f$ semantics is interpreted as a “weak next” operator as in [12].

2.1.5 Negation Normal Form for Linear Temporal Logic

Another concept that is particularly helpful for the purposes of this project, especially for time-complexity related results (see Chapter 3), is the negation normal form for LTL.

Definition 4. (Negation normal form for LTL) An LTL formula ϕ is in *negation normal form*, *i.e.* NNF, if all negation operators $\neg$ only appears before proposition atoms in ϕ.

Any LTL formula ϕ can be converted into NNF using logical equivalences, and such conversion does not increase the number of temporal operators in the NNF formula [19]. In addition, such conversion increases the number of Boolean connectives linearly, with the notable exception of the double implication $\leftrightarrow$ having worst-case exponential increase.

Therefore, in this report, we implicitly assume that when discussing the length of any LTL formula ϕ, we are referring to its equivalent form without the $\leftrightarrow$ connective. In such case, from the above assumptions, we shall infer that converting ϕ into NNF can be performed in linear time.

2.2 Generalized Reactivity of Rank 1

A major focus of this projects lies on specifications in *generalized reactivity of rank 1* (GR(1)), a logical subset of LTL. GR(1) formulae expresses properties of a system modelled by two agents, namely the *environment* agent and the *controller* agent. For the set of variables (i.e. propositional atoms) $\mathcal{V}$ in a GR(1) specification,

- the *input* variables $\mathcal{X} \subseteq \mathcal{V}$ are valuated by the environment;
- the *output* variables $\mathcal{Y} = \mathcal{V}/\mathcal{X} \subseteq \mathcal{V}$ are valuated by the controller.

In this section, for any set of propositional atoms $\mathcal{P}$, $\mathbf{X}\,\mathcal{P}$ is used to represent the primed version of all atoms in $\mathcal{P}$, namely

$$\mathbf{X}\,\mathcal{P} = \{\mathbf{X}\,p \mid p \in \mathcal{P}\}.$$

In addition, $B(\mathcal{W})$ is written to denote any propositional logic formula, using “variables” in the set $\mathcal{W}$. Notice that $B(\mathcal{W})$ shall not use any temporal operators, except when $\mathbf{X}\,\mathcal{P} \subseteq \mathcal{W}$ for some set of propositional atoms $\mathcal{P}$; then the primed atoms in $\mathbf{X}\,\mathcal{P}$ are treated as if they are propositional atoms.

2.2.1 GR(1) Specifications and Formulae

The definition of a GR(1) specification is hereby presented as in [2].

Definition 5 (GR(1) specification [2])**.** A GR(1) *specification* can be represented as the pair $\langle \mathcal{E}, \mathcal{S} \rangle$, where

- the environment specification is a tuple of *assumptions* $\mathcal{E} = \langle \varphi^{\mathcal{E}}_{init}, \varphi^{\mathcal{E}}_{inv}, \phi^{\mathcal{E}}_{fair} \rangle$, and
- the controller specification is a tuple of *guarantees* $\mathcal{S} = \langle \varphi^{\mathcal{S}}_{init}, \varphi^{\mathcal{S}}_{inv}, \phi^{\mathcal{S}}_{fair} \rangle$.

Any component with the superscript $^{\mathcal{E}}$ belongs to the assumptions, and any component with the superscript $^{\mathcal{S}}$ belongs to the guarantees. The assumptions and the guarantees, respectively, contain the following:

- The *initial conditions* $\varphi^{\mathcal{E}}_{init}, \varphi^{\mathcal{S}}_{init}$. They take the forms $\varphi^{\mathcal{E}}_{init} \equiv B(\mathcal{X})$, and $\varphi^{\mathcal{S}}_{init} \equiv B(\mathcal{V})$ respectively. They define the initial states (i.e. Boolean valuations of propositional atoms) for the environment and the controller.
- The *invariants* $\varphi^{\mathcal{E}}_{inv}, \varphi^{\mathcal{S}}_{inv}$. They take the forms $\varphi^{\mathcal{E}}_{inv} \equiv \mathbf{G}\, B(\mathcal{V} \cup \mathbf{X}\, \mathcal{X})$, and $\varphi^{\mathcal{S}}_{inv} \equiv \mathbf{G}\, B(\mathcal{V} \cup \mathbf{X}\, \mathcal{V})$ respectively. They restrict the one-step transitional behaviour of the system, in terms of the possible propositional atom valuations for the environment and the controller in the next time step.
- The *fairness conditions* (or *justices*) in $\phi^{\mathcal{E}}_{fair}, \phi^{\mathcal{S}}_{fair}$. They are represented by the conjunctions $\phi^{\mathcal{E}}_{fair} \equiv \bigwedge_{i=1\ldots n} \mathbf{G}\,\mathbf{F}\, B_i(\mathcal{V})$, and $\phi^{\mathcal{S}}_{fair} \equiv \bigwedge_{j=1\ldots m} \mathbf{G}\,\mathbf{F}\, B_j(\mathcal{V})$. They define, for both the environment and the controller, the states (i.e. valuations) which must occur infinitely many times within one execution.

One of the original works on reactive synthesis in GR(1) [2] provides a strict definition for GR(1) formulae. However, there are more recent works [6, 8] that uses an alternative, simpler (non-strict) definition for GR(1) formulae instead. As the latter definition fits better with the interests of the project, this report will adhere to this version and provide it here.

Definition 6 (GR(1) formula)**.** Given any GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$ as in Definition 5, the (non-strict) GR(1) *formula* for $\langle \mathcal{E}, \mathcal{S} \rangle$ is

$$\phi \equiv \phi^{\mathcal{E}} \rightarrow \phi^{\mathcal{S}},$$

where $\phi^{\mathcal{E}} \equiv \varphi^{\mathcal{E}}_{init} \wedge \varphi^{\mathcal{E}}_{inv} \wedge \phi^{\mathcal{E}}_{fair}$ and $\phi^{\mathcal{S}} \equiv \varphi^{\mathcal{S}}_{init} \wedge \varphi^{\mathcal{S}}_{inv} \wedge \phi^{\mathcal{S}}_{fair}$. All boolean connectives are interpreted the same way as for an LTL formula.

Each GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$ uniquely determines one GR(1) formula ϕ, and vice versa. Theses two concepts are referred to interchangeably in this report.

2.2.2 GR(1) Games, Realizability

A GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$ also uniquely defines a GR(1) *game* between the environment and the controller. Game states are the valuations of propositional atoms $\mathcal{V}$ in the specification.

Plays (or *executions*) of the GR(1) game are generated by the environment and the controller interacting with each other. The initial state for the two agents is determined by the initial conditions in $\langle \mathcal{E}, \mathcal{S} \rangle$. Any one-step transition taken in the play must be consistent with the invariants in $\langle \mathcal{E}, \mathcal{S} \rangle$. And at each step of the play,

- the environment starts by selecting one valuation of all propositional atoms in $\mathcal{X}$, with the goal of satisfying the assumptions, while forcing the violation of any guarantees (i.e. leaving the controller unable to satisfy them);
- the controller then selects one valuation of all propositional atoms in $\mathcal{Y}$, with the goal of satisfying (if still possible) the guarantees.

For an arbitrary play of the GR(1) game generated as described above, it is defined to be *winning* for the controller if,

1. the play is infinite, and
2. the winning condition is satisfied by the play, which for GR(1) is the formula $\phi_{win} \equiv \phi^{\mathcal{E}}_{fair} \rightarrow \phi^{\mathcal{S}}_{fair}$.

Intuitively, the controller wins if the play does not terminate, and if the fairness conditions in $\langle \mathcal{E}, \mathcal{S} \rangle$ are all satisfied.

In a GR(1) game, strategies are functions that decides the next move for the agent. In particular, a *strategy for the environment* takes the history of previous states visited in a play and its last state as inputs, then outputs a environment valuation of $\mathcal{X}$. Likewise (but not symmetric), a *strategy for the controller* takes the history of previous states visited in a play, its last state, its current environment valuation of $\mathcal{X}$ as inputs, then outputs a controller valuation of $\mathcal{Y}$.

A *winning strategy for the controller* is one that is winning in all possible plays of the game. And the specification defining the GR(1) game is *realizable* if such strategies for the controller can be found. Finding winning controller strategies in GR(1) specifications defines the problem of *reactive synthesis*, for which an polynomial-time algorithm is given in [2], along with formal definitions for games and strategies.

2.2.3 Unrealizability, Counterstrategies and Counterexamples

Conversely, a GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$ is *unrealizable* if a winning controller strategy cannot be found for the game it defines. In such case there exists a *counterstrategy* [20] that is "winning" for the environment.

A play of the GR(1) game where environment follows the counterstrategy is called a *counterexample* (also referred to in literature as a *countertrace* or a *counterplay*). And in an arbitrary counterexample of the unrealizable GR(1) game defined by $\langle \mathcal{E}, \mathcal{S} \rangle$, it must be the case that

- the play is finite, meaning that either

  – the initial assumption $\varphi^{\mathcal{E}}_{init}$ is satisfied while the initial guarantee $\varphi^{\mathcal{S}}_{init}$ is unsatisfiable, or
  – all states in the play satisfies both invariants $\varphi^{\mathcal{E}}_{inv}$ and $\varphi^{\mathcal{S}}_{inv}$, except the last state in which the invariant guarantee $\varphi^{\mathcal{S}}_{inv}$ is unsatisfiable; or
- the play is infinite, and it satisfies all fairness assumptions in $\phi^{\mathcal{E}}_{fair}$, but at least one fairness guarantee in $\phi^{\mathcal{S}}_{fair}$ is unsatisfiable.

Thus, a counterstrategy can be visualised as a tree-like graph. Its nodes represents the game states (i.e. $\mathcal{V}$-valuations) that has the same history of visited states, where the environment makes the same $\mathcal{X}$-valuations. And its edges represent the possible $\mathcal{Y}$-valuations made by the controller. Each complete path in the graph represents one counterexample.

Notably, Könighofer et al. [20] proposed an efficient algorithm for computing counterstrategies for unrealizable GR(1) specifications, which is nowadays foundation of many assumptions refinement and repair approaches. A more in-depth discussion in Chapter 5 details a variety of related works in this area.

2.3 Causal Reasoning

The concept of causality has been a recurring focus in philosophy since the time of the ancient civilisations. It has always remained as a pivotal component in the fields of scientific research, medicine, law, history among many other studies. For the purposes of the project, this report focuses on what Halpern refers to as *actual causality*, which is only concerned with particular events [16], and not contributions to macroscopic trends.

Giving causality a formal definition has continuously been a massively challenging endeavour. The modern interpretation of actual causality originates from the work of Lewis [21], which introduced the idea of *counterfactual dependencies.* In such interpretation, given distinct events E_c, E_e, one can say that E_c is the cause of E_e if and only if, in an "alternative world" where E_c had not occurred, E_e would not have occurred either.

More recently, Halpern and Pearl formalized actual causality using *structural equations* [15], which has gained popularity in computer science, artificial intelligence and related fields.

2.3.1 Actual Causality by Halpern and Pearl

The *Halpern-Pearl definition of actual causality* (subsequently abbreviated as the *HP definition*) pictures the world as variables and their valuations. Structural equations model potential causal relations between these variables. The key components formalizing the HP definition is given in this section, with reference to the work of Halpern [16], beginning with the formalization of a *causal model.*

Definition 7 (Causal model [15])**.** A *causal model* M can be represented as the pair $\langle \mathcal{S}, \mathcal{F} \rangle$, in which

- the *signature* $\mathcal{S}$ is represented as the tuple $\langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$. All variables in causal model M can be classified into either
  - the set of *exogenous* variables $\mathcal{U}$, valuated solely by factors external to the model,
  - the set of *endogenous* variables $\mathcal{V}$, valuated by the exogenous variables, directly or indirectly,

  such that $\mathcal{U}$ and $\mathcal{V}$ are disjoint. In addition, $\mathcal{R}$ maps every variable in the model $W \in \mathcal{U} \cup \mathcal{V}$ to a set of valuations $\mathcal{R}(W)$ that W can take, where $\mathcal{R}(W) \neq \varnothing$. Namely, $\mathcal{S}$ is a signature of the modelled variables with their possible valuations;

- $\mathcal{F}$ maps every endogenous variable $X \in \mathcal{V}$ to a function F_X,

$$F_X : (\prod_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\prod_{V \in \mathcal{V}/\{X\}} \mathcal{R}(V)) \to \mathcal{R}(X),$$

  and they are, collectively for all endogenous variables, called the *modifiable structural equations*. Namely, F_X takes in a valuation of every variable in $\mathcal{U}$ and $\mathcal{V}$ as input, excluding that of X itself, then outputs valuation for X.

As an example taken from [15], assume causal model M contains $U \in \mathcal{U}$ (exogenous variable) and $X, Y, Z \in \mathcal{V}$ (endogenous variables), and X is valuated by structural equation $F_X(U, Y, Z) = U + Y$, commonly represented by the equation $X = U + Y$. In this case, in the valuation where $U = 1$ and $Y = 4$, then $X = 5$ irrespective of the valuation of Z.

In addition, $M_{X \leftarrow x}$ represents the new causal model derived from M, where $F_X \in \mathcal{F}$ is now set to be $X = x$, and nothing else modified. $M_{\vec{X} \leftarrow \vec{x}}$ is similarly defined for a set (or vector) of variables $\vec{X}$, and a a set (or vector) of valuations $\vec{x}$ in respective order. Now, the definition of a *causal formula* can be formalized in reference to [16].

Definition 8 (Causal formula [16])**.** Consider any signature $S = \langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$. A *causal formula* over signature S can be expressed as $\psi \equiv [Y_1 \leftarrow y_1, ..., Y_k \leftarrow y_k]\varphi$, commonly abbreviated as $[\vec{Y} \leftarrow \vec{y}]\varphi$, in which

- φ is a formula given in propositional logic, over *primitive events* $V = v$ for $V \in \mathcal{V}$ and $v \in \mathcal{R}(V)$, and
- each $Y_i \in \mathcal{V}$ is a unique endogenous variable, and $y_i \in \mathcal{R}(Y_i)$, for all $i = 1, ..., k$.

Intuitively, a causal formula $\psi \equiv [\vec{Y} \leftarrow \vec{y}]\varphi$ has the meaning that, given that variables in $\vec{Y}$ are valuated directly to $\vec{y}$ respectively, effectively overriding their original structural equations, then φ holds.

A context [16] $\vec{u}$ for a causal model M is one unique, legal valuation of all exogenous variables in $\mathcal{U}$ that satisfies all equations in $\mathcal{F}$. Then, given a causal model M, a context $\vec{u}$ over M, and a causal formula ψ, $\langle M, \vec{u} \rangle \vDash \psi$ is written to denote that ψ evaluates to true in M given $\vec{u}$, defined inductively below [16]:

- $\langle M, \vec{u} \rangle \models W = w$ if W valuates to w in the only possible solution of the structural equations in M, under the context $\vec{u}$.

- $\langle M, \vec{u} \rangle \models \varphi$ if the propositional logical formula φ, constructed with primitive events, holds in M given $\vec{u}$.

- $\langle M, \vec{u} \rangle \models [\vec{W} \leftarrow \vec{w}]\varphi$ if $\langle M_{\vec{W} \leftarrow \vec{w}}, \vec{u} \rangle \models \varphi$.

This report is now ready to present the two versions of HP definition below, which forms the theoretical basis of this project.

2.3.2 The Original Halpern-Pearl Definition

The first version of HP definition is referred to as the *original* definition.

Definition 9 (Causality, original definition [15])**.** For any causal model M, context $\vec{u}$, and propositional logic formula φ over primitive events, $\vec{X} = \vec{x}$ is the *actual cause* of φ in $\langle M, \vec{u} \rangle$ when all of below holds.

- AC1.
$$\langle M, \vec{u} \rangle \models \vec{X} = \vec{x}, \text{ and } \langle M, \vec{u} \rangle \models \varphi,$$
$\vec{X} = \vec{x}$ represents some conjunction of the form $X_1 = x_1 \wedge ... \wedge X_n = x_n$. Namely, both $\vec{X} = \vec{x}$ and φ must actually hold.

- AC2. There exist two disjoint sets $\vec{Z}, \vec{W}$ such that $\vec{Z} \cup \vec{W} = \mathcal{V}$ (the set of endogenous variables), $\vec{X} \subseteq \vec{Z}$, and there exist alternative valuations $\vec{x}'$ for $\vec{X}$, $\vec{w}'$ for $\vec{W}$ respectively, where:

  - AC2(a) (*necessity*).
  $$\langle M, \vec{u} \rangle \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}']\neg\varphi.$$
  In words, had $\vec{X}$ not taken the values $\vec{x}$, φ would not have held under the contingency $\vec{W} = \vec{w}'$.

  - AC2(b) (*sufficiency*). Given $\langle M, \vec{u} \rangle \models \vec{Z} = \vec{z}$, then for any arbitrary subset $\vec{Z}' \subseteq \vec{Z}$,
  $$\langle M, \vec{u} \rangle \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}', \vec{Z}' \leftarrow \vec{z}]\varphi.$$
  In words, if $\vec{X}$ takes the alternative valuation $\vec{x}'$, while the variables in the subset $\vec{Z}'$ takes their respective valuations in $\vec{z}$ as in reality, then φ holds even under the contingency $\vec{W} = \vec{w}'$.

- AC3. $\vec{X}$ is minimal, meaning that no strict subset $\vec{X}' \subset \vec{X}$ exist such that AC1 and AC2 still holds by replacing $\vec{X}$ with $\vec{X}'$.

2.3.3 The Modified Halpern-Pearl Definition

The second version is a modification by Halpern on the original definition, *i.e.* Definition 9, which we will refer to as the *modified* definition.

Definition 10 (Causality, modified definition [16])**.** For any causal model M, context $\vec{u}$, and propositional logic formula φ over primitive events, $\vec{X} = \vec{x}$ is the *actual cause* of φ in $\langle M, \vec{u} \rangle$ when all of below holds.

- AC1 as in Definition 9.
- AC2, modified. There exists a subset $\vec{W}$ of the endogenous variables $\mathcal{V}$, and an alternative valuation $\vec{x}'$ for $\vec{X}$, where given $\langle M, \vec{u} \rangle \vDash \vec{W} = \vec{w}$, then

$$\langle M, \vec{u} \rangle \vDash [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}] \neg \varphi,$$

  in which case $\langle \vec{W}, \vec{w}, \vec{x}' \rangle$ forms a *witness.*
- AC3 also as in Definition 9.

In both versions, every primitive event of the form $X_i = x_i$ in the conjunction $\vec{X} = \vec{x}$ is considered a *part* of the cause of formula φ holding in $\langle M, \vec{u} \rangle$. Arguably, there does not exist a most "correct" definition of causality, but both versions of the HP definition mentioned above have been shown to be useful under case analyses [16] as well as in wider academic literature.

2.3.4 Beer *et al.* Causality for LTL Trace Violations

The work by Beer *et al.* [1] aims to provide an intuitive visual explanation for counterexamples. To approach this, they first formalizes the set of singleton causes for a specification failing on one specific counterexample provided, referring to the original HP definition, *i.e.* Definition 9, then gives an algorithm for calculating an approximate for the set of causes. Their subsequent implementations for visualization is beyond the scope of interest for this project.

To apply actual causality to counterexamples of pontentially infinite lengths, they truncates the path to become finite, by replacing looping transitions with a loop indicator. These truncated paths were then interpreted based on the LTL$_f$ semantics[1] given by Defintion 3.

In their work, a cause is defined as the Boolean valuation of a propositional atom p in a state q, which will be referred to in this report as a *singleton cause.* Potential causes are written as pairs $\langle q, p \rangle$. It is also possible to derive

- the counterfactual pair $\overline{\langle q, p \rangle}$ from $\langle q, p \rangle$, by switching the Boolean valuation of atom p in state q,

[1]The *weak LTL semantics* as defined in [22], which is almost identical to Definition 9, is used in their original work.

- the set of counterfactual pairs $\hat{C} = \{\overline{\langle q,p \rangle} | \langle q,p \rangle \in C\}$ from the original set of pairs C, by switching the Boolean valuation of atoms for all pairs in C,
- the execution $\sigma^{\overline{\langle q,p \rangle}}$ from σ by switching the Boolean valuation of p in q on σ,
- the execution $\sigma^{\hat{C}}$ from σ by switching the Boolean valuation of atoms for all pairs in the set C.

And now, the Beer *et al.* definition of the cause for LTL trace violations can be introduced.

Definition 11 (Beer *et al.* Causality for LTL trace violations [1])**.** Given an LTL formula φ, and an execution trace σ such that $\sigma \not\vDash \varphi$. Then the pair $\langle q,p \rangle$ is a *(singleton) cause* for φ being violated on σ, if there exist a set of pairs C where $\langle q,p \rangle \notin C$, such that under LTL$_f$ semantics,

- $\langle q,p \rangle$ is *critical* [23, 1] for $\sigma^{\hat{C}} \not\vDash \varphi$, namely that $\sigma^{\hat{C}} \not\vDash \varphi$ but $\sigma^{\{\overline{\langle q,p \rangle}\} \cup \hat{C}} \vDash \varphi$, and
- $\sigma^{\hat{S}} \not\vDash \varphi$ for any subset $S \subseteq C$.

In words, a pair $\langle q,p \rangle$ is considered a singleton cause if a set of pairs C (known as a *witness set*) can be found, where under the contingency of $\hat{C}$, formula φ can be made to hold in the trace by switching the Boolean valuation of $\langle q,p \rangle$ in the execution (i.e. if $\langle q,p \rangle$ does not occur as in the original execution), but switching the values of C or its subsets alone does not suffice to make φ hold in the trace.

It is possible to see that this definition, a derivation from Definition 9 with the model M and the context $\vec{u}$ replaced with the execution σ, captures the notion of counterfactual dependence. Notice that no structural equations are explicitly mentioned in this adaptation.

However, this report would like to argue that the Beer *et al.* definition contains inadequacies for the automated symbolic diagnosis of counterexamples. In particular:

- Finding all causes following the Beer *et al.* definition is thought to be a Σ_2^P-hard problem [1] in the polynomial hierarchy [24]. As a countermeasure to this, an approximation algorithm has also been given in their work, which can provably calculate an over-approximation in polynomial time "in terms of the length of the LTL formula φ and finite execution σ" [1]. But alternatively, if we consider that ...
- ... the Beer *et al.* definition uses the original HP definition, which in itself is a Σ_2^P-complete problem. Therefore, there exists the possibility of switching over to the modified HP definition which is DP-complete to compute, where DP $=$ NP $\cap$ coNP and is a simpler complexity class compared to Σ_2^P [16].
- Furthermore, the semantics of singleton causes in the Beer *et al.* definition contains ambiguity, especially under the interpretation of modified HP definition. This is demonstrated in details in Section 3.1

For the reasons mentioned above, this project aims at constructing an entirely **new definition** of *causality for counterexample violations*, which (1) is based on the modified HP definition given in Definition 10, and (2) follows a representation of causes that is more informative, intuitive and unambiguous.

Chapter 3

Causes of Violations to LTL Formulae

As promised in Chapter 2, this chapter now formally proposes a new and original definition of causality for LTL property violations on counterexamples. Then, a new algorithm is presented for computing all causes of LTL property violations in a given counterexample.

We begin by providing a motivated example in Section 3.1, to point out inadequacies in the existing causal definition in [1]. Section 3.2 provides the necessary prerequisites for the new definition. Section 3.3 details the new definition, and employ it firsthand on the motivating example. Section 3.4 further showcases the new definition with an additional motivating example. Section 3.5 describes the algorithm in details. And finally, Sections 3.6 and 3.7 show proofs for soundness, bounded completeness and time complexity of the algorithm.

3.1 Motivating Example: Request-Acknowledge System

In addition to the arguments listed in Chapter 2, in order to better justify for the need of a new causal definition, a motivating example is provided below, taken from the very work of Beer *et al.* [1].

Example 3.1 (Request-acknowledge system [1]). Consider a system with a controller that receives requests from two sources in the environment, which we model with the propositional atoms `req1` and `req2`. The controller itself can also issue an acknowledgement to any request, modelled as the atom `ack`.

Now, consider a part of the controller specification, as the LTL formula

$$\phi = \mathbf{G}((\mathtt{req1} \lor \mathtt{req2}) \to \mathbf{X}(\mathtt{ack})),$$

in words, at any timestep during an execution, if one request from either sources is received, the controller must issue an acknowledgement in the next timestep.

Finally, consider a finite counterexample σ, visualised below showing 4 states, and the propositional atoms that are true in each state:

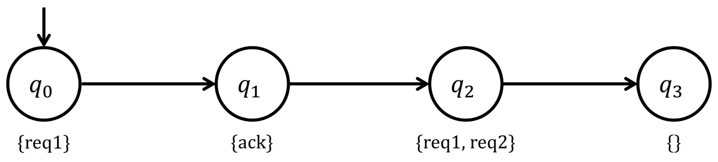


Figure 3.1: Counterexample σ to the request-acknowledge system.

And as we can see, $\sigma \nvDash \phi$ under LTL$_f$ semantics.

According to the Beer *et al.* definition, the singleton causes for the violation of ϕ in σ are $\langle q_2, \texttt{req1}\rangle$, $\langle q_2, \texttt{req2}\rangle$ and $\langle q_3, \texttt{ack}\rangle$. It may not seem intuitive why $\langle q_3, \texttt{ack}\rangle$ is a cause of violation; it is indeed the falsity of `ack`, *i.e.* $\texttt{ack} \notin \lambda_\sigma(q_3)$, that caused the violation of ϕ.

Furthermore, in the absence of additional structural equations, we do not have extra information on events $\langle q_2, \texttt{req1}\rangle$ and $\langle q_2, \texttt{req2}\rangle$ for causal explanations, *e.g.* if one of them is necessary for the violation, or whether there is a precedence for blame. Therefore, it seems unlikely that either $\langle q_2, \texttt{req1}\rangle$ or $\langle q_2, \texttt{req2}\rangle$ alone can be considered a complete cause; had `req1` in state q_2 alone been switched to falsity in a counterfactual execution, `req2` being true would still violate σ, and vice versa.

Theoretically, if adhering to the original HP definition, these singletons would be considered valid causes since we can always find a suitable set $\vec{W}$ of events for each singleton that satisfies the spirit of Definition 9.

However, if we consider the modified HP definition, then the singletons $\langle q_2, \texttt{req1}\rangle$ and $\langle q_2, \texttt{req2}\rangle$ will no longer be causes. Since we can only take set $\vec{W}$ of events that actually occurred in σ, and that no structural equations are available, switching value of `req1` in q_2 alone does not make ϕ hold in $\sigma^{\overline{\langle q_2, \texttt{req1}\rangle} \cup \hat{W}}$ for any $\vec{W}$ (and similarly for `req2`).

3.2 Prerequisites for Formalizing Causality

First and foremost, we shall set a clear definition for the problem that this project aims to tackle, before we can proceed to construct the new definition:

What is considered a cause for the violation of a system property, expressed as a single LTL formula, on a provided counterexample execution, according to the modified HP definition of actual causality?

In addition, for all discussions in this chapter, we shall implicitly assume that when an execution σ is discussed with an LTL formula ϕ for the purposes of causality, they share the same set of propositional atoms $\mathcal{V}$.

3.2.1 Candidate Causal Sets and Singletons

We begin by giving the definition for candidate causal sets, as well as singletons.

Definition 12 (Candidate causal sets, singletons)**.** Given an execution σ, represented as transition system $T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$, a nonempty set $C \subseteq Q_\sigma \times \mathcal{V} \times \{tt, ff\}$ is considered a *candidate causal set* (or simply *candidate set*), which we also write as $\sigma \vDash C$, if for each $\langle q, p, v \rangle \in C$, v is the Boolean value representing whether v is in the valuation of state q in T_σ, *i.e.* $v \equiv p \in \lambda(q)$.

In the case where C is a candidate causal set, each $\langle q, p, v \rangle \in C$ is known as a *singleton.*

We can see that every singleton represents an actual atomic valuation in some state in σ.

3.2.2 Causal Model for Counterexample Violations

We can observe that the Beer *et al.* definition (*i.e.* Definition 11) does not explicitly refer to structural equations, which are core to the HP definitions, which may seem counter-intuitive. However, this report would like to defend this choice by Beer *et al.* and formalize it as the observation given below.

Observation 2. The causal model for violation of some LTL formula ϕ on some execution σ should only contain structural equations that can be directly inferred by σ, and should contain only endogenous variables. Any contexts for the causal model should also be empty.

To justify for the observation, the only information regarding the erroneous behaviour in the system that we are concerned with comes from the execution trace σ.

So crucially, this report considers *all atomic valuations in all states in σ to be independent events, and therefore inadequate to suggest that any dependency relationship exists between them.* This argument has been illustrated by Example 3.1.

It also follows that we would not need to consider external factors that could alter the valuation (*i.e.* labelling function λ_σ of σ) from outside the model. Additionally, since there are no exogenous variables, any context of the causal model can only be empty.

Thus, from Observation 2, we can adapt Halpern's notion of causal models [16] to our problem definition.

Definition 13 (Causal model for counterexample violations)**.** Given an execution σ, represented as transition system $T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$, and an LTL formula ϕ, the causal model for the violation of ϕ on σ would be

$$M_\sigma = \langle \langle \emptyset, Q_\sigma \times \mathcal{V}, \mathcal{R} \rangle, \mathcal{F} \rangle,$$

where for any $\langle q, p \rangle \in Q_\sigma \times \mathcal{V}$, $\mathcal{R}(q, p) = \{tt, ff\}$, and $\mathcal{F}(q, p) = p \in \lambda_\sigma(q)$.

Notice that, with this definition, each execution σ is one-to-one mapped to its model M_σ. Likewise, since only the empty context is allowed, each σ is one-to-one mapped to $\langle M_\sigma, \emptyset \rangle$.

3.2.3 Alteration Traces, Counterfactual Sets, Counterfactual Traces

Every definition of actual causality is arguably founded on some notion of counterfactuals. Here, we proceed by providing the definition of alteration traces.

Definition 14 (Alteration traces)**.** Given an execution σ, ($T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma\rangle$), and some set $S \subseteq Q_\sigma \times \mathcal{V} \times \{tt, ff\}$ (not necessarily a candidate set), the alteration trace σ^S, represented as the new transition system $T_\sigma^S = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma^S\rangle$, where for every $\langle q, p, v\rangle \in Q_\sigma \times \mathcal{V} \times \{tt, ff\}$,

- if $\langle q, p, v\rangle \in S$ and
  - if $v = tt$ then $\lambda_\sigma^S(q) = \lambda_\sigma(q) \cup \{p\}$, or
  - if $v = ff$ then $\lambda_\sigma^S(q) = \lambda_\sigma(q) \setminus \{p\}$,
- otherwise, if $\langle q, p, v\rangle \notin S$ then $\lambda_\sigma^S(q) = \lambda_\sigma(q)$.

Conceptually, the alteration trace σ^S of an execution σ on a set $S \subseteq Q_\sigma \times \mathcal{V} \times \{tt, ff\}$ is created by altering the atomic valuations of σ as specified by S. And implicitly, $\sigma^\emptyset$ is identical to σ. As an example, Figure 3.2 shows an execution trace σ and its alteration traces on sets $S_1 = \{\langle q_1, \texttt{ack}, ff\rangle\}$ and $S_2 = \{\langle q_0, \texttt{req2}, tt\rangle, \langle q_1, \texttt{ack}, tt\rangle\}$.

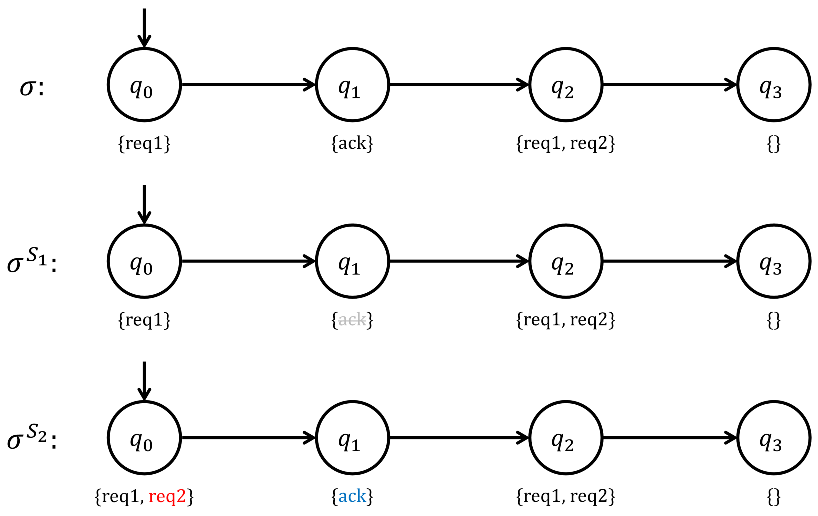


Figure 3.2: σ and its alteration traces on S_1, S_2.

Next, we give the definitions for counterfactual sets and counterfactual traces.

Definition 15 (Counterfactual sets)**.** Given an execution σ, ($T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma\rangle$), and a candidate causal set C taken from σ, the counterfactual set $\hat{C}$ derived from C is one where, for every $\langle q, p, v\rangle \in Q_\sigma \times \mathcal{V} \times \{tt, ff\}$,

$$\langle q, p, v\rangle \in C \text{ iff } \langle q, p, \neg v\rangle \in \hat{C}.$$

In plain words, the counterfactual set $\hat{C}$ taken from a candidate set C is one where all true atomic valuations in C are switched to false, and vice versa. It always follows that $\sigma \not\models \hat{C}$.

And finally, we formalize the concept of counterfactual traces.

Definition 16 (Counterfactual traces)**.** Given any execution σ, and a candidate causal set C taken from σ, the *counterfactual trace* $\sigma^{\hat{C}}$ is simply the alteration trace of σ on the counterfactual set $\hat{C}$.

3.2.4 Temporal and Causal Model Semantics

Since for any execution σ and its corresponding causal model M_σ by Definition 13, σ is one-to-one mapped to $\langle M_\sigma, \emptyset \rangle$, we are now able to bridge the gap between causal model semantics of Halpern and Pearl and linear temporal semantics, by stating that for any $C, S \subseteq Q_\sigma \times \mathcal{V} \times \{tt, \mathit{ff}\}$, we have

$$\begin{aligned} \langle M_\sigma, \emptyset \rangle \models \vec{W} = C &\text{ iff } \sigma \models C, \\ \langle M_\sigma, \emptyset \rangle \models \varphi &\text{ iff } \sigma \models \phi, \\ \langle M_\sigma, \emptyset \rangle \models [\vec{W} \leftarrow S]\varphi &\text{ iff } \sigma^S \models \phi, \end{aligned}$$

where φ is some propositional logic representation of ϕ.

3.2.5 No Witnesses for the Prosecution

As an interesting consequence of Observation 2, when adhering the modified HP definition, the choice of the witness set S_W (originally $\vec{W}$) is no longer relevant for fulfilling AC2$^{(m)}$. Since there are no structural equations involved, and S_W can only be selected from events that actually occurred in execution σ, *i.e.* singletons, then we have

$$\text{for any set } S_W \text{ such that } \sigma \models S_W, \text{ then } \sigma \models \phi \text{ iff } \sigma^{S_W} \models \phi \text{ for any LTL formula } \phi,$$

meaning that under the problem definition given, we *no longer need to select a witness set* for causality of counterexample violations.

3.3 Proposed Definition of Causality

With now all the necessary prerequisites, this report now present the new definition on the causality for LTL property violations on counterexamples, based on modified HP definition, *i.e.* Definiton 10.

Definition 17 (Causality for violations on counterexamples)**.** Given an execution σ ($T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma\rangle$), and an LTL formula ϕ using the propositional atoms in $\mathcal{V}$, a nonempty set $C \subseteq Q_\sigma \times \mathcal{V} \times \{tt, ff\}$ is a *cause* for violation of ϕ on counterexample σ if

- **AC1**. $\sigma \vDash \neg\phi$, and $\sigma \vDash C$ *i.e.* C is a candidate causal set.
- **AC2**$^{(m)}$. $\sigma^{\hat{C}} \vDash \phi$, namely ϕ holds in the counterfactual trace $\sigma^{\hat{C}}$.
- **AC3**. C is minimal, namely for any $C' \subset C$, $\sigma^{\hat{C'}} \vDash \neg\phi$.

With Definition 17, if σ satisfies ϕ, then no causes exist according to definition, since AC1 can not be fulfilled with any candidate causal set C.

A cause can contain either one or more singletons. Where a cause contains multiple singletons, it should be interpreted as a conjunction, *i.e.* the combination of the singletons forms the cause. The number of singletons in a cause is referred to as the *size* of the cause.

Now, we can employ the new Definition 17 firsthand to Example 3.1.

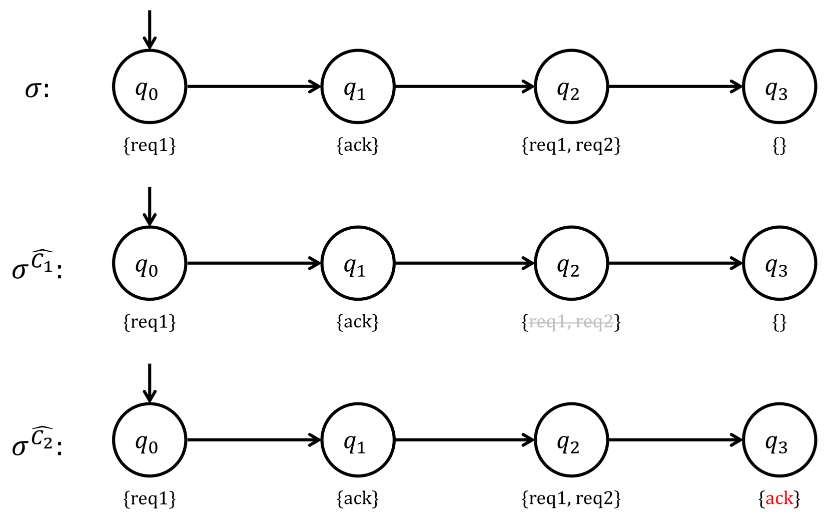


Figure 3.3: σ from Example 3.1 and its counterfactual traces on C_1, C_2.

Given $\phi = \mathbf{G}((\texttt{req1} \vee \texttt{req2}) \rightarrow \mathbf{X}(\texttt{ack}))$, we already know that $\sigma \vDash \neg\phi$. We can also see that, considering the atoms $\mathcal{V} = \{\texttt{req1}, \texttt{req2}, \texttt{ack}\}$, then our candidate causal sets can be selected from the set of all singletons

$$\begin{aligned} S_{q,p,v} = \{&\langle q_0, \texttt{req1}, tt\rangle, \langle q_0, \texttt{req2}, ff\rangle, \langle q_0, \texttt{ack}, ff\rangle,\\ &\langle q_1, \texttt{req1}, ff\rangle, \langle q_1, \texttt{req2}, ff\rangle, \langle q_1, \texttt{ack}, tt\rangle,\\ &\langle q_2, \texttt{req1}, tt\rangle, \langle q_2, \texttt{req2}, tt\rangle, \langle q_2, \texttt{ack}, ff\rangle,\\ &\langle q_3, \texttt{req1}, ff\rangle, \langle q_3, \texttt{req2}, ff\rangle, \langle q_3, \texttt{ack}, ff\rangle\}, \end{aligned}$$

and any $C \subseteq S_{q,p,v}$ would satisfy **AC1**.

Now, consider the two candidate sets $C_1 = \{\langle q_2, \texttt{req1}, \mathit{tt}\rangle, \langle q_2, \texttt{req2}, \mathit{tt}\rangle\}$, $C_2 = \{\langle q_3, \texttt{ack}, \mathit{ff}\rangle\}$. Figure 3.3 shows the counterexample σ and its counterfactual traces on C_1 and C_2.

Since they are both subsets of $S_{q,p,v}$, then they both satisfy **AC1**. In addition, we see that the counterfactual traces $\sigma^{\hat{C_1}}$ and $\sigma^{\hat{C_2}}$ both satisfy ϕ; therefore C_1 and C_2 satisfy $\mathbf{AC2}^{(m)}$.

Finally, we see that C_2 with size 1 is trivially minimal. And to see why C_1 is minimal, consider its subsets of size 1, namely $C_1' = \{\langle q_2, \texttt{req1}, \mathit{tt}\rangle\}$ and $C_1'' = \{\langle q_2, \texttt{req2}, \mathit{tt}\rangle\}$. Then, for their counterfactual traces, $\sigma^{\hat{C_1'}} \vDash \neg\phi$ and $\sigma^{\hat{C_1''}} \vDash \neg\phi$. Hence, both C_1 and C_2 satisfy **AC3**.

In fact, C_1 and C_2 are the only two causes by Definition 17, since now that we know they are causes, we can only check for new candidate sets $C \subseteq S_{q,p,v} \setminus (C_1 \cup C_2)$, and

$$\begin{aligned} S_{q,p,v} \setminus (C_1 \cup C_2) = \{&\langle q_0, \texttt{req1}, \mathit{tt}\rangle, \langle q_0, \texttt{req2}, \mathit{ff}\rangle, \langle q_0, \texttt{ack}, \mathit{ff}\rangle,\\ &\langle q_1, \texttt{req1}, \mathit{ff}\rangle, \langle q_1, \texttt{req2}, \mathit{ff}\rangle, \langle q_1, \texttt{ack}, \mathit{tt}\rangle,\\ &\langle q_2, \texttt{ack}, \mathit{ff}\rangle,\\ &\langle q_3, \texttt{req1}, \mathit{ff}\rangle, \langle q_3, \texttt{req2}, \mathit{ff}\rangle\}, \end{aligned}$$

so no new causes can be identified from the remainder of available singletons.

3.4 Motivating Example: Minepump

To better demonstrate Definition 17, in this section, this report introduces an additional motivating example.

Example 3.2 (Minepump)**.** Consider a system that monitors the conditions in a coal mine, where the controller aims to prevent flooding. The system can monitor whether the water level in the mine is higher than safety threshold (`HighWater`), and whether methane is present (`Methane`). The controller can operate a pump that removes water from the mine (`Pump`).

Now, as part of the formal specification, the LTL formula

$$\phi = \mathbf{G}(\texttt{HighWater} \rightarrow \mathbf{X}\,\texttt{Pump}) \wedge \mathbf{G}(\texttt{Methane} \rightarrow \mathbf{X}(\neg\texttt{Pump})),$$

is the conjunction of two safety properties: the pump should be (1) turned on following the detection of high water level, and (2) turned off following the detection of methane.

Finally, consider a finite counterexample σ, visualised below showing 4 states, and the propositional atoms that are true in each state:

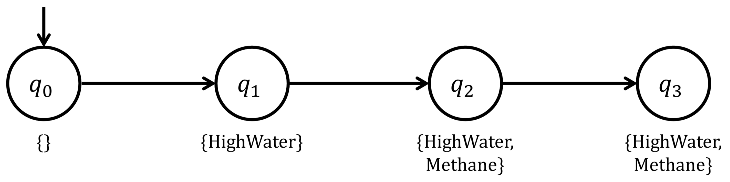


Figure 3.4: Counterexample σ to minepump.

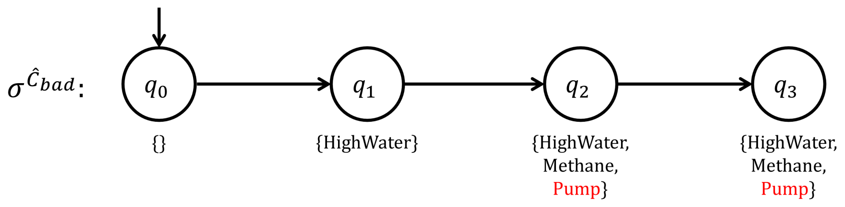
And as we can see, $\sigma \vDash \neg\phi$ under LTL$_f$ semantics. Now, considering the atoms $\mathcal{V} = \{\texttt{HighWater}, \texttt{Methane}, \texttt{Pump}\}$, the set of all singletons are

$$\begin{aligned}
S_{q,p,v} = \{&\langle q_0, \texttt{HighWater}, \mathit{ff}\rangle, \langle q_0, \texttt{Methane}, \mathit{ff}\rangle, \langle q_0, \texttt{Pump}, \mathit{ff}\rangle,\\
&\langle q_1, \texttt{HighWater}, \mathit{tt}\rangle, \langle q_1, \texttt{Methane}, \mathit{ff}\rangle, \langle q_1, \texttt{Pump}, \mathit{ff}\rangle,\\
&\langle q_2, \texttt{HighWater}, \mathit{tt}\rangle, \langle q_2, \texttt{Methane}, \mathit{tt}\rangle, \langle q_2, \texttt{Pump}, \mathit{ff}\rangle,\\
&\langle q_3, \texttt{HighWater}, \mathit{tt}\rangle, \langle q_3, \texttt{Methane}, \mathit{tt}\rangle, \langle q_3, \texttt{Pump}, \mathit{ff}\rangle\},
\end{aligned}$$

and any $C \subseteq S_{q,p,v}$ would satisfy **AC1**.

At first inspection, upon seeing the left-hand side of the conjunction, it would be reasonable to consider $\langle q_2, \texttt{Pump}, \mathit{ff}\rangle$ to be causal. But a causal set with $\langle q_2, \texttt{Pump}, \mathit{ff}\rangle$ alone does not satisfy $\mathbf{AC2}^{(m)}$, so with similar reasoning, it would be tempting to also consider $\langle q_3, \texttt{Pump}, \mathit{ff}\rangle$.

However, the candidate set $C_{bad} = \{\langle q_2, \texttt{Pump}, \mathit{ff}\rangle, \langle q_3, \texttt{Pump}, \mathit{ff}\rangle\}$ is not a cause by Definition 17, since the counterfactual trace $\sigma^{\hat{C}_{bad}}$, visualised in Figure 3.5 violates the right-hand side of the conjunction in ϕ, hence $\mathbf{AC2}^{(m)}$ is not fulfilled.



$\sigma^{\hat{C}_{bad}}$:
q_0
q_1
q_2
q_3
{}
{HighWater}
{HighWater, Methane, Pump}
{HighWater, Methane, Pump}

Figure 3.5: Atlteration trace of σ from Example 3.2 on $\hat{C}_{bad}$.

Instead, consider $C_1 = \{\langle q_2, \texttt{Pump}, \mathit{ff}\rangle, \langle q_2, \texttt{HighWater}, \mathit{tt}\rangle\}$ which is a cause by Definition 17: $C_1 \subseteq S_{q,p,v}$ (**AC1**), the counterfactual trace $\sigma^{\hat{C}_1} \vDash \phi$ under LTL$_f$ semantics ($\mathbf{AC2}^{(m)}$), and neither singleton is a cause on its own (**AC3**).

Alternatively, $C_2 = \{\langle q_1, \texttt{HighWater}, \mathit{tt}\rangle, \langle q_2, \texttt{HighWater}, \mathit{tt}\rangle\}$ is another cause (see Figure 3.6).

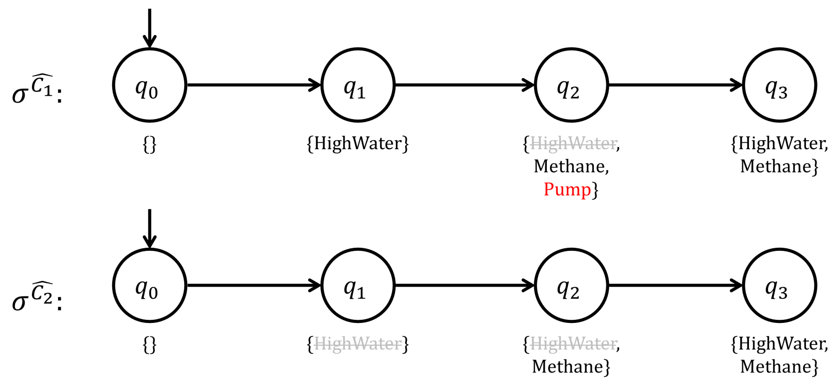


Figure 3.6: Counterfactual traces of σ from Example 3.2 on C_1, C_2.

3.5 Proposed Algorithm for Causality Computation

Definition 17 gives the formal notion of when one set of singletons is considered a cause for counterexample violation. It is not unusual, however, for there to be *multiple* alternative causes on the violation of some property on a single execution. Therefore, it is more practical to build a procedure capable of finding all the causes in one counterexample, which is what this report believes to be the proper way of causal computation.

Hence, this report now presents the new Algorithm 1 below, which returns the complete set of causes S_C, for each input execution σ and LTL property ϕ. The details of Algorithm 1 is shown as below.

Notice that a *bound* is also taken as input to the algorithm, which limits the maximum size of candidate causal set generated (see Lines 7 and 8). Following Definition 12, the total number of candidate causal sets in an execution σ would be exponential to the length of σ. Therefore, upper-bounding the size of candidate sets in Algorithm 1 is a crucial design choice, in order to achieve tractability at the cost of ignoring larger possible causes.

Algorithm 1 also uses two helper subroutines:

- GetSingletons(σ), which returns all singletons present in the execution σ, based on all available propositional atoms in $\mathcal{V}$, and
- GetCounterfactualTrace(σ, C), which returns the counterfactual trace $\sigma^{\hat{C}}$ by Definition 16, assuming the input C is a valid candidate causal set.

3.6 Soundness and Bounded Completeness

Next, we demonstrate that Algorithm 1 is sound, and complete under bound.

Theorem 1. Algorithm 1 is *sound*, and *complete under bound*, in accordance with Definition 17.

Algorithm 1 Finding all causes for counterexample violations.

```
Require: For $\sigma$, the transition system $T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$.
Require: $bound \geq 1$.
Ensure: $S_C$ is the set of all causes of maximum size $bound$.
 1: procedure FindViolationCauses($\sigma$, $\phi$, $bound$)
 2:     $S_C \leftarrow \emptyset$
 3:     if $\sigma \vDash \phi$ then
 4:         return $\emptyset$                          ▷ if $\sigma$ does not violate $\phi$, return $\emptyset$ (AC1)
 5:     end if
 6:     $S_{q,p,v} \leftarrow$ GetSingletons($\sigma$)
 7:     for $size \leftarrow 1, \text{MIN}(|S_{q,p,v}|, bound)$ do
 8:         $candidateCauses \leftarrow \{C \mid C \subseteq S_{q,p,v},\ |C| = size\}$
 9:         for all $C \in candidateCauses$ do
10:             $\sigma^{\hat{C}} \leftarrow$ GetCounterfactualTrace($\sigma, C$)
11:             if $\sigma^{\hat{C}} \vDash \phi$ then
12:                 $S_C \leftarrow S_C \cup \{C\}$                          ▷ (AC2$^{(m)}$)
13:                 $S_{q,p,v} \leftarrow S_{q,p,v} \setminus C$             ▷ ensure minimality (AC3)
14:             end if
15:         end for
16:     end for
17:     return $S_C$
18: end procedure
```

Proof. Consider any LTL formula ϕ and execution σ ($T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$). Take any $bound \in \mathbb{N}$ where $bound$. Then, Assume that an execution of procedure FindViolationCauses($\sigma, \phi, bound$) returns the set S_C.

(*Soundness*). Take arbitrary set $C \in S_C$. We will show that C is a cause according to Definition 1.

AC1. We know that $C \in S_C$, so from Lines 3 and 4 we can infer that $\sigma \vDash \neg\phi$. In addition, from Lines 8 and 9, we see that C is always a subset of $S_{q,p,v}$, which itself only contains singletons. Hence, we can infer that C contains singletons only, and by Definition 12, C is a candidate causal set, *i.e.* $\sigma \vDash C$.

AC2$^{(m)}$. We see C must have been added into S_C at Line 12, which is conditional upon Line 11. Therefore, $\sigma^{\hat{C}} \vDash \phi$ must hold.

AC3. To show that C is minimal, assume for the sake of contradiction there exists $C' \subset C$, such that $\sigma^{\hat{C}'} \vDash \phi$.

We see that $|C'| < |C|$, and since C contains singletons only, we can infer that $|C'|$ also only contains singletons. So the procedure must checked C' before checking C in the for-loop between Lines 7 and 16. Then from assumption and Lines 11, we know Lines 12 and 13 must have been executed by the procedure for C'.

Now, take any $\langle q, p, v \rangle \in C'$. By the arguments above, $\langle q, p, v \rangle$ must have been removed from $S_{q,p,v}$ before the procedure checks for C. Hence, when it eventually gets to generate C on Line 8, we see $\langle q, p, v \rangle \notin C$. But by definition of strict subset, $\langle q, p, v \rangle \in C$ must hold from assumption.

We have reached a contradiction, and our assumption must have been false. Therefore, C must be minimal.

We have shown that **AC1**, **AC2**$^{(m)}$, **AC3** holds for set C. Therefore, by Definition 17, C is a cause for the violation of ϕ on σ. $\square$

(*Bounded completeness*). Take any arbitrary cause C by Definition 17, such that the size of C is no greater than *bound*. We will show that $C \in S_C$ must hold.

Firstly, since **AC1** holds for cause C to exist, we know that $\sigma \vDash \neg\phi$. Therefore, the procedure will not reach Line 4. In addition, $\sigma \vDash C$, hence C must be a candidate set; so for any $\langle q, p, v \rangle \in C$, then $\langle q, p, v \rangle$ must be in the set of all singletons of σ (at Line 6).

Then, by **AC3**, we know for any $C' \subset C$, then $\sigma^{\hat{C'}} \nvDash \phi$, and therefore C' is not a cause by Definition 17. Therefore, we can observe that $\langle q, p, v \rangle \in S_{q,p,v}$ must hold up to the point where the for-loop between Lines 7 and 16 reaches when *size* equals to $|C|$.

So we can infer that when *size* $= |C|$, C must be in the set *candidateCauses* on Line 8, thus would be checked by the procedure in the for-loop between Lines 9 and 15.

Finally, given that **AC2**$^{(m)}$ holds, we know that $\sigma^{\hat{C}} \vDash \phi$, so C must be added to S_C on Line 12, hence $C \in S_C$. $\square$

3.7 Time Complexity under LTL$_f$

It is often desirable to justify for the termination or time complexity of an algorithm. Fortunately, we are able to prove for a *polynomial time complexity* for Algorithm 1, under the premises that input execution trace σ is finite, and under LTL$_f$ semantics.

Before we can show our proof, we present a pivotal result by Fionda and Greco [25] regarding the time complexity of LTL$_f$ semantics.

Lemma 1. For any finite execution σ and LTL formula ϕ in NNF by Definition 4, checking whether $\sigma \vDash \phi$ under LTL$_f$ semantics is feasible in $O(|\sigma|^2 \cdot |\phi|)$ time.

Proof. A proof is given in [25], with the exception of the temporal **U** operator. Nevertheless, we will show that the time complexity shown in [25] still holds with the addition of operator **U**.

Specifically, for any LTL formula $\phi = \psi_1 \, \mathbf{U} \, \psi_2$, we would like to make an extension of the algorithm for computing $sat(v, \sigma)$ in [25] shown in pseudocode below, where v is the leaf node in parse tree for ϕ, $\lambda(v) = \mathbf{U}$, and v_1, v_2 are the left and right children of v in $pt(\phi)$.

```
$sat(v, \sigma) \leftarrow \emptyset$
for $i \leftarrow 0, |\sigma|$ do
    for $j \leftarrow i, |\sigma|$ do
        if $j \in sat(v_2, \sigma)$ then
            $sat(v, \sigma) \leftarrow sat(v, \sigma) \cup \{i\}$
        end if
```

```
            if $j \notin sat(v_1, \sigma)$ then
                break
            end if
        end for
    end for
```

And $\sigma \vDash \phi$ if $0 \in sat(v, \sigma)$. Assume that time complexities for computing the sets $sat(v, \sigma)$, $sat(v_1, \sigma)$ and $sat(v_2, \sigma)$ are represented as functions $T(v, \sigma)$, $T(v_1, \sigma)$ and $T(v_2, \sigma)$ respectively. The sets $sat(v_1, \sigma)$ and $sat(v_2, \sigma)$ are computed recursively in advance.

The double for-loops are traversed $\mathrm{O}(|\sigma|^2)$ times in total, and in each iteration only elementary set operations are performed. Assuming set implementation based on red-black trees [26], each such operations takes $\mathrm{O}(log(T(v, \sigma)))$ time. So we have

$$T(v, \sigma) = T(v_1, \sigma) + T(v_2, \sigma) + O(|\sigma|^2 \cdot log(T(v, \sigma))). \tag{3.1}$$

Now, observe that by substituting the function T with time complexity given in [25], from equality (3.1) we obtain

$$O(|\sigma|^2 \cdot |\phi|) = O(|\sigma|^2 \cdot |\psi_1|) + O(|\sigma|^2 \cdot |\psi_2|) + O(|\sigma|^2 \cdot log(|\sigma|^2 \cdot |\phi|)). \tag{3.2}$$

And since $|\phi| = |\psi_1| + |\psi_2|$, equality (3.2) still holds. Therefore, the time complexity of checking LTL_f satisfiability is $\mathrm{O}(|\sigma|^2 \cdot |\phi|)$ even when considering the **U** operator. □

And with **Lemma 1**, we are able to proceed to our proof.

Theorem 2. Assume that counterexample σ is a finite execution, a finite set of propositional atoms $\mathcal{V}$ is considered, and that LTL_f semantics is used for satisfiability checking of LTL formula ϕ. Then Algorithm 1 runs in polynomial time with respect to the length of σ and sizes of ϕ, $\mathcal{V}$.

Proof. Take any finite counterexample execution σ ($T_\sigma = \langle Q_\sigma, A, \delta_\sigma, q_0, \mathcal{V}, \lambda_\sigma \rangle$), LTL formula ϕ in an equivalent form without the $\leftrightarrow$ connective, and constant $bound \in \mathbb{N}$ where $bound \geq 1$. We will denote the lengths $|\sigma| = m$, $|\phi| = n$, and $|\mathcal{V}| = k$. Assume that LTL_f semantics is used. We will show that the asymptotic time complexity of FindViolationCauses($\sigma, \phi, bound$) in Algorithm 1 is a polynomial of m, n and k.

Firstly, for any checks on $\sigma \vDash \phi$ to have runtime $O(m^2 \cdot n)$, we need to implicitly convert it into NNF, which has time complexity of $O(n)$.

Then, the conditional on Line 3 checks whether $\sigma \vDash \phi$ under LTL_f semantics, which according to **Lemma 1** is in $\mathrm{O}(m^2 \cdot n)$ time. Now, there are two cases:

Case 1. $\sigma \vDash \phi$, then the procedure terminates with runtime in $\mathrm{O}(m^2 \cdot n)$.

Case 2. $\sigma \nvDash \phi$, then our proof continues.

The GetSingletons(σ) subroutine on Line 6 effectively iterates through every state $q \in Q_\sigma$, and evaluates every propositional atom $p \in \mathcal{V}$ to obtain the set $S_{q,p,v}$. Since there are $O(m)$ states and k atoms, then the runtime of GetSingletons(σ) would be in

$O(m \cdot k)$. In addition, the initial size of $S_{q,p,v}$ would also be in $O(m \cdot k)$.

Now, in the outer for-loop between Lines 7 and 16, *size* is upper bounded by *bound*. So, the loop will execute for a constant number of iterations.

On Line 8, generating *candidateCauses*, *i.e.* all candidate sets of a certain size from $S_{q,p,v}$, involves creating a maximum of $\binom{|S_{q,p,v}|}{bound}$ subsets, each of size *bound*, totalling to a upper-bound runtime in $O(\binom{|S_{q,p,v}|}{bound} \cdot bound) = O((m \cdot k)^{bound})$. Thus the inner for-loop between Lines 9 and 15 will run for $O((m \cdot k)^{bound})$ iterations, each time checking a single candidate set C. In each iteration:

- The GetCounterfactualTrace(σ, C) subroutine on Line 10 simply switches the Boolean value of p on q to $\neg v$ for each $\langle q, p, v\rangle \in C$. And since the upper bound of $|C|$ is simply *bound*, this subroutine is considered to run in constant time.

- Next, the conditional on Line 11 checks whether $\sigma \vDash \phi$ under LTL$_f$ semantics with O($m^2 \cdot n$) time (**Lemma 1**). There are also two cases:

  Case 2(a). $\sigma \nvDash \phi$. Then no other computation is made.

  Case 2(b). $\sigma \vDash \phi$. Lines 12 and 13 can be decomposed into basic set operations, each of which shall take a $O(log\,|S|)$ runtime for any set S, assuming set implementation based on red-black trees [26].

  On Line 12, the size of set of all found causes S_C is bounded by the size of the *candidateCauses*, *i.e.* $|S_C|$ has size $O((m \cdot k)^{bound})$, so adding a single C to S_C takes $O(log((m \cdot k)^{bound})) = O(log(m \cdot k))$ in the worst case.

  For Line 13, we know the size of $S_{q,p,v}$ is in $O(m \cdot k)$, so removing no more than a constant *bound* number of elements (in C) from $S_{q,p,v}$ also runs in $O(log(m \cdot k))$ time.

However, when combining the time complexities inside the inner loop (Lines 10 to 14), we can identify $O(m^2 \cdot n)$ as the dominant term, which stands for the time complexity of each run of the inner loop.

And therefore, all runs on the inner loop takes an upper-bound time of $O(m^{bound+2} \cdot n \cdot k^{bound})$, as well as including the outer loop; and so is for the outer loop as $O(m^{bound+2} \cdot n \cdot k^{bound})$ is the dominant term, for a constant number of iterations.

All unspecified operations in the algorithm can be performed in constant time.

So in conclusion, since $bound \geq 1$, the dominant term in all operations of the procedure is therefore $O(m^{bound+2} \cdot n \cdot k^{bound})$, which is the overall asymptotic time complexity of FindViolationCauses($\sigma, \phi, bound$). $\square$

Chapter 4

Evaluation on GR(1) Specifications

With Algorithm 1 presented in Chapter 3, this chapter focuses on its direct evaluation over datasets from the academic literature, in addition to our earlier theoretical analysis. We are using our own implementation of the approximation algorithm proposed by Beer *et al.* [1] for Definition 11 as the baseline for the evaluation.

In particular, Section 4.1 begins by providing an overview of `Caupybara`, the implementation featuring Algorithm 1. Then, Section 4.2 describes the dataset used for evaluation. Section 4.3 details our methodology for evaluation. Finally, Sections 4.4 and 4.5 presents our analysis of the data collected, in quantitative and qualitative aspects.

4.1 Overview of the Implementation

Another major contribution of this project is the code implementation of Algorithm 1 as the `Caupybara` tool. `Caupybara` (a protmanteau of "*cau*sality" and "*capybara*") is a command-line program written in the functional programming language Scala, with its main feature being the causal computation procedure that follows Algorithm 1.

Additionally, since by the time of writing this report, there does not seems to be an openly available implementation of the Beer *et al.* approximation algorithm [1], it has also been realized by `Caupybara` as an alternative causality mode, used for computing baseline causes in our evaluation.

Apart from the causality algorithms, `Caupybara` also includes custom implementations of an LTL parser, a execution trace parser, as well as a custom LTL$_f$ satisfiability checker for finite traces based the work of Fionda *et al.* [25]. Theoretically, it would be possible to substitute the LTL$_f$ checker with other state-of-the-art tools, such as `Aalta` [27] or `BLACK` [28] which promises fast computation for finite and infinite LTL semantics.

There are a few differences between Algorihm 1 and its implementation in `Caupybara`. Below are some crucial difference(s) that this report deemed necessary to highlight.

- Instead of using set implementations based on red-black trees, which is available as

`TreeSet` in Scala, the implementation in `Caupybara` uses the conventional `HashSet` implementation, taking advantage of its $O(1)$ average-case time complexity for elementary set operations [29]. However, due to its theoretical $O(n)$ worst-case complexity, we have refrained from using it during our proofs.

- Some minor changes has been made to the LTL$_f$ satisfiability checker by Fionda *et al.* in order to fit our definition of LTL$_f$ semantics (see Definition 3).

4.2 The Dataset by Buckworth *et al.*

For our evaluation, we are using the dataset presented by Buckworth *et al.* [12], hereafter refer to as "the dataset". It is derived from 6 realizable GR(1) specifications frequently discussed in relevant academic literature on case studies (Arbiter, Genbuf, Lift, Minepump, Traffic Single and Traffic Updated) in the following way:

1. For each original specification, Buckworth *et al.* performs various mutations by strengthening its assumptions, creating up to 5 unique and unrealizable new specifications, which are then added to the dataset.

2. And for each unrealizable specification in the dataset, Buckworth *et al.* computes up to 10 unique counterexample executions from a generated counterstrategy.

Selected statistics on the specifications and counterexample traces in each case study is shown in Table 4.1.

The evaluations of this project will focus on (1) the unrealizable mutated specifications and (2) their respective counterexamples.

Case study	# spec.	# traces	Trace length mean	# asm. mean	# gar. mean	# violated asm. mean
Arbiter	1	10	1.6	1.0	4.0	1.0
Genbuf	5	5	1.4	27.2	79.4	1.4
Lift	5	10	2.1	7.0	10.0	1.0
Minepump	5	25	2.1	1.0	2.0	1.0
Traffic Single	5	50	2.3	2.8	1.4	1.0
Traffic Updated	5	50	1.8	4.4	3.4	2.4
All	26	150	2.0	4.0	5.5	1.5

Table 4.1: Selected statistics on case studies [12].

4.3 Evaluation Methodology

Since Algorithm 1 is based on a newly proposed definition of causality, and due to the nature of actual causality, it is rather difficult give a direct measure of quality of the causes computed by the new tool. Nevertheless, it is possible to draw some comparisons between the causes computed by Algorithm 1 and the *baseline* approximation algorithm by Beer *et* *al.* [1] due to the proximity of the formats of their causes. And more importantly, these would be worthwhile comparisons since Algorithm 1 aims to provide an improvement over the baseline.

In particular, for each unrealizable specification in the dataset, the causes for its violation on each of its counterexample trace is computed on fragments of the specification, classified into three *categories*:

1. its individual assumptions,
2. its individual guarantees,
3. the conjunction of its assumptions.

For each category, causes are computed and collected by both Algorithm 1 and the baseline, made possible by the `Caupybara` tool. Example 4.1 shows how violations causes are categorized, and then computed for a specification.

Example 4.1. (Causal computation for a Traffic Single specification) An unrealizable specification for the Traffic Single case study has the following assumptions and guarantees.

Assumptions:

$$\begin{aligned}
\varphi_1^{\mathcal{E}} &= \mathbf{G}(\texttt{green} \vee \neg\texttt{car}),\\
\varphi_2^{\mathcal{E}} &= \mathbf{G}(\texttt{police} \vee \mathbf{X}(\neg\texttt{car}) \vee \neg\texttt{green}),\\
\varphi_3^{\mathcal{E}} &= \mathbf{G}\,\mathbf{F}(\neg\texttt{police}).
\end{aligned}$$

Guarantee:

$$\varphi_1^{\mathcal{S}} = \mathbf{G}(\texttt{car} \rightarrow \mathbf{F}(\texttt{green})).$$

And there are 6 counterexamples, $\sigma_1, \sigma_2 ... \sigma_6$. Then the category of causes are classified as:

- Category 1: union of unique causes on σ_i for $i \in [1, 6]$, for violations of assumptions $\varphi_1^{\mathcal{E}}$, $\varphi_2^{\mathcal{E}}$ and $\varphi_3^{\mathcal{E}}$ individually.
- Category 2: union of causes on σ_i for $i \in [1, 6]$, for violations of guarantee(s) $\varphi_1^{\mathcal{S}}$ (individually).
- Category 3: union of causes on σ_i for $i \in [1, 6]$, for violation of the conjunction of assumptions, *i.e.* $\varphi_1^{\mathcal{E}} \wedge \varphi_2^{\mathcal{E}} \wedge \varphi_3^{\mathcal{E}}$.

Before analyzing the differences between Algorithm 1 and the baseline, the definitions of two key evaluation metrics, strength and coverage, is given below.

Definition 18 (Strength and coverage)**.** Let S_{Beer} be the set of causes computed by the baseline, and S_{Meng} be the set of causes computed by Algorithm 1 on the same set of specifications and evaluations. Take any baseline cause $\langle q, p \rangle \in S_{\text{Beer}}$.

We say that $\langle q, p \rangle$ is *covered by* S_{Meng} if $\{\langle q, p, v \rangle\} \in S_{\text{Meng}}$ for some $v \in \{tt, ff\}$, *i.e.* $\langle q, p, v \rangle$ alone is a computed cause by Algorithm 1.

Then, the *coverage* of S_{Beer} by Algorithm 1 is defined as the fraction

$$\text{coverage} = \frac{\#\text{ of causes in } S_{\text{Beer}} \text{ covered by } S_{\text{Meng}}}{\text{Total } \#\text{ of causes in } S_{\text{Beer}}}.$$

Finally, we define some $C \in S_{\text{Meng}}$ to be *stronger* than $\langle q, p \rangle$ if $\langle q, p, v \rangle \in C$ for some $v \in \{tt, ff\}$, and size of C is greater than 1, *i.e.* $\langle q, p, v \rangle$ is a part of some computed cause by Algorithm 1.

For our *quantitative* analysis, we make a numeric comparison of the numbers of causes identified belonging in each category, between Algorithm 1 and the baseline. Then, we will provide additional statistics for sizes of causes computed by Algorithm 1, including their minimum, maximum and mean sizes. Finally, we will compute the coverage of the sets of baseline causes by Algorithm 1.

And for our *qualitative* analysis, we will take representative and interesting cases from the dataset, and try to elaborate the differences in the causes computed by Algorithm 1 and the baseline. In particular, we will focus on cases where the coverage of baseline causes by Algorithm 1 is less than 100%, hereafter referred to as cases of *imperfect coverage*. In such cases, we will try to see if stronger causes were found by Algorithm 1, and provide justification if not.

4.4 Quantitative Analysis

The quantitative results are shown in Tables 4.2, 4.3 and 4.4, for each categories of causes we identified.

In the Arbiter, Lift and Minepump case studies, Algorithm 1 is able to provide 100% coverage of baseline causes, but without discovering new causes, across all categories. This was possibly due to the simplistic nature of the specifications in these case studies, or the limited lengths of their counterexamples, or a combination of both.

In Traffic Single, baseline causes in Category 1 are 100% covered by Algorithm 1 for all specifications, some with additional causes found; but for some of its specifications, there are Category 2 and 3 baseline causes that are not covered by Algorithm 1. And for some of the specifications within the Traffic Updated case study, there are Category 1 and 3 baseline causes that are not covered by Algorithm 1. Finally, among all categories in Genbuf, we found cases of imperfect coverage among all three categories.

Case study	Specification	Baseline	Algorithm 1				**Coverage**
		# causes	# causes	Size min	Size max	Size mean	(%)
Arbiter	dropped0	2	2	1	1	1.0	100
Genbuf	dropped10	1	1	1	1	1.0	100
	dropped107	4	3	1	2	1.333	50
	dropped115	1	1	1	1	1.0	100
	dropped118	1	1	1	1	1.0	100
	dropped122	5	5	1	1	1.0	100
Lift	dropped0	2	2	1	1	1.0	100
	dropped1	3	3	1	1	1.0	100
	dropped10	3	3	1	1	1.0	100
	dropped11	4	4	1	1	1.0	100
	dropped13	1	1	1	1	1.0	100
Minepump	dropped0	2	2	1	1	1.0	100
	dropped1	4	4	1	1	1.0	100
	dropped2	3	3	1	1	1.0	100
	dropped4	2	2	1	1	1.0	100
	dropped10	3	3	1	1	1.0	100
Traffic Single	dropped0	5	5	1	1	1.0	100
	dropped1	5	7	1	1	1.0	100
	dropped2	6	7	1	1	1.0	100
	dropped10	6	7	1	1	1.0	100
	dropped11	6	6	1	1	1.0	100
Traffic Updated	dropped3	2	2	1	1	1.0	100
	dropped7	3	9	2	2	2.0	0
	dropped9	1	1	1	1	1.0	100
	dropped11	3	9	2	2	2.0	0
	dropped12	2	2	1	1	1.0	100

Table 4.2: Category 1 causes (individual assumptions).

Case study	Specification	Baseline	Algorithm 1				**Coverage**
		# causes	# causes	Size min	Size max	Size mean	(%)
Arbiter	dropped0	6	6	1	1	1.0	100
Genbuf	dropped10	0	0	0	0	0.0	-
	dropped107	0	0	0	0	0.0	-
	dropped115	16	8	1	1	1.0	50
	dropped118	2	2	1	1	1.0	100
	dropped122	2	1	2	2	2.0	0
Lift	dropped10	3	3	1	1	1.0	100
	(*all else*)	0	0	0	0	0.0	-
Minepump	(*all*)	0	0	0	0	0.0	-
Traffic Single	dropped0	5	4	1	1	1.0	80
	dropped1	4	4	1	1	1.0	100
	dropped2	2	2	1	1	1.0	100
	dropped10	4	4	1	2	1.5	50
	dropped11	0	0	0	0	0.0	-
Traffic Updated	(*all*)	0	0	0	0	0.0	-

Table 4.3: Category 2 causes (individual guarantees).

Case study	Specification	Baseline	Algorithm 1				**Coverage**
		# causes	# causes	Size min	Size max	Size mean	(%)
Arbiter	dropped0	2	2	1	1	1.0	100
Genbuf	dropped10	1	0	0	0	0.0	0
	dropped107	4	0	0	0	0.0	0
	dropped115	1	1	1	1	1.0	100
	dropped118	1	1	1	1	1.0	100
	dropped122	5	0	0	0	0.0	0
Lift	dropped0	2	2	1	1	1.0	100
	dropped1	3	3	1	1	1.0	100
	dropped10	3	3	1	1	1.0	100
	dropped11	4	4	1	1	1.0	100
	dropped13	1	1	1	1	1.0	100
Minepump	dropped0	2	2	1	1	1.0	100
	dropped1	4	4	1	1	1.0	100
	dropped2	3	3	1	1	1.0	100
	dropped4	2	2	1	1	1.0	100
	dropped10	3	3	1	1	1.0	100
Traffic Single	dropped0	5	8	1	2	1.375	100
	dropped1	5	8	1	2	1.5	80
	dropped2	6	8	1	2	1.375	83
	dropped10	6	9	1	2	1.333	83
	dropped11	6	7	1	2	1.429	67
Traffic Updated	dropped3	2	2	1	1	1.0	100
	dropped7	3	9	2	2	2.0	0
	dropped9	1	1	1	1	1.0	100
	dropped11	3	9	2	2	2.0	0
	dropped12	2	1	1	1	1.0	50

Table 4.4: Category 3 causes (conjunction of assumptions).

To summarise, a total of 78 comparisons are drawn between causes found by Algorithm 1 and the baseline among all categories. All baseline causes are covered in 61 of these comparisons, 4 of which with additional causes found by Algorithm 1. A case of imperfect converge was found in 17 of the 78 comparisons, where

- 6 cases were found in Genbuf,
- 6 cases were found in Traffic Single, and
- 5 cases were found in Traffic Updated.

Alternatively, we can classify cases of imperfect coverage by category, where

- 3 cases occurred for Category 1 causes,
- 4 cases occurred for Category 2 causes, and
- 10 cases occurred for Category 3 causes.

4.5 Qualitative Analysis

In this section, we would like to focus on the 17 cases of imperfect coverage revealed in Section 4.4. Fortunately, for each of these cases, it is possible to find one of the three reasons for imperfect coverage, namely (1) Algorithm 1 found stronger causes than baseline (in 10 cases), (2) due to a better interpretation of violations in fairness properties (in 2 cases) and (3) due to conflicts in conjuncts eliminating candidate causes (in the remaining 5 cases).

4.5.1 Stronger Causes Found by Algorithm 1

Upon closer inspection, we can identify 10 cases of imperfect coverage where stronger causes were computed by Algorithm 1. They include all 3 cases for Category 1 baseline causes, 2 cases for Category 2 causes in Genbuf (dropped122) and Traffic Single (dropped10), as well as 5 cases for Category 3 causes in Traffic Single (dropped1, dropped2, dropped10) and Traffic Updated (dropped7, dropped11).

These cases represent the desirable improvements on causal semantics in Definition 17 over the baseline, demonstrated by Example 3.1 and its follow-up in Chapter 3. They are also exemplified by Example 4.2 taken from the Traffic Single (dropped10) specification.

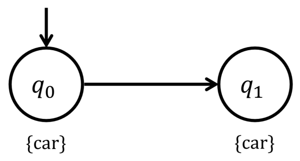


Figure 4.1: σ for Example 4.2.

Example 4.2. Consider the GR(1) guarantee $\varphi^{\mathcal{S}} = \mathbf{G}(\texttt{car} \rightarrow \mathbf{F}(\texttt{green}))$, and its counterexample trace σ shown in Figure 4.1.

The set of causes found by the baseline is

$$S_{\text{Beer}} = \{\langle q_0, \texttt{car}\rangle, \langle q_0, \texttt{green}\rangle, \langle q_1, \texttt{green}\rangle\},$$

and the set of causes found by Algorithm 1 is

$$S_{\text{Meng}} = \{\{\langle q_0, \texttt{car}, tt\rangle, \langle q_1, \texttt{car}, tt\rangle\}, \{\langle q_0, \texttt{green}, ff\rangle, \langle q_1, \texttt{car}, tt\rangle\}, \{\langle q_1, \texttt{green}, ff\rangle\}\}.$$

So for each cause in S_{Beer}, there is a corresponding stronger cause in S_{Meng}.

4.5.2 Improved Interpretation of Fairness Violations

In at least 2 of the cases of imperfect coverage, namely for Category 2 baseline causes in Genbuf (dropped115) and Traffic Single (dropped0), causes computed by Algorithm 1 instead reflects a more accurate interpretation of fairness property violations on finite traces. Consider Example 4.3 taken from Traffic Single (dropped0).

Example 4.3. Take the GR(1) fairness guarantee $\varphi^{\mathcal{S}} = \mathbf{G}\,\mathbf{F}(\neg\texttt{car})$, and its counterexample trace σ shown in Figure 4.2.

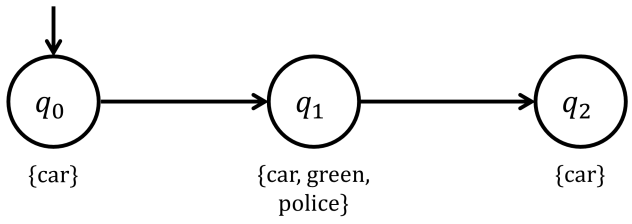


Figure 4.2: Counterexample σ for Example 4.3.

The set of causes found by the baseline is $S_{\text{Beer}} = \{\langle q_0, \texttt{car}\rangle, \langle q_1, \texttt{car}\rangle, \langle q_2, \texttt{car}\rangle\}$; the set of causes found by Algorithm 1 is simply $S_{\text{Meng}} = \{\{\langle q_2, \texttt{car}, tt\rangle\}\}$.

To see why $C_{bad} = \{\langle q_0, \texttt{car}, tt\rangle\}$ is not a cause by Definition 17 under LTL$_f$ semantics, take the counterfactual trace $\sigma^{\hat{C}_{bad}}$ (as shown in Figure 4.3). But then $\sigma^{\hat{C}_{bad}}[1..] \models \neg\, \mathbf{F}(\neg\texttt{car})$, therefore $\sigma^{\hat{C}_{bad}} \models \neg\, \mathbf{G}\,\mathbf{F}(\neg\texttt{car})$ and C_{bad} does not fulfil $\mathbf{AC2}^{(m)}$. Similar arguments can be used against suggesting candidates $\{\langle q_1, \texttt{car}, tt\rangle\}$ and $\{\langle q_0, \texttt{car}, tt\rangle, \langle q_1, \texttt{car}, tt\rangle\}$ as causes.

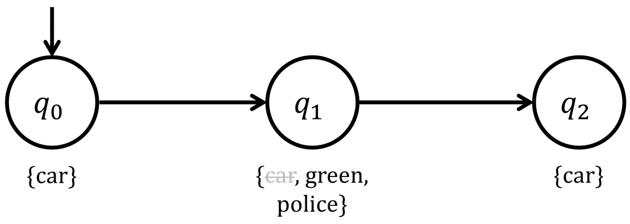


Figure 4.3: The counterfactual trace $\sigma^{\hat{C}_{bad}}$.

Even though a finite counterexample may not capture fairness violations very precisely, it is arguably more justified for its cause to point to events occurring at the far end of the execution.

4.5.3 Imperfect Coverage due to Conflicting Conjuncts

Taking conjunction of temporal formulae that restricts or conflicts each other could eliminate causal candidates for Definition 17, as demonstrated earlier by Example 3.2. This is the reason of imperfect coverage, specifically for Category 3 baseline causes, on the remaining 5 cases involving Genbuf (dropped10, dropped107, dropped122), Traffic Single (dropped11) and Traffic Updated (dropped12) specifications. Such cases can be best demonstrated by an example taken from Genbuf (dropped10), as shown in Example 4.4.

Example 4.4. The Genbuf (dropped10) specification $\langle \mathcal{E}, \mathcal{S}\rangle$ in GR(1) contains 28 assumptions $\varphi_0^{\mathcal{E}}, \varphi_1^{\mathcal{E}}, ..., \varphi_{27}^{\mathcal{E}}$. Among these assumptions are

$$\begin{aligned}
\varphi_5^{\mathcal{E}} &= \neg\texttt{rtob\_ack0}, \\
\varphi_{19}^{\mathcal{E}} &= \mathbf{G}(\texttt{btor\_req0}), \\
\varphi_{26}^{\mathcal{E}} &= \mathbf{G}\,\mathbf{F}((\texttt{btor\_req0} \wedge \texttt{rtob\_ack0}) \vee (\neg\texttt{btor\_req0} \wedge \neg\texttt{rtob\_ack0})).
\end{aligned}$$

And consider the counterexample σ for $\langle \mathcal{E}, \mathcal{S} \rangle$ as shown in Figure 4.4.

The set of baseline causes for the violation of $\varphi_0^{\mathcal{E}} \wedge \varphi_1^{\mathcal{E}} \wedge ... \wedge \varphi_{27}^{\mathcal{E}}$ is $S_{\text{Beer}} = \{\langle 0, \texttt{btor\_req0} \rangle\}$. However, no causes were found by Algorithm 1.

We see that for any cause C for the violation of $\varphi_0^{\mathcal{E}} \wedge \varphi_1^{\mathcal{E}} \wedge ... \wedge \varphi_{27}^{\mathcal{E}}$ by Definition 17 to exist, it will need to include the singleton $\langle 0, \texttt{btor\_req0}, \mathit{ff} \rangle$ in order to satisfy $\varphi_{19}^{\mathcal{E}}$. Now, in order to satisfy $\varphi_{26}^{\mathcal{E}}$, then we also need to include $\langle 0, \texttt{rtob\_ack0}, \mathit{ff} \rangle$ in C.

But then, the counterfactual trace $\sigma^{\hat{C}}$ will violate the initial assumption $\varphi_5^{\mathcal{E}}$. Therefore, no causes exist for the violation of conjunction on σ.

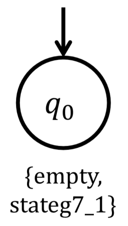


Figure 4.4: σ for Example 4.4.

Chapter 5

Related Works

This project aims to lay a solid foundation for the interrelation between temporal causality and GR(1) unrealizability repairs. Therefore, this chapter provides a summary of relevant literature in theses two areas, and also discuss who they relate to the contents of this report.

Section 5.1 briefly discusses existing academic works focusing on causality for LTL, as well as how their approaches differ from this project. Section 5.2 is a broad discussion about counterexamples and unrealizability, including existing approaches to assumptions refinement / repair, and how unrealizability affects adaptive systems.

5.1 Causality and Linear Temporal Logic

In Chapter 2, we have discussed in great details about the actual causality by Halpern and Pearl, as well as the work of Beer *et al.* on explaining temporal violations. Subsequent research has further explored the application of actual causality to linear temporal logic, categorizing approaches into two primary streams based on their causal representations, each referred as the *event-order logic* and *hyperproperty* streams in this report.

5.1.1 Event-Order Logic Temporal Causality

The event-order logic (EOL) stream of temporal causality originates from the work of Leitner-Fischer and Leue [30] which focuses on the development of an algorithm for causal computation in complex system models.

In their work, in order to create an abstract and more compact representation of counterexample executions, the concept of *events* are used to represent state transitions in these executions. Then, the events on executions are represented by EOL formula, which provides an order-preserving logical representation of the sequence of events. Finally, a definition of causality for property violations is provided, giving causes as EOL formula, based on the original HP definition [15] with the addition of non-occurrence and order rules.

An important property of EOL is the possibility to establish subset and superset relationship between EOL formulae. With this property, an graph-based algorithm is introduced for computing property violation causes, operating on *subset graphs*, in which counterexamples are placed in a lattice of subset relationships of their EOL representation. Subsequently, this algorithm is implemented as the `SpinCause` tool, and has been demonstrated to be reasonably efficient for causal computation on industrial specifications. However, in their original work, they are not able to provide proofs for the soundness and completeness of the algorithm.

Subsequently, extensions to the work by Leitner-Fischer and Leue can be found in the academic literature. A. Beer *et al.* [31] proposes a new symbolic algorithm for computing causes by the Leitner-Fischer and Leue definition based on bounded model checking. Improved computational efficiency has been observed in this new algorithm, and additionally, proofs of its soundness and bounded completeness had been provided.

Both [30] and [31] are limited to finite counterexample traces. Caltais *et al.* [32] then suggests in their work for an update to EOL, *i.e.* extended EOL, with the ability to represent lasso-shaped infinite counterexamples, as well as an updated causal definition in extended EOL. However, by the time of writing this report, it is believed that no algorithm has been formally designed and implemented for causal computation in extended EOL.

In comparison with Definition 17, the causal definition by Leitner-Fischer and Leue, as well as its updated version, is still based on the more complex original HP definition. In addition, it appears that the EOL representation lacks flexibility when being extended to support infinite counterexamples; by contrast, an adaptation of Algorithm 1 to support infinite LTL semantics appears to be trivial, at least in theory, by simply using an infinite LTL satisfiability checker.

5.1.2 Hyperproperty-Based Temporal Causality

The theoretical foundation of the hyperproperty stream of temporal causality is the work of Coenen *et al.* [33] focusing on reactive systems. In their work, causality for observed effects occurring on lasso-shaped executions is represented in HyperQPTL, and extension of LTL with quantification over propositional atoms as well as execution paths. Then, the definition of causes for an effect observed on an execution is provided based on the modified HP definition [16].

Following this foundational literature, a more recent work by Beutner *et al.* [34] presents the design of an algorithm, implemented as the `CATS` tool, which checks if a given candidate is the cause for some temporal effect according to the definition by Coenen *et al.* . In addition, `CATS` can also search for additional causes when the user provides with a causal sketch.

In contrast to `Caupybara` and the tool built in [31], `CATS` is not able to perform exhaustive computations that attempts to find all causes in a given trace. Additionally, there is a separation of propositional atoms into input and output variables in the Coenen *et al.* definition, used exclusively by causes and effects respectively, whereas no such restriction exists for Definition 17.

5.2 Counterexamples and Unrealizability in GR(1)

Unrealizability in GR(1) specifications is often due to weaknesses in their environmental assumptions [5] which give the environmental agents too much power, such that they can form counterstrategies to force unsatisfiability of controller guarantees. In such cases, the desired outcome of fixing unrealizability is a new specification with sufficiently yet minimally strong assumptions, limiting the freedom of environmental agents to the degree that they can not reproduce any counterstrategies, while not introducing unnecessary assumptions that overly restrict environmental behaviors.

Such a search problem in nature has been defined over spaces of possible assumptions doubly exponential in the number of propositional atoms [8], which became a major obstacle for efficient repairs of unrealizability. To avoid traversing through such space directly, many recent approaches to fixing unrealizable assumptions are counterstrategy guided, i.e. relying on counterstrategies to highlight the root causes of unrealizability. These approaches are pioneered by the work on specification debugging by Könighofer *et al.* [20], where a new algorithm had been design has been introduced to extract counterexamples from counterstrategies, initially intended to illustrate the sources of unrealizability as visual traces for human engineers.

5.2.1 Assumptions Refinement and Assumptions Repair

To design an error-free controller specification, especially for a complex system, is often considered a difficult task [6, 7]. The *assumptions refinement* and *assumptions repair* problems, often used interchangeably, are concerned with helping engineers correct GR(1) specifications at the initial design stage. In particular, an assumptions refinement or repair procedure taking an unrealizable but satisfiable specification, usually formalized by the GR(1) formula $\phi^{\mathcal{E}} \rightarrow \phi^{\mathcal{S}}$, then computes a set of supplementary assumptions $\{\varphi_1^{\mathcal{E}}, \varphi_2^{\mathcal{E}}, ...\}$, such that the strengthened formula $\phi^{\mathcal{E}} \wedge (\varphi_1^{\mathcal{E}} \wedge \varphi_2^{\mathcal{E}} \wedge ...) \rightarrow \phi^{\mathcal{S}}$ becomes both realizable and satisfiable.

Such an approach has been made possible by the initial work of Könighofer *et al.* , which allows for efficient generation of counterexamples. Following the initial advancement, Li *et al.* [6] proposes using templates, and taking user scenarios as input, to first build an initial set of candidate assumptions, then iteratively compute additional assumptions from counterstrategies in a process referred to as mining. The use of templates decreases the chance of traversing into uninteresting parts of the assumptions search space, compared to a hypothetical, purely brute-force method. Similarly, Alur *et al.* in their work [7] suggests generating candidate LTL formula templates known as patterns, which can be satisfied by all executions in a counterstrategy. However, this approach requires a human engineer to pick the subset of variables considered for pattern generation, thus the approach has been referred to as semi-automated.

Cavezza and Alrajeh then proposes an automated refinement approach [8] based on Craig interpolation [35]. This approach does not require any human input, hence deemed fully-automated. In this approach, the unrealizable cores are first calculated, then refinements are found automatically via interpolation over unrealizable-cores form of assumptions and guarantees, translated into propositional logical formulae. Such calculations are performed symbolically in a proof-by-refutation fashion. The significance of computing the unrealizable cores is that it yields the cause of unrealizability for a certain counterstrategy, which is targeted by the following iterative refinement process directly. Therefore, it constitutes the automated fault localization step in the entire process, which was shown to steepen the rate of convergence within the iterative refinement step, both demonstrated in theory and in application [8]. Note that, subsequent to this work, more efficient ways to compute GR(1) unrealizable cores for has been found, such as in [36].

Alternatively, Maoz *et al.* [9] creates the design of a fully-automated assumptions repair procedure, `JVTS-Repair`, which uses the symbolic JVTS [37] representation of counterstrategies to find repairs for specifications, without the need to extract counterexamples. In their experimentation setup, `JVTS-Repair` is shown to be much more efficient that the template-based implementation in [7], given that the human input did not provide any insight to unrealizability. However, this approach is also demonstrated in their work to be sound but incomplete. Notably, Maoz *et al.* in the same literature gives an alternative assumptions repair algorithm, `GLASS`, by direct analysis on the unsatisfiable specification, without the using any counterstrategies.

All of the existing approaches introduced in this section, however, are arguably limited in their own fashions. Earlier template-based methods are, first and foremost, not fully automated; besides, these approaches are not designed to uncover or target the cause of unrealizability in a specification, which potentially contributed to lower operational efficiency. The more recent JVTS-based approach, though being fully automated, and more efficient thanks to their use of symbolic representation for counterstrategies, does not seem to be directed towards cause of unrealizability in a way akin to [8]. Finally, the interpolation-based approach, which computes unrealizable cores for fault localization, is able to achieve faster convergence; however, it generates weaker assumptions than those of template-based approaches, since in each iteration, the approach only requires the assumptions to be true in one extracted counterexample, rather than the entire counterstrategy computed [8].

5.2.2 Unrealizability in Adaptive Systems

There are more recent academic research on reactive systems that focuses on systems with adaptive capabilities, such as the work by Buckworth *et al.* [12], the intermediate results of which is used for the evaluation of this project in Chapter 4. Therefore, it seems essential to briefly discuss this work in this section.

A fixed reactive system, albeit one that is realizable at design time, may fail in unforeseen scenarios [12]. Therefore, the goal of Buckworth *et al.* in their work [12] is to enable adaptive capabilities on reactive systems based on the MORPH reference architecture in [10]. In particular, this is achieved by an automated procedure that weakens usually the assumptions, and sometimes the guarantees, in the original GR(1) specification for the reactive system.

However, reducing the strength of assumptions can lead to unrealizability in the new specification. Therefore, Buckworth *et al.* in [12] focuses on designing a procedure that preserving realizability of the specification during weakening, referred to as a graceful degradation, achieved by first (1) weakening the assumptions until they are no longer violated on the initial counterexample, then (2) weaken the guarantees until the new specification becomes realizable again, all in a loop-based manner with candidate weakening.

It is for this reason that the evaluation of this project in Chapter 4 chooses to use the specification after being weakened in step (1), along with the new counterexamples for unrealizability after the weakening. The causality computed by Algorithm 1 could potentially point to the core reasons of unrealizability, helping their procedure achieve faster convergence.

Chapter 6

Conclusions

Applying causality to temporal logics is an active area of research. Solutions to this problem could bring significant improvements to current artificial intelligence paradigms, such as state-of-the-art large language models, which inherently lack the capabilities of temporal reasoning [38].

In this report, we focus on applying actual causality, formalized by Halpern and Pearl [16], to linear temporal logic for explaining violations in counterexamples. By revisiting the definition by Beer *et al.* [1], Chapter 3 proposes the new Definition 17, which is updated to the new, modified HP definition of causality, and utilizes a causal representation which is more informative, and arguably more intuitive.

Consequently, this report also presents Algorithm 1, a new bounded algorithm for automated causal computation based on Definition 17 in Chapter 3, which is sound, complete under bounds, and runs in polynomial time complexity with respect to its inputs under the finite assumption of input LTL formula. Evaluation are performed in Chapter 4 on our implementation of this algorithm, `Caupybara`, against our own implementation of the approximation algorithm by Beer *et al.* [1] as baseline. The dataset of GR(1) specifications and finite counterexamples originates from the work of Buckworth *et al.* [12] focusing on the construction of adaptive reactive systems.

Out of the 78 comparisons conducted for our evaluation, causes found by Algorithm 1 in 61 of them completely cover those found by the baseline; in 10 of the comparisons are stronger than the baseline; in 2 of the comparisons has imperfect coverage due to more precise interpretations of fairness guarantees; and in 5 of the comparisons has imperfect coverage due to conflicting conjuncts.

Given the time constraints, there are potential extensions on this project which are left unexplored. For the enhancement of this project, the following directions are suggested below for possible future works.

Infinite LTL semantics. Recent academic literature have demonstrated various tools that performs LTL satisfiability checks on infinite traces, such as `Aalta` [27] and `BLACK` [28]. A possible direction would be to explore the integration of these tools into `Caupybara`, replacing its existing LTL checker, and examine on the causes computed by `Caupybara` on infinite counterexamples.

More extensive evaluations. Additional evaluations can be performed, such as producing quantitative evidence for the time complexity of Algorithm 1. For instance, it would be helpful if we can verify that increasing *bound* has an exponential impact its runtime, whereas increasing the length of σ or the size of ϕ inputs only increases runtime by some polynomial factor.

Additionally, the current evaluation dataset includes only counterexamples with a mean length of 2.0 states. Causal computation on larger counterexamples could yield additional and interesting insights.

Application on reactive systems. Our initial results suggest the potential for the application of this project on unrealizability repairs in reactive systems. Several questions remain open for exploration. These include the possibility of adapting Definition 17 to cover entire GR(1) counterstrategies, and whether integrating Algorithm 1 could expedite the rate of convergence for assumptions refinement and repair procedures, as well as adaptive system designs.

Bibliography

[1] Beer I, Ben-David S, Chockler H, Orni A, Trefler RJ. Explaining counterexamples using causality. Formal Methods Syst Des. 2012;40(1):20-40. Available from: https://doi.org/10.1007/s10703-011-0132-2.

[2] Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'ar Y. Synthesis of Reactive(1) designs. J Comput Syst Sci. 2012;78(3):911-38. Available from: https://doi.org/10.1016/j.jcss.2011.08.007.

[3] Pnueli A, Rosner R. On the synthesis of a reactive module. In: Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989. ACM Press; 1989. p. 179-90. Available from: https://doi.org/10.1145/75277.75293.

[4] Maoz S, Ringert JO. GR(1) synthesis for LTL specification patterns. In: Nitto ED, Harman M, Heymans P, editors. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015. ACM; 2015. p. 96-106. Available from: https://doi.org/10.1145/2786805.2786824.

[5] Cavezza DG, Alrajeh D, György A. Minimal assumptions refinement for realizable specifications. In: Bae K, Bianculli D, Gnesi S, Plat N, editors. FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering, Seoul, Republic of Korea, July 13, 2020. ACM; 2020. p. 66-76. Available from: https://doi.org/10.1145/3372020.3391557.

[6] Li W, Dworkin L, Seshia SA. Mining assumptions for synthesis. In: Singh S, Jobstmann B, Kishinevsky M, Brandt J, editors. 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011, Cambridge, UK, 11-13 July, 2011. IEEE; 2011. p. 43-50. Available from: https://doi.org/10.1109/MEMCOD.2011.5970509.

[7] Alur R, Moarref S, Topcu U. Counter-strategy guided refinement of GR(1) temporal logic specifications. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. IEEE; 2013. p. 26-33. Available from: https://ieeexplore.ieee.org/document/6679387/.

[8] Cavezza DG, Alrajeh D. Interpolation-based GR(1) assumptions refinement. In: Legay A, Margaria T, editors. Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. vol. 10205 of Lecture Notes in Computer Science; 2017. p. 281-97. Available from: https://doi.org/10.1007/978-3-662-54577-5_16.

[9] Maoz S, Ringert JO, Shalom R. Symbolic repairs for GR(1) specifications. In: Atlee JM, Bultan T, Whittle J, editors. Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. IEEE / ACM; 2019. p. 1016-26. Available from: https://doi.org/10.1109/ICSE.2019.00106.

[10] Braberman VA, D'Ippolito N, Kramer J, Sykes D, Uchitel S. MORPH: A reference architecture for configuration and behaviour self-adaptation. In: Filieri A, Maggio M, editors. Proceedings of the 1st International Workshop on Control Theory for Software Engineering, CTSE@SIGSOFT FSE 2015, Bergamo, Italy, August 31 - September 04, 2015. ACM; 2015. p. 9-16. Available from: https://doi.org/10.1145/2804337.2804339.

[11] Alrajeh D, Benjamin P, Uchitel S. Adaptation^2: adapting specification learners in assured adaptive systems. In: 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021. IEEE; 2021. p. 1347-52. Available from: https://doi.org/10.1109/ASE51524.2021.9678919.

[12] Buckworth T, Alrajeh D, Kramer J, Uchitel S. Adapting specifications for reactive controllers. In: 18th IEEE/ACM Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2023, Melbourne, Australia, May 15-16, 2023. IEEE; 2023. p. 1-12. Available from: https://doi.org/10.1109/SEAMS59076.2023.00012.

[13] Löding C, Madhusudan P, Neider D. Abstract learning frameworks for synthesis. In: Chechik M, Raskin J, editors. Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. vol. 9636 of Lecture Notes in Computer Science. Springer; 2016. p. 167-85. Available from: https://doi.org/10.1007/978-3-662-49674-9_10.

[14] Jha S, Seshia SA. A theory of formal synthesis via inductive learning. Acta Informatica. 2017;54(7):693-726. Available from: https://doi.org/10.1007/s00236-017-0294-5.

[15] Halpern JY, Pearl J. Causes and explanations: A structural-model approach, Part I: causes. CoRR. 2000;cs.AI/0011012. Available from: https://arxiv.org/abs/cs/0011012.

[16] Halpern JY. A modification of the Halpern-Pearl definition of causality. In: Yang Q, Wooldridge MJ, editors. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. AAAI Press; 2015. p. 3022-33. Available from: http://ijcai.org/Abstract/15/427.

[17] Huth M, Ryan MD. Logic in computer science - Modelling and reasoning about systems (2. ed.). Cambridge University Press; 2004.

[18] Giacomo GD, Vardi MY. Linear temporal logic and linear dynamic logic on finite traces. In: Rossi F, editor. IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. IJCAI/AAAI; 2013. p. 854-60. Available from: https://dl.acm.org/doi/10.5555/2540128.2540252.

[19] Gastin P, Oddoux D. Fast LTL to Büchi Automata Translation. In: Berry G, Comon H, Finkel A, editors. Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings. vol. 2102 of Lecture Notes in Computer Science. Springer; 2001. p. 53-65. Available from: https://doi.org/10.1007/3-540-44585-4_6.

[20] Könighofer R, Hofferek G, Bloem R. Debugging formal specifications using simple counterstrategies. In: Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA. IEEE; 2009. p. 152-9. Available from: https://doi.org/10.1109/FMCAD.2009.5351127.

[21] Lewis D. Causation. Journal of Philosophy. 1973;70(17):556-67. Available from: http://www.jstor.org/stable/2025310.

[22] Eisner C, Fisman D, Havlicek J, Lustig Y, McIsaac A, Campenhout DV. Reasoning with temporal logic on truncated paths. In: Jr WAH, Somenzi F, editors. Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. vol. 2725 of Lecture Notes in Computer Science. Springer; 2003. p. 27-39. Available from: https://doi.org/10.1007/978-3-540-45069-6_3.

[23] Chockler H, Halpern JY, Kupferman O. What causes a system to satisfy a specification? CoRR. 2003;cs.LO/0312036. Available from: http://arxiv.org/abs/cs/0312036.

[24] Stockmeyer LJ. The polynomial-time hierarchy. Theor Comput Sci. 1976;3(1):1-22. Available from: https://doi.org/10.1016/0304-3975(76)90061-X.

[25] Fionda V, Greco G. The complexity of LTL on finite traces: hard and easy fragments. In: Schuurmans D, Wellman MP, editors. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. AAAI Press; 2016. p. 971-7. Available from: https://doi.org/10.1609/aaai.v30i1.10104.

[26] Cormen TH, Leiserson CE, Rivest RL, Stein C. In: Introduction to algorithms. fourth edition. ed. Cambridge, Massachusett: The MIT Press; 2022. p. 275 296. ISBN: 978-02-623-6750-9.

[27] Li J, Yao Y, Pu G, Zhang L, He J. Aalta: an LTL satisfiability checker over infinite/finite traces. In: Cheung S, Orso A, Storey MD, editors. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014. ACM; 2014. p. 731-4. Available from: https://doi.org/10.1145/2635868.2661669.

[28] Geatti L, Gigante N, Montanari A, Venturato G. SAT meets tableaux for linear temporal logic satisfiability. J Autom Reason. 2024;68(2):6. Available from: https://doi.org/10.1007/s10817-023-09691-1.

[29] Performance characteristics — docs.scala-lang.org. Lausanne, Switzerland: EPFL;. [Accessed 11-06-2024]. https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html.

[30] Leitner-Fischer F, Leue S. Causality checking for complex system models. In: Giacobazzi R, Berdine J, Mastroeni I, editors. Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings. vol. 7737 of Lecture Notes in Computer Science. Springer; 2013. p. 248-67. Available from: https://doi.org/10.1007/978-3-642-35873-9_16.

[31] Beer A, Heidinger S, Kühne U, Leitner-Fischer F, Leue S. Symbolic Causality Checking Using Bounded Model Checking. In: Fischer B, Geldenhuys J, editors. Model Checking Software - 22nd International Symposium, SPIN 2015, Stellenbosch, South Africa, August 24-26, 2015, Proceedings. vol. 9232 of Lecture Notes in Computer Science. Springer; 2015. p. 203-21. Available from: https://doi.org/10.1007/978-3-319-23404-5_14.

[32] Caltais G, Guetlein SL, Leue S. Causality for general LTL-definable properties. In: Finkbeiner B, Kleinberg S, editors. Proceedings 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST@ETAPS 2018, Thessaloniki, Greece, 21st April 2018. vol. 286 of EPTCS; 2018. p. 1-15. Available from: https://doi.org/10.4204/EPTCS.286.1.

[33] Coenen N, Finkbeiner B, Frenkel H, Hahn C, Metzger N, Siber J. Temporal causality in reactive systems. In: Bouajjani A, Holík L, Wu Z, editors. Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022,

Virtual Event, October 25-28, 2022, Proceedings. vol. 13505 of Lecture Notes in Computer Science. Springer; 2022. p. 208-24. Available from: https://doi.org/10.1007/978-3-031-19992-9_13.

[34] Beutner R, Finkbeiner B, Frenkel H, Siber J. Checking and sketching causes on temporal sequences. In: André É, Sun J, editors. Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II. vol. 14216 of Lecture Notes in Computer Science. Springer; 2023. p. 314-27. Available from: https://doi.org/10.1007/978-3-031-45332-8_18.

[35] Craig W. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J Symb Log. 1957;22(3):269-85. Available from: https://doi.org/10.2307/2963594.

[36] Maoz S, Shalom R. Unrealizable cores for reactive systems specifications. CoRR. 2021;abs/2103.00297. Available from: https://arxiv.org/abs/2103.00297.

[37] Kuvent A, Maoz S, Ringert JO. A symbolic justice violations transition system for unrealizable GR(1) specifications. In: Bodden E, Schäfer W, van Deursen A, Zisman A, editors. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017. ACM; 2017. p. 362-72. Available from: https://doi.org/10.1145/3106237.3106240.

[38] Shah C, Bender EM. Situating search. In: Elsweiler D, editor. CHIIR '22: ACM SIGIR Conference on Human Information Interaction and Retrieval, Regensburg, Germany, March 14 - 18, 2022. ACM; 2022. p. 221-32. Available from: https://doi.org/10.1145/3498366.3505816.