Efficient Discovery of Actual Causality Using Abstraction Refinement

Arshia Rafieioskouei and Borzoo Bonakdarpour

***Abstract*—*Causality* is the relationship where one event contributes to the production of another, with the cause being partly responsible for the effect and the effect partly dependent on the cause. In this article, we propose a novel and effective method to formally reason about the causal effect of events in engineered systems, with application for finding the root-cause of safety violations in embedded and cyber–physical systems. We are motivated by the notion of *actual causality* by Halpern and Pearl, which focuses on the causal effect of particular events rather than type-level causality, which attempts to make general statements about scientific and natural phenomena. Our first contribution is formulating discovery of actual causality in computing systems modeled by transition systems as an satisfiability modulo theory solving problem. Since datasets for causality analysis tend to be large, in order to tackle the scalability problem of automated formal reasoning, our second contribution is a novel technique based on *abstraction refinement* that allows identifying for actual causes within smaller abstract causal models. We demonstrate the effectiveness of our approach (by several orders of magnitude) using three case studies to find the actual cause of violations of safety in 1) a neural network controller for a mountain car; 2) a controller for a Lunar Lander obtained by reinforcement learning; and 3) an MPC controller for an F-16 autopilot simulator.**

***Index Terms*—Causality, cyber–physical systems (CPSs), root-cause analysis, safety failures.**

I. Introduction

IN A CAUSAL system, the output of the system is influenced only by the present and past inputs. In other words, in a causal system, the present and future outputs depend solely on past and present inputs, not on future inputs. Causality addresses the logical dependencies between events and reflects the essence of event and action flows in systems. Engineers generally build causal systems, that is, structures, systems, and processes that seek to tie effects to their causes. This also includes approaches to explain the root-cause of failures that violate safety standards, especially in safety-critical systems.

In this context, embedded and cyber–physical systems (CPSs) are no exceptions. In fact, root-cause analysis has been of interest to both academic and industrial circles for decades, aiming not just to find safety violations but also to precisely *explain* why they happened. This means proving mathematically that safety would not have been violated in the absence of the identified cause. Formalizing and reasoning about causal explanations is much harder than just finding “bugs” and often aims to identify the earliest flawed decisions by controllers that lead to violations of safety requirements. Finding such causes provides engineers with tremendous insights to design more reliable systems, but it has been a long-standing and very challenging problem for various reasons, from defining a formal definition of causal effect of events to the high-computational complexity of counterfactual reasoning.

There is a wealth of research on causality analysis in the context of embedded and component-based systems from different perspectives [1], [2], [3], [4], [5], [6], [7], [8], [9]. Recently, there has been great interest in using temporal logics to reason about causality and explain bugs [10], [11], [12], [13]. In the CPS domain, using causality to repair AI-enabled controllers has recently gained interest [14], [15]. However, these lines of work either focus on only modeling aspects of causality or do not address the problem of scalability in automated reasoning about causality, which inherently involves a combinatorial blow up to enumerate counterfactuals.

Objectives

This article is concerned with the following problem. Given are 1) a formal operational description of a computing system $\mathcal{T}$ (e.g., a transition system of a CPS) and 2) a logical predicate φ_e that describes the effect (e.g., a safety failure) as input. Our goal is to identify a predicate φ_c that describes the cause of φ_e happening (e.g., the earliest bad decision made by a controller). We note that φ_e can be given by the user or can be found by using a verification or testing technique. Hence, the way the effect is identified is irrelevant to the problem studied in this article.

The first natural step is to formalize the definition of causality and in fact, there are several interpretations of the meaning of causality. In this article, we are motivated by the notion of *actual causality* by Halpern and Pearl (HP) [16], which focuses on the causal effect of particular events, rather than type causality, which attempts to make general statements about scientific and natural phenomena (e.g., smoking causes cancer). Actual causality is a formalism to deal with *token-level* causality, which aims to find the causal effect of individual events (in our context, in embedded and CPS), as

Manuscript received 7 August 2024; accepted 9 August 2024. This work was supported by the United States NSF Award CCF under Grant 2320050. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Borzoo Bonakdarpour.)*

The authors are with Michigan State University, East Lansing, MI 48824 USA (e-mail: rafieios@msu.edu; borzoo@msu.edu).

Digital Object Identifier 10.1109/TCAD.2024.3448299

opposed to *type-level* causality, which intends to generalize the causal effect of types of events.

As we aim at analyzing executions of computing systems (e.g., models or data logs of a CPS), we first formalize causal models in (possibly infinite-state) *transition systems*, rather than the classic set of structural equations [16]. We show that formalizing the three conditions of actual causality yields a second-order logic formula of the form $\varphi_{\mathsf{hp}} \triangleq \exists\tau.\exists\tau'.\forall\tau''.\psi$, where τ, τ', and τ'' range over the set of executions of a transition system and ψ stipulates the relation between actual and counterfactual worlds. More specifically.

1) The outermost existential quantifier in φ_{hp} intends to establishes a relationship between the cause and the effect in an execution τ (known as the *AC1* necessity condition in the HP framework [16]). That is, the cause and then the effect *actually* happen in τ.
2) The inner existential quantifier aims at refuting the causal effect relation in the counterfactual world (known as the *AC2(a)* condition, stipulating the "but-for" condition under contingencies). That is, when the cause does not happen, the effect will not happen.
3) The universal quantifier requires that if the cause happens in any execution τ'' that is to τ (as far as the variables contributing to the cause and effect are concerned), then the effect should also happen (known as the *AC2(b)* sufficiency condition).

This formula exhibits a quantifier alternation and indeed, the problem of deciding actual causality in a causal model is known to be DP-complete [17] in the size of the model, illustrating the computational complexity of the problem. To deal with this complexity, we propose an effective method to formally reason about actual causality using decision procedures to solve satisfiability modulo theory (SMT). Although there has been tremendous progress in developing efficient SMT solvers, they may not scale well when dealing with very large causal models or data logs. To tackle this problem, we introduce a novel technique based on *abstraction refinement* that allows identifying causes within smaller abstract causal models. This abstraction simplifies the model and attempts to view it from a higher level, while preserving the causal relations.

Although the idea of abstracting causal models in terms of structural equations has been studied in [18], [19], and [20], these works develop an exact simulation which may not exist or do not attempt to establish a relation between actual causes in the abstract and concrete causal models. Our technique incorporates two levels of abstraction to reason about actual causality (i.e., formula $\varphi_{\mathsf{hp}} \triangleq \exists\tau.\exists\tau'.\forall\tau''.\psi$). More specifically, our approach works as follows (see Figs. 1 and 2). Given a concrete causal model $\mathcal{T}$.

1) We first compute an under-approximate model $\check{\mathcal{T}}$ of $\mathcal{T}$. This model is used to find witnesses for conditions *AC1* and *AC2(a)* (i.e., the existential quantifiers). If not successful, we refine $\check{\mathcal{T}}$ (e.g., by including states that are in $\mathcal{T}$ and not in $\check{\mathcal{T}}$) and try again.
2) If the previous step succeeds, we compute an over-approximate model $\hat{\mathcal{T}}$ to verify condition *AC2(b)* for the universal quantifier. If successful, then an actual cause is identified and the algorithm terminates. Otherwise, we refine $\hat{\mathcal{T}}$ (e.g., by excluding states that are in $\hat{\mathcal{T}}$ and not in $\mathcal{T}$) and repeat the second step.[1]

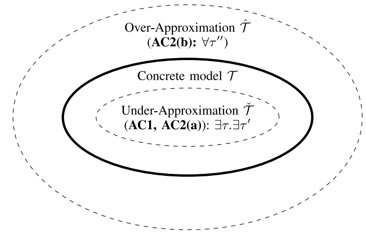

Fig. 1. Over/under-approximations of the concrete model and their relation to HP conditions of the form $\exists\exists\forall$.

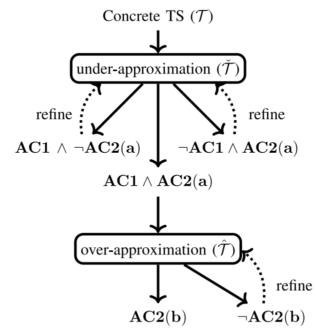

Fig. 2. Overall idea of our algorithm—steps of abstraction-refinement approach.

We prove the correctness of our approach by showing that our algorithm is sound (but not necessarily complete).

We have implemented[2] our approach using the Python programming language and utilized the SMT solver Z3 [21] and data analysis libraries [22], [23] to construct our solver and abstraction technique. We conduct experiments on three case studies to find the actual cause of violations of safety in 1) a neural network controller for a mountain car [24]; 2) a controller for a Lunar Lander obtained by reinforcement learning [24]; and 3) an MPC controller for an F-16 autopilot simulator [25]. Our experiments demonstrate the effectiveness of our abstraction-refinement technique by several orders of magnitude compared to the SMT-based approach for concrete causal models.

In summary, our contributions are the following. We: 1) formulate the classic HP framework by transition systems and introduce an SMT-based decision procedure to identify actual causes in a computing system; 2) introduce a technique based on abstraction refinement to deal with scalability of formal reasoning about actual causality; and 3) conduct three rigorous experimental evaluations on AI-enabled as well as non-AI controllers in CPS.

Our work is the first step in automating discovery of actual cause of failures, and our experiments show that we are able to identify the earliest bad decisions by controllers that lead to violations of safety requirements.

[1] Alternatively, one can return to the first step and start from scratch.

[2] Source code and all trace logs available at https://github.com/TART-MSU/Causality_abs_refinement.

Organization: The remainder of this article is organized as follows. Section II presents the classic HP framework for actual causality. In Section III, we introduce our formulation of HP for transition systems as well as a translation to an SMT-based decision procedure to identify actual causes. Our abstraction-refinement technique is introduced in Section IV. We present our experimental evaluation in Section V. Related work is discussed in Section VI. Finally, we make concluding remarks and discuss future work in Section VII. Proofs of correctness are available in [26].

II. Preliminaries—Actual Causality

In this section, we present the notion of *actual causality* by HP [16] as the baseline preliminary concept. Since, the definition in [16] is not a natural model of computation, in Section III, we will adapt the concepts in this section to transition systems and second-order logic formulas in order to reason about actual causality in computing systems. We will consistently use the *Mountain Car* running example to explain the definitions and concepts throughout this article.

A. Causal Models

Definition 1: A *signature* $\mathcal{S}$ is a tuple $(\mathcal{U}, \mathcal{V}, \mathcal{R})$, where $\mathcal{U}$ is set of *exogenous* variables (variables that represent factors outside the control of the model), $\mathcal{V}$ is a set of *endogenous* variables (variables whose values are ultimately determined by the values of the endogenous and exogenous variables). $\mathcal{R}$ is a function that associates with every variable $Y \in \mathcal{U} \cup \mathcal{V}$ a nonempty set $\mathcal{R}(Y)$ of possible values for Y.

Following Definition 1, a *state* is a valuation of a vector of variables $\vec{X} = (X_1, \ldots, X_n)$ in $\mathcal{U} \cup \mathcal{V}$, where each variable $X \in \vec{X}$ is assigned a value from $\mathcal{R}(X)$.

Definition 2: A *basic causal model* $\mathcal{M}$ is a pair $(\mathcal{S}, \mathcal{F})$, where $\mathcal{S}$ is a signature and $\mathcal{F}_X$ defines a function that associates with each endogenous variable X a *structural equation* $\mathcal{F}_X$ that maps $\mathcal{R}(\mathcal{U} \cup \mathcal{V} - \{X\})$ to $\mathcal{R}(X)$, so $\mathcal{F}_X$ determines the value of X, given the values of all the other variables in $\mathcal{U} \cup \mathcal{V}$.

It is important to highlight that exogenous variables cannot be linked to a function; thus, assigning values to exogenous variables, denoted as $\vec{u}$, is referred to as a *context*.

Definition 3: An *intervention* entails setting the values of endogenous variables, denoted as $\vec{X} \leftarrow \vec{x}$, and this notation signifies that the variables within set $\vec{X}$ are assigned values $\vec{x} = (x_1, \ldots, x_n)$.

The structural equations define what happens in the presence of interventions. Setting the value of some variables $\vec{X}$ to $\vec{x}$ in a causal model $\mathcal{M} = (\mathcal{S}, \mathcal{F})$ results in a new causal model, denoted $\mathcal{M}_{\vec{X} \leftarrow \vec{x}}$, which is identical to $\mathcal{M}$, except that $\mathcal{F}$ is replaced by $\mathcal{F}^{\vec{X} \leftarrow \vec{x}}$: for each variable $Y \notin \vec{X}$, $\mathcal{F}_Y^{\vec{X} \leftarrow \vec{x}} = \mathcal{F}_Y$ while for each X' in $\vec{X}$, the equation $\mathcal{F}_{X'}$ is replaced by $X' = x'$. Thus, we define a *causal model* $\mathcal{M}$ by a tuple $(\mathcal{S}, \mathcal{F}, \mathcal{I})$, where $(\mathcal{S}, \mathcal{F})$ is a basic causal model (see Definition 2) and $\mathcal{I}$ is a set of *allowed interventions*. Following [20], the sets of "allowed interventions" ensure that the interventions can be appropriately limited to include only those that can be abstracted.

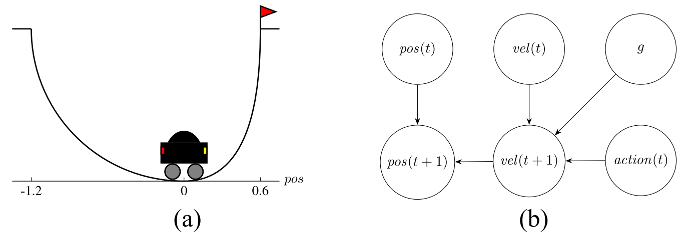

Fig. 3. (a) Schematic of the mountain car example. (b) Graph illustrating the causal model and relationships between the variables at a snapshot in time t.

Example 1: Consider a car located in a valley and aiming to reach the top of a mountain [see Fig. 3(a)]. At each time step, the controller of the car determines whether to apply positive or negative acceleration to guide the car toward the mountain top. We define signature $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ for this example as follows. Let

$$\mathcal{U} = \{\text{pos}(0), \text{vel}(0), g\}$$

be the set of exogenous variables, denoting the initial position, initial velocity, and the gravitational force on the car, respectively. Let

$$\mathcal{V} = \{\text{pos}(t), \text{vel}(t), \text{action}(t)\}$$

be the set of endogenous variables, denoting the position, velocity, and the controller action, respectively, at each time step t, where $t \neq 0$. We also set

$$\mathcal{R}(\text{pos}(t)) = [-1.2, 0.6]$$
$$\mathcal{R}(\text{vel}(t)) = [-0.07, 0.07]$$
$$\mathcal{R}(\text{action}(t)) = \{-1, 0, 1\}$$

where -1, 0, and 1 are assigned as accelerate to the left, do not accelerate, and accelerate to the right, respectively, for all $t \geq 0$.

Now, we define the causal model $(\mathcal{S}, \mathcal{F})$ based on the system dynamics for each $t > 0$ by structural equations

$$\mathcal{F}_{\text{pos}(t+1)} = \mathcal{F}_{\text{pos}(t)} + \mathcal{F}_{\text{vel}(t)} \quad (1)$$
$$\mathcal{F}_{\text{vel}(t+1)} = \mathcal{F}_{\text{vel}(t)} + 0.001\mathcal{F}_{\text{action}(t)} - g.\cos\big(3\mathcal{F}_{\text{pos}(t)}\big). \quad (2)$$

To illustrate the dependencies of the system, we can use a causal graph, as shown in Fig. 3(b). In this model, $\mathcal{M}_{\text{action}(t) \leftarrow 1}$ denotes the model obtained by an intervention, where the action(t) is set to 1 at time t (for some $t > 0$).

B. Causal Formulas

To precisely define actual causality, formal language is essential for articulating causal statements with clarity and rigor, in particular to formalize causes and effects. We use an extension of propositional logic, wherein primitive events take the form $\vec{X} = \vec{x}$, representing an endogenous variable $\vec{X}$ and a possible value $\vec{x}$ for $\vec{X}$. The combination of primitive events is achieved through standard propositional connectives, such as $\{\wedge, \vee, \neg\}$. Thus, in this article, we are only concerned with causal formulas that are state-based (and not temporal).

Given a signature $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$, a *primitive event* is a formula of the form $X = x$, for $X \in \mathcal{V}$ and $x \in \mathcal{R}(X)$. A *causal* *formula* (over $\mathcal{S}$) is one of the form $[Y_1 \leftarrow y_1, \ldots, Y_k \leftarrow y_k]\varphi$, where φ is Boolean combination of primitive events, $Y_1, \ldots, Y_k$ are distinct variables in $\mathcal{V}$, and $y_i \in \mathcal{R}(Y_i)$. Such a formula is abbreviated as $[\vec{Y} \leftarrow \vec{y}]\varphi$. The special case where $k = 0$ is abbreviated as $[]\varphi$ or, more often, just φ. Intuitively, $[\vec{Y} \leftarrow \vec{y}]\varphi$ says that φ would hold if Y_i were set to y_i, for $i = 1, \ldots, k$.

A causal formula ψ is true or false in a causal model, given a context. We use a pair $(\mathcal{M}, \vec{u})$ consisting of a causal model $\mathcal{M}$ and context $\vec{u}$ as a *causal setting*. Hence, we write $(\mathcal{M}, \vec{u}) \models \psi$ if the causal formula ψ is true in the causal setting $(\mathcal{M}, \vec{u})$. We are restricted to recursive models, where given a context, no cyclic dependencies exists. In a recursive model, $(\mathcal{M}, \vec{u}) \models X = x$ if the value of X is x once we set the exogenous variables to $\vec{u}$. Given a model $\mathcal{M}$, the model that describes the result of this intervention is $\mathcal{M}_{\vec{Y} \leftarrow \vec{y}}$. Thus, $(\mathcal{M}, \vec{u}) \models [\vec{Y} \leftarrow y]\psi$ iff $(\mathcal{M}_{\vec{Y} \leftarrow \vec{y}}, \vec{u}) \models \psi$. Mathematical formalism serves to express the intuition precisely encapsulated within the formula $[\vec{Y} \leftarrow \vec{y}]\psi$ is true in a causal setting $(\mathcal{M}, \vec{u})$ exactly if the formula ψ is true in the model that results from the intervention, in the same context $\vec{u}$.

Example 2: Context $\vec{u}$ in causal setting $(\mathcal{M}, \vec{u})$ in our example is determined by system inputs: initial velocity, initial position, and gravity

$$\vec{u} = \Big\{(\text{vel}(0) \leftarrow 0.01), (\text{pos}(0) \leftarrow 0), (g \leftarrow 0.0025)\Big\}$$

where we defined $\mathcal{M}$ in Example 1. To conduct causal analysis, the car at time $t = 0$ decides to set action(0) = 1, but it fails to reach the goal. We defined causal formula to express failure as follows:

$$\varphi_{\textsf{fail}} \triangleq \Big(\text{pos}(n) \neq 0.6\Big)$$

where 0.6 is the flag position and n is the last car state.

C. Actual Causality

Definition 4: $\vec{X} \leftarrow \vec{x}$ is an *actual cause* of φ in causal setting $(\mathcal{M}, \vec{u})$, if the following three conditions hold.

1) *AC1:* $(\mathcal{M}, \vec{u}) \models [\vec{X} \leftarrow \vec{x}]\varphi$.
2) *AC2(a):* There is a partition of $\mathcal{V}$ (set of endogenous variables) into two disjoint subsets $\vec{Z}$ and $\vec{W}$ (i.e, $\vec{Z} \cap \vec{W} = \emptyset$) with $\vec{X} \subseteq \vec{Z}$ and a setting $\vec{x}'$ and $\vec{w}$ of the variables in $\vec{X}$ and $\vec{W}$, respectively, such that
$$(\mathcal{M}, \vec{u}) \models \left[\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}\right] \neg\varphi.$$
3) *AC2(b):* For all subsets $\vec{Z}'$ of $\vec{Z} - \vec{X}$, we have
$$(\mathcal{M}, \vec{u}) \models \left[\vec{X} \leftarrow \vec{x}, \vec{W} \leftarrow \vec{w}, \vec{Z}' \leftarrow \vec{z}^*\right]\varphi$$
where $\vec{z}^*$ denotes that variables in $\vec{Z}'$ are fixed at their values in the actual context.
4) *AC3:* $\vec{X}$ is minimal; no subset of $\vec{X}$ satisfies AC1 and AC2.

Roughly speaking Definition 4 expresses the following. AC1 says that $\vec{X} = \vec{x}$ cannot be considered a cause of φ unless both $\vec{X} = \vec{x}$ and φ actually happen. AC2(a) says that the but-for condition holds under the contingency $\vec{W} = \vec{w}$. Also, changing the value of some variable in $\vec{X}$ results in changing the value(s) of some variable(s) in $\vec{Z}$ (perhaps recursively), which finally results in the truth value of φ changing. Finally, AC2(b) provides a sufficiency condition: if the variables in $\vec{X}$ and an arbitrary subset $\vec{Z} - \vec{X}$ of other variables on the causal path are held at their values in the actual context, then φ holds even if $\vec{W}$ is set to $\vec{w}$ (the setting for $\vec{W}$ used in AC2(a)). The types of events that the HP definition allows as actual causes are ones of the form $X_1 = x_1 \wedge \cdots \wedge X_k = x_k$, that is, conjunctions of primitive events; this is often abbreviated as $\vec{X}$. In Section III, Example 3, we will provide an example on how actual cause of formula $\varphi_{\textsf{fail}}$ can be identified using our proposed technique.

III. SMT-Based Discovery of Actual Causality

In this section, we transform the components of the HP framework presented in Section II into transition systems and a second-order formula to express actual causality. Such a transition system can model the operational behavior of a system (e.g., a controller). Our technique can be agnostic to the details of the system and only take a set of execution traces.

Recall that a causal model $\mathcal{M}$ is of the form $(\mathcal{S}, \mathcal{F}, \mathcal{I})$, where $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$. Also, recall that a state is a mapping from the variables in $\mathcal{U} \cup \mathcal{V}$ to their respective domain of values. We start with representing $\mathcal{M}$ with a set of traces obtained from a transition system that essentially describes how the state of all variables in $\mathcal{U} \cup \mathcal{V}$ evolve over time by structural equations $\mathcal{F}_X$, for every $X \in \mathcal{V}$.

A. Transition Systems

Definition 5: A *transition system* $\mathcal{T}$ corresponding to a causal model $\mathcal{M}$ is a tuple $\mathcal{T} = (\Sigma, \Delta, \sigma^0, \Lambda)$, where: 1) Σ is the set of all possible states obtained from all possible valuations of variables in $\mathcal{U} \cup \mathcal{V}$; 2) Δ is a function mapping states in 2^Σ to a state in Σ (recall that $\mathcal{F}_X$ is a function); 3) $\sigma^0 \in \Sigma$ is the initial state, and 4) Λ is a function mapping states in 2^Σ to an atomic proposition from a set AP (e.g., given by causal formulas).

Following Definition 5, given a causal setting $(\mathcal{M}, \vec{u})$, the corresponding *causal transition system* is one that is acyclic and fixes σ_0 according to $\vec{u}$. An intervention $\vec{X} \leftarrow \vec{x}$ is simply a set of transitions in Δ where in the target state $\vec{X} = \vec{x}$ holds, denoted by $\Delta_{\vec{X} \leftarrow \vec{x}}$. We denote the set of all possible interventions in $\mathcal{T}$ by $\mathcal{I}_{\mathcal{T}}$.

Definition 6: A *path* of a transition systems $\mathcal{T} = (\Sigma, \Delta, \sigma^0, \Lambda)$ is a sequence of states of form $\sigma_0\sigma_1 \ldots$, where for all $i \geq 0$ 1) $\sigma_0 = \sigma^0$ and 2) $(\sigma_i, \sigma_{i+1}) \in \Delta$. The *trace* corresponding to a path $\sigma_0\sigma_1 \ldots$ is the sequence $\tau = \Lambda(\sigma_0)\Lambda(\sigma_1) \ldots$.

Let Tr denote the set of all traces of a transition system.

Example 3: Fig. 4 shows three traces τ_0, τ_1, and τ_2 for our mountain car example for context $\vec{u} = (\text{pos}(0) = 0.0, \text{vel}(0) = 0.02)$. In each step, the controller makes acceleration decisions. Dotted transitions means the next state is not the immediate next time step. The nth state is the last state of the trace. As can be seen, traces τ_0 and τ_2 never reach position 0.6 (i.e., satisfying causal formula $\varphi_{\textsf{fail}}$,

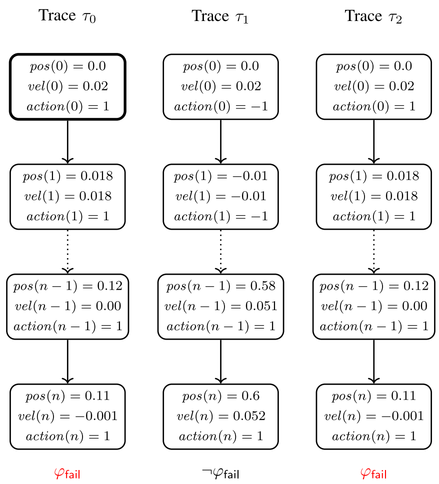

Fig. 4. Three traces for the mountain car example.

meaning failing to reach the flag), while trace τ_1 does (i.e., violating causal formula φ_{fail}, meaning successfully reaching the flag).

We introduce three temporal operators to express the occurrence of causes and effects in traces: 1) for a state σ and a proposition $p \in \mathsf{AP}$ iff $\sigma \models p$ iff $p \in \Lambda(\sigma)$; 2) a trace $\tau = \tau_0\tau_1\ldots$ satisfies formula $\Box p$ (read as "always p" and denoted $\tau \models \Box p$) iff $\forall i \geq 0. \tau_i \models p$; 3) a trace $\tau_0\tau_1\ldots$ satisfies formula $\Diamond p$ (read as 'eventually p' and denoted $\tau \models \Diamond p$) iff $\exists i \geq 0. \tau_i \models p$; and 4) a trace $\tau_0\tau_1\ldots$ satisfies formula $p\,\mathcal{U}\,q$ (read as "p until q" denoted $\tau \models p\,\mathcal{U}\,q$) iff $\exists i \geq 0. (\tau_i \models q \wedge (\forall j < i. \tau_j \models p))$.

B. SMT-Based Formulation of Actual Causality

An SMT decision problem generally consists of two components: 1) the SMT instance (i.e., data elements, such as variables, domains, functions, sets, etc.) and 2) SMT constraints (i.e., first-order modulo theory involving quantified Boolean predicates with arithmetic). In the context of our problem, the SMT instance consists of two parts:

1) A set of elements for expressing a transition system $\mathcal{T}$ or 2) a set of traces Tr (e.g., from a data log). While the latter is simply a set of sequences of states (defined as a function from natural numbers to the full set of states), the former is specified by Boolean formulas from the unrolled transition system, similar to standard bounded model checking [27] without loops.
2) Our SMT model formalize conditions *AC1*, *AC2(a)*, and *AC2(b)* of Definition 4 for transition systems (see Fig. 5). For simplification, we omit AC3 (minimality of cause), as it is not the most important constraint to reason about causal effect of events in a system. Condition *AC1* (in Fig. 5) means in the set Tr, there exists at least one trace τ, where effect φ_e appears after cause φ_c holds. Condition *AC2(a)* requires the existence of one trace τ', where neither cause φ_c nor effect φ_e hold.

Additionally, trace τ' is not equivalent to trace τ (identified in *AC1*) as far as variables in W or Z are concerned (i.e., the counterfactual worlds). The remaining endogenous variables, the ones in $\vec{W}$, are off to the side, so to speak, but may still have an indirect effect on what happens. Condition *AC2(b)* requires that for all traces τ'' that are similar to τ as far as causal variables in Z are concerned, if cause φ_c holds, then effect φ_e hold some time in the future.

We clarify that while SMT solvers cannot directly encode temporal operators, one can easily encode them using the above expanded definitions by quantifiers over traces.

SMT Decision Problem

Given are 1) a causal transition system $(\mathcal{T}, \vec{u})$ (or a set of traces Tr expressed as a mapping from natural numbers to states); 2) a causal formula φ_e; 3) an uninterpreted function representing φ_c; and 4) constraints *AC1*, *AC2(a)*, and *AC2(b)*. The corresponding SMT instance is satisfiable iff the interpreted φ_c is an actual cause of φ_e in $\mathcal{T}$.

Example 4: We aim to identify the cause of the failure, denoted as φ_{fail}, explained in our running example. For the sake of argument, let $X = \{\text{action}(0) = 1\}$. Since both pos and vel are dependent on the value of action, they are part of $\vec{Z}$ or the causal path. That is

$$\vec{Z} = \big\{\text{pos}(t), \text{action}(t), \text{vel}(t) \mid t > 1\big\}$$

and, hence, $W = \{\}$ (since $\vec{W} \cap \vec{Z} = \emptyset$). We now analyze the conditions of HP.

1) Starting with *AC1*, one can instantiate τ (in Fig. 5) with concrete trace τ_0 in Fig. 4, indicating the satisfaction of the first condition.
2) Moving to *AC2(a)*, which involves counterfactual reasoning, when we change the actual setting in *AC1* to a counterfactual value $\text{action}(0) = -1$, the car eventually reaches the goal (i.e., $\text{pos} = 0.6$). This change allows the car to initiate a leftward movement, acquiring the necessary momentum to reach the flag, so flipping the failure φ_{fail} to success (i.e., $\neg\varphi_{\mathsf{fail}}$). Consequently, *AC2(a)* is satisfied by instantiating τ' (in Fig. 5) with concrete trace τ_1 (in Fig. 4). Also, notice that condition $\tau_1 \not\equiv_Z \tau_0$ is satisfied.
3) Considering *AC2(b)*, notice that trace τ_2 is identical to τ_0 as far as the variables in $\vec{Z}$ are concerned (i.e., $\tau_0 \equiv_Z \tau_2$). Also, since $\vec{W} = \{\}$, changing variables in $\vec{W}$ while preserving the actual context results in an equivalent scenario to *AC1*, which is already satisfied. Thus, the only trace that can instantiate τ'' (in Fig. 5) is τ_2, in which φ_{fail} becomes true. Note that the reason τ_0 and τ_2 are trace-equivalent is indeed due to the fact that $\vec{W} = \{\}$. Hence, *AC2(b)* hold.

This means in this set of traces, $\text{action}(0) = 1$ is the actual cause of failure for the car to reach the flag.

In the ideal world, one has to have all possible traces for combinatorial enumeration to evaluate *AC2(b)*. However, this is far from reality and most trace data logs (e.g., by some testing mechanisms, fuzzing, mutation testing, some automaton, etc.) include only a subset of possibilities. Our goal in this article is to identify causal effects *within* a given set of traces.

$$\textbf{AC1} \triangleq \exists \tau \in \mathsf{Tr}.\ \big(\tau \models \neg\varphi_e\, \mathcal{U}\, (\varphi_c \wedge \Diamond \varphi_e)\big) \qquad (\varphi_c \text{ causes } \varphi_e \text{ in } \tau)$$

$$\textbf{AC2(a)} \triangleq \exists \tau' \in \mathsf{Tr}.\big(\tau' \models \Box(\neg\varphi_c \ \wedge\ \neg\varphi_e)\big) \wedge \big(\tau \not\equiv_Z \tau' \vee \tau \not\equiv_W \tau'\big) \qquad (\text{changes in the causal inhibits } \varphi_e)$$

$$\textbf{AC2(b)} \triangleq \forall \tau'' \in \mathsf{Tr}.\big((\tau'' \models (\neg\varphi_e\, \mathcal{U}\, \varphi_c) \wedge (\tau \equiv_Z \tau'' \wedge \tau \not\equiv_W \tau'')\big) \rightarrow \big(\tau'' \models \Diamond \varphi_e\big) \ (\text{in traces similar to } \tau,\ \varphi_c \text{ causes } \varphi_e)$$

Fig. 5. HP conditions adapted for causal transition systems.

Finally, as mentioned in the introduction, decision procedure for verification of actual causality is DP-complete [17], signifying the computation difficulty of automated reasoning about causality. This means our SMT-based problem is indeed dealing with a decision problem that is DP-complete, setting the complexity of our SMT-based solution.

IV. Abstraction Refinement for Causal Models

In this section, we propose our abstraction-refinement technique and its application in reasoning about actual causality, as presented in Section III.

A. Overall Idea

Generally speaking, the traditional abstraction approach to handle an existential quantifier is *under-approximation*, where we start from a subset of behaviors and attempt to instantiate the quantifier. If successful, then the problem is solved. Otherwise, we refine the abstraction, by including addition behaviors and try again. On the contrary, to handle universal quantifiers, the traditional abstraction approach is *over-approximation*, where we start from a subset of behaviors and attempt to verify universality. If successful, then the problem is solved. Otherwise, we need to ensure that the counterexample is not *spurious* (due to over-approximation). If it is, we refine the abstraction by excluding the counterexample and try again.

The overall idea of our technique is as follows (see Fig. 2). Observe that the logic formula for actual causality is of the form $\exists\exists\forall$ (see Fig. 5). Given a transition system $\mathcal{T}$ and causal formula φ_e as the effect, we proceed as follows.

1) *Step 1:* Compute an under-approximation $\check{\mathcal{T}}$ and an over-approximation $\hat{\mathcal{T}}$. We first attempt to instantiate the existential quantifiers in *AC1* and *AC2(a)* in $\check{\mathcal{T}}$. If instantiating one of the quantifiers does not succeed, we refine $\check{\mathcal{T}}$ and repeat step 1.
2) *Step 2:* When step 1 succeeds, we compute $\hat{\mathcal{T}}$ and verify the universal quantifier in *AC2(b)* for $\hat{\mathcal{T}}$. If successful, the witness to τ is a trace where the actual cause happens and we also obtain a witness to φ_c by the SMT solver. Otherwise, we can either refine $\hat{\mathcal{T}}$ and repeat step 2 or refine $\check{\mathcal{T}}$ and return to step 1.

We show that termination of these steps results in identifying an actual cause φ_c in $\mathcal{T}$ for φ_e. This algorithm, however, may never terminate and, thus, our approach is sound but not complete. We also remark that our heuristic based on abstraction refinement is sound but not complete (e.g., similar to the CEGAR [28] technique in model checking) to solve the general DP-complete problem. The computation complexity of our solution, therefore, does not change.

B. Approximating Causal Transition Systems

We first fix some notation. For a *concrete* causal transition system $\mathcal{T} = (\Sigma, \Delta, \sigma^0, \Lambda)$ (the one given as input for causal reasoning), let us denote an over-approximate causal transition system by $\hat{\mathcal{T}} = (\hat{\Sigma}, \hat{\Delta}, \hat{\sigma}^0, \hat{\Lambda})$ and an under-approximate causal transition system by $\check{\mathcal{T}} = (\check{\Sigma}, \check{\Delta}, \check{\sigma}^0, \check{\Lambda})$. We denote the domain of endogenous (respectively, exogenous) variables of $\mathcal{T}$ by $\mathcal{R}(\mathcal{V}_\mathcal{T})$ (respectively, $\mathcal{R}(\mathcal{U}_\mathcal{T})$.

Given an over-approximate causal transition system $\hat{\mathcal{T}}$, we construct a sequence $\hat{\mathcal{T}}_0 \geq \hat{\mathcal{T}}_1 \geq \cdots \hat{\mathcal{T}}_k$ of over-approximations, where (1) $\hat{\mathcal{T}}_k = \hat{\mathcal{T}}$, and $\hat{\mathcal{T}}_{i+1}$ is a refinement of $\hat{\mathcal{T}}_i$, for $0 \leq i < k$, which we compute using *counterexamples*. A counterexample is a state of $\hat{\Sigma}_i$ that is not in Σ. Over-approximation state mapping is a function which map states from $\mathcal{T}$ to $\hat{\mathcal{T}}$, i.e., $\hat{h}{:}2^\Sigma \mapsto \hat{\Sigma}$.

Assumption 1: In this article, we only allow over-approximation state mappings $\hat{h}$ that preserve the equality of traces as far as variables in Z are concerned. That is, for two concrete transitions (σ_0, σ_1) and (σ_0', σ_1'), if 1) $\sigma_0 \equiv_Z \sigma_0'$ and 2) $\sigma_1 \not\equiv_Z \sigma_1'$, then we have 1) $\sigma_0 \equiv_Z \hat{h}(\sigma_0')$ and 2) $\sigma_1 \not\equiv_Z \hat{h}(\sigma_1')$. Otherwise, we will not be able to prove the soundness of Algorithm 1 with regard to causal paths. We will elaborate more in the requirement in proof of Theorem 2. We will also explain in Section V, how this assumption is ensured in our implementation

We need an additional function: $\hat{w} : \mathcal{I}_\mathcal{T} \mapsto \mathcal{I}_{\hat{\mathcal{T}}}$ which maps concrete interventions to over-approximation interventions.

Definition 7: Given a subset of endogenous variables in Σ, called $\vec{X}$, and $\vec{x} \in 2^\Sigma$, let

$$\mathsf{Rst}(\Sigma, \vec{x}) = \{\vec{v} \in 2^\Sigma{:}\vec{x} \text{ is the restriction of } \vec{v} \text{ to } \vec{X}\}.$$

This definition carries to a transition system $\mathcal{T} = (\Sigma, \sigma^0, \Delta, \Lambda)$ in a straightforward fashion as follows. The restriction of a set of values $\vec{x}$ on Σ is a subset $\Sigma|_{\vec{x}} \subseteq \Sigma$ restricted to those states, where $\vec{X} = \vec{x}$. The set of restricted transitions is obviously those start and end in states in $\Sigma|_{\vec{x}}$.

We now explain how we compute the above functions. Given $\hat{h}$, we define: $\hat{w}(\Delta_{\vec{X}\leftarrow\vec{x}}) = \hat{\Delta}_{\vec{Y}\leftarrow\vec{y}}$ if 1) $\vec{y} \in 2^{\hat{\Sigma}}$ and 2) $\hat{h}(\mathsf{Rst}(\Sigma|_{\vec{x}})) = \mathsf{Rst}(\hat{\Sigma}|_{\vec{y}})$. Hence, for every intervention in $\Delta_{\vec{X}\leftarrow\vec{x}}$, there is only one intervention in $\hat{\Delta}_{\vec{Y}\leftarrow\vec{y}}$. If such a $\vec{Y}$ and $\vec{y}$ do not exist, we take $\hat{w}(\Delta_{\vec{X}\leftarrow\vec{x}})$ to be *undefined*. Let

Algorithm 1: Finding Actual Cause of φ_e in $\mathcal{T}$

Input: $\mathcal{T} = (\Sigma, \Delta, \sigma^0, \Lambda)$, causal formula φ_e, allowed interventions $\mathcal{I}^{\check{h}}_{\mathcal{T}}$, $\alpha = [0, 1]$, $\beta \geq 0$
Output: Causal formula φ_c

1 $\check{\mathsf{Tr}} \leftarrow \check{h}(\mathsf{Tr})$ using α;
2 **while** *true* **do**
3 $\quad \{\varphi_c, \check{\tau}, \check{\tau}'\} \leftarrow \mathrm{SMT}(\check{\mathsf{Tr}}, \mathbf{AC1} \wedge \mathbf{AC2(a)})$;
4 $\quad$ **if** $\neg\varphi_c$ **then**
5 $\quad\quad \hat{\mathcal{T}} \leftarrow \hat{h}(\mathcal{T})$ using β and φ_c;
6 $\quad\quad$ **while** *true* **do**
7 $\quad\quad\quad$ result $\leftarrow \mathrm{SMT}(\hat{\mathcal{T}}, \varphi_c, \mathbf{AC2(b)})$;
8 $\quad\quad\quad$ **if** *result* **then**
9 $\quad\quad\quad\quad$ **return** φ_c ;
10 $\quad\quad\quad$ **else**
11 $\quad\quad\quad\quad \hat{\mathcal{T}} \leftarrow \widehat{\mathsf{Refine}}(\hat{\mathcal{T}}, \Sigma - \hat{\Sigma}, \mathcal{I}^{\hat{h}}_{\mathcal{T}})$;
12 $\quad\quad\quad$ **end**
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 $\quad$ Increase α;
16 $\quad \check{\mathsf{Tr}} \leftarrow \check{h}(\mathsf{Tr})$ using α;
17 **end**

$\mathcal{I}^{\hat{h}}_{\mathcal{T}}$ be the set of interventions for which $\hat{w}$ is defined, and let $\mathcal{I}_{\hat{\mathcal{T}}} = \hat{w}(\mathcal{I}^{\hat{h}}_{\mathcal{T}})$.

Based on this definition, it becomes evident that not all interventions in $\mathcal{I}_{\mathcal{T}}$ will have corresponding mappings in $\mathcal{I}_{\hat{\mathcal{T}}}$ or $\mathcal{I}_{\check{\mathcal{T}}}$. This is due to the fact that $\check{h}$ and $\hat{h}$ may aggregate states, resulting in some $\mathcal{I}_{\mathcal{T}}$ representing only partial interventions on $\mathcal{I}_{\hat{\mathcal{T}}}$ or $\mathcal{I}_{\check{\mathcal{T}}}$. In this context, the introduction of a notion termed allowed intervention becomes crucial. This notion is essential as certain interventions in the abstract model may lack definition or relevance in a well-defined concrete model. Consequently, within this framework of definitions, the translation of interventions is not universal; rather, only essential interventions that can be meaningfully mapped are considered.

We follow a similar but simpler procedure for under-approximations. Given an under-approximate causal transition system $\check{\mathcal{T}}$, we construct a sequence $\check{\mathcal{T}}_0 \leq \check{\mathcal{T}}_1 \leq \cdots \check{\mathcal{T}}_k$ of under-approximations, where (1) $\check{\mathcal{T}}_k = \check{\mathcal{T}}$, and $\check{\mathcal{T}}_{i+1}$ is a refinement of $\check{\mathcal{T}}_i$, for $0 \leq i < k$, which we compute using *counterexamples*. A counterexample is a state of Σ that is not in $\check{\Sigma}_i$. In this article, since we begin causal analysis from a trace log Δ, we compute an under-approximation by a subset of the input set of traces. That is, $\check{h}(\mathsf{Tr}) \subseteq \mathsf{Tr}$.

C. Detailed Description of Algorithm

The input to Algorithm 1 is a concrete transition systems $\mathcal{T}$ (more specifically, its trace set) and a causal formula φ_e. Also, α and β and are parameters used in computing $\hat{h}$ and $\check{h}$, respectively. α indicates the subset size of $\check{h}$ and β is a threshold to compute Euclidean distance of states for over-approximation. We are restricted to a set of allowed interventions $\mathcal{I}^{\hat{h}}_{\mathcal{T}}$. Our objective is to identify states of $\mathcal{T}$, where causal formula φ_c holds as an actual cause in the trace $\tau = \check{\Lambda}(\sigma_0)\check{\Lambda}(\sigma_1)\ldots$

Line 1 initializes the under-approximation $\check{\mathcal{T}}$, with parameter α indicating the number of traces to use and map in $\check{h}$ function. In lines 3–16, the algorithm computes whether the SMT query returns φ_c as the cause for effect φ_e in the current under-approximation and over-approximation. Specifically, in line 3, the SMT function receives $\check{\mathcal{T}}$ as the under-approximation and constraints of *AC1* and *AC2(a)* specified in Fig. 5, and it returns the result φ_c as the cause. The SMT solver also returns a witness trace $\tau \in \check{\mathsf{Tr}}$. In line 16, if the result of SMT query in line 3 is unsatisfiability, then the algorithm chooses more traces Tr by increasing α. Indeed, lines 15 and 16 establish the refinement for the under-approximate model.

If a cause φ_c is identified by satisfying *AC1* and *AC2(a)*, we use this cause to initialize over-approximation in line 5 (to ensure Assumption 1), where we include all original states as well as potentially unreachable states by creating an abstract representation by function $\hat{h}$, such that all states in $\mathcal{T}$ map to $\hat{\mathcal{T}}$, and also similar states are merged into a single abstracted state in $\hat{\mathcal{T}}$. The distance threshold for merging states is controlled by the parameter β. If the distance between any pair of states is less than β, those states will be merged. Consequently, a smaller β results in a larger number of abstract states, while a larger β leads to a smaller number of abstract states.

In lines 7–11, we focus on verifying *AC2(b)* using the over-approximation. In line 7, the SMT query takes $\hat{\mathcal{T}}$ as the current over-approximate model and φ_c as output from line 3. It then examines whether all traces for which φ_c and φ_e hold can be modified by changing states such that φ_e still holds. If the SMT solver returns SAT, then φ_c is returned as the actual cause, where φ_c is a Boolean expression on the atomic propositions related to states in a specific trace. If the result is not SAT, in line 11, we use counterexample(s) in $\Sigma - \hat{\Sigma}$, allowed interventions identified by $\hat{w}(\mathcal{I}_{\mathcal{T}})$. These counterexamples are then eliminated by Refine, and the resulting model is assigned to the new $\hat{\mathcal{T}}$. We emphasize that in the refinement step for over-approximation (line 11), it is crucial to consider restricted interventions, denoted as $\mathcal{I}^{\hat{h}}_{\mathcal{T}}$. This consideration is necessary because, in a concrete model, certain interventions may not be directly mapped to their counterparts in the over-approximated model. Consequently, the refinement process must incorporate $\mathcal{I}^{\hat{h}}_{\mathcal{T}}$ as an essential input, utilizing it effectively during the mapping process to ensure consistency of model translation.

Theorem 1: Let $\mathcal{T}$ be a concrete causal transition system and φ_c and φ_e be two causal formulas. If φ_c is an actual cause of φ_e identified by Algorithm 1 (for $\hat{\mathcal{T}}$ and $\check{\mathsf{Tr}}$), then φ_c is an actual cause of φ_e in $\mathcal{T}$.

D. Correctness

In this section, we formally prove the soundness of Algorithm 1.

Theorem 2: Let $\mathcal{T}$ be a concrete causal transition system and φ_c and φ_e be two causal formulas. If φ_c is an actual cause of φ_e identified by Algorithm 1 (for $\hat{\mathcal{T}}$ and $\check{\mathsf{Tr}}$), then φ_c is an actual cause of φ_e in $\mathcal{T}$.

V. Experimental Evaluation

This section first provides an overview of the implementation details of the algorithm proposed in Section IV-C. We also evaluate our technique on three case studies: 1) mountain car; 2) Lunar Lander environments from OpenAI Gym [24]—commonly used evaluation benchmarks for learning-enabled CPS; and 3) an F-16 autopilot simulator [25] that uses an MPC controller.

A. Implementation

To identify the actual cause of failures in our studies, we need to generate traces consisting of those that do not violate safety and those that do violate safety. We need successful traces to find conterfactuals for failure scenarios, where the same conditions lead to success through different decisions.

In our experiments, we use 47 networks for the mountain car experiment [15], [29] and generate over 570 neural networks, trained with Deep Reinforcement Learning [30], for the Lunar Lander case study. Consequently, the success rates of traces in satisfying φ_e used in our experiments were 17% and 11% for the case studies in Sections V-C and V-D, respectively. In the case study in Section V-E, the success rates were 21% and 33% for the first and second scenarios, respectively. This is because the non-AI MPC controller typically makes better decisions than the AI controller.

We have implemented Algorithm 1 using the Python programming language. Algorithm 1 is implemented through two approaches. First, the Z3 SMT solver [21], and second (for nonsymbolic cases), by employing a search method to find traces in datasets that meet the HP conditions. For instance, if we find a trace that leads to failure, we take this sample and search for other traces with the same features, except for the decision that caused the failure in the original trace. To accomplish this, we utilize built-in data science search algorithms in [22] and [23]. In fact, in our case studies, we are dealing with large-sized data rather than symbolic properties. Therefore, in Section V-F, we will demonstrate that searching through the dataset is more efficient compared to Z3. While Z3 is primarily employed for its robust capabilities in theorem proving and constraint solving, it is not as effective for finding traces in a large set of already generated traces that meet certain conditions.

For execution of Algorithm 1, specific strategies are adopted in refinement of the under- and especially over-approximate (function Refine in line 11) models in cases of unsatisfiability. In the under-approximation model, a parameter α is utilized to incorporate additional traces. This parameter can be progressively increased to obtain more traces, thereby refining the under-approximation. In the over-approximation model, a parameter β is used within the mapping function to dictate the threshold for the distance between states. When the distance between a group of states is less than this threshold, they are merged into a single state to simplify the model. Moreover, in refining the over-approximate model, the algorithm checks for the existence of counterexample states that violate the over-approximation. If such states are identified, they are removed from the model to ensure its accuracy [28].

Assumption 1 for both our case studies is implemented as follows that in the over-approximation function, it is crucial not to merge states that transition to different outcomes. For instance, in the mountain car example, if there are two traces that differ only in their actions but have the same position and velocity, and the under-approximation model identifies that action might be a possible cause of failure, these states cannot be merged in the over-approximation model. This is because merging them would obscure the distinction between a trace leading to failure and another leading to success.

B. Experimental Settings

All of our experiments were conducted on a single core of the Apple M2 Pro CPU, which features a 10-core architecture and operates @3.7 GHz. Given a set of collected traces, we applied our techniques in four different modes to identify the cause of failure (safety violations): 1) *Only_Z3* is the implementation, where we only use the SMT solver Z3 to discover actual causality (the technique proposed in Section III); 2) *Abs_Z3* is the implementation, where Algorithm 1 uses Z3; 3) *Only_DA* is the implementation, where we only use the search algorithms in [22] and [23] in lieu of an SMT solver; and 4) *Abs_DA* is the implementation, where Algorithm 1 uses the search algorithms in [22] and [23] in lieu of an SMT solver.

C. Case Study 1: Mountain Car

Out first case study is the continuation of our running example. In Fig. 3(a), the car is initially positioned in the valley between two mountains with the objective being to navigate it to the peak of the right mountain before a set deadline. The system incorporates three variables in accordance with (1) and (2), specifying the domain for each variable as $\mathrm{pos}(t) \in [-1.2, 0.6]$, $\mathrm{vel}(t) \in [-0.07, 0.07]$, and $\mathrm{action}(t) \in [-1, 1]$. Here, action represents a learning-based function f, implemented using various pretrained neural networks of different dimensions

$$\mathrm{action}(t) = f(\mathrm{pos}(t), \mathrm{vel}(t)).$$

The car's mission is to achieve $\mathrm{pos}(t) = 0.6$ before the time limit of $t = 100$ episodes. Our study explores various initial settings for $\mathrm{pos}(0)$, $\mathrm{vel}(0)$, and the function f to find the cause of the vehicle's failure to reach its target. In our study, we began by collecting data by assigning different initial values to the variables $\mathrm{pos}(0)$ and $\mathrm{vel}(0)$, which were treated as external (exogenous) variables. We also utilized various combinations of pretrained neural networks as the decision-making mechanism for acceleration. The action controller in the mountain car scenario employs a neural network characterized by a rectangular architecture with varied dimensions. The sigmoid function serves as the activation mechanism for both the input and hidden layers, whereas the Tanh function is utilized for the output layer. This approach represents a modification of the methodology detailed in [15] and [29].

By executing multiple initial valuations with distinct neural networks, we generated a substantial set of traces, each indicating whether the car reached its destination within 100 episodes.

D. Case Study 2: Lunar Lander

In this case study, the space lander is initially positioned at a certain altitude from the ground, aiming to land on the

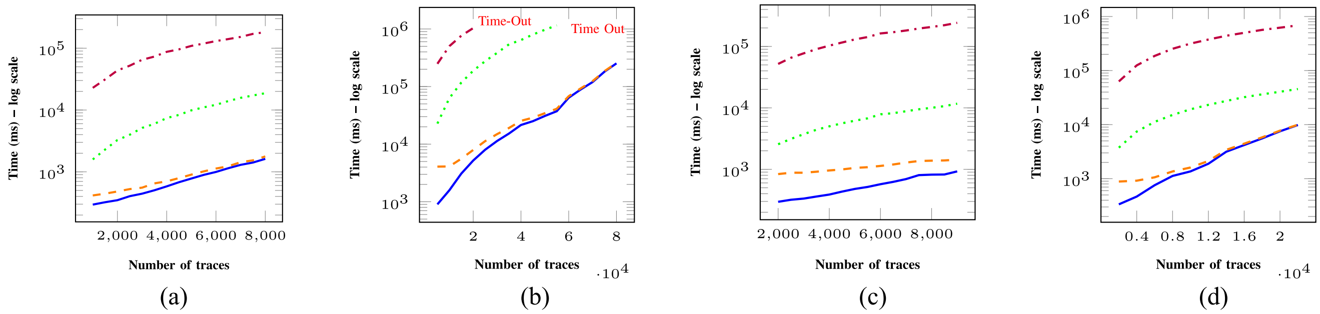

Fig. 6. Comparison of four modes of our implementation for various case studies. The legend is as follows: (a) —— represents *Abs_DA*, (b) - - - represents *Abs_Z3*, (c) ...... represents *Only_DA*, and (d) -.-.- represents *Only_Z3*.

designated landing pad. The landing pad is always located at $(0 \pm \epsilon, 0 \pm \epsilon)$. In the Lunar Lander system, there are eight variables (e.g., x and y coordinates, velocity, angular velocity, angle, etc.), which are intrinsic to the system, and an additional seven variables that are configured to represent the environment (e.g., wind, gravity, turbulence power, etc.). For a comprehensive overview of this case study, refer to [24]. The exogenous variables we consider are four different values for wind: $\{0, 5, 10, 15\}$, three values for gravity: $\{-8, -10, -12\}$, and three values for turbulence power: $\{0.8, 1.5, 2\}$. Moreover, in our experiment, we focus on a subset of the endogenous variables, specifically $\text{pos}_x(t)$, $\text{pos}_y(t)$, $\text{vel}_x(t)$, $\text{vel}_y(t)$, and $\text{action}(t)$. These variables correspond to the horizontal and vertical positions, horizontal and vertical velocities, and the action controlling the engines of the lander, respectively.

In our model, action denotes a learning-based function f, which is implemented using various pretrained neural networks with different dimensions

$$\text{action}(t) = f\big(\text{pos}_x(t), \text{pos}_y(t), \text{vel}_x(t), \text{vel}_y(t)\big).$$

In the Lunar Lander environment, there are four discrete actions available for controlling the lander, denoted as $\text{action} = \{0, 1, 2, 3\}$.

1) 0: Do nothing.
2) 1: Fire the left orientation engine.
3) 2: Fire the main engine.
4) 3: Fire the right orientation engine.

The action controller designed for the Lunar Lander experiment is based on deep reinforcement learning principles, as explored in [30]. The neural networks employed in this experiment are rectangular in shape and utilize the rectified linear unit (ReLU) activation function to introduce nonlinearity and enhance the learning capability of the model. We performed multiple simulations with varying initial values for $\text{pos}_x(0)$, $\text{pos}_y(0)$, $\text{vel}_x(0)$, $\text{vel}_y(0)$, wind, gravity, and turbulence each paired with different neural networks. This procedure produced a substantial set of traces, with each trace indicating whether the lander successfully landed on the landing pad within the time frame of $t < 500$ episodes or not.

E. Case Study 3: F-16 Autopilot MPC Controller [25]

This benchmark models both the inner-loop and outer-loop controllers of the F-16 fighter jet. We explore two scenarios. The first scenario involves reaching a specified altitude set point while maintaining a certain speed. The second scenario tests whether the automated collision avoidance system can recover the aircraft from a critical moment.

1) First Scenario: In this scenario, the aircraft's goal is to reach a certain altitude while maintaining a specified speed within a timeline of t. There are 16 state variables (e.g., altitude, airspeed, pitch, yaw, roll, power–lag, angle of attack (AoA) noted as α, etc.). Our exogenous variables are the initial settings for altitude(0), $\alpha(0)$, airspeed(0), pitch(0), and the power lag that the engine suffers (power–lag). Our endogenous variables are altitude(t), $\alpha(t)$, airspeed(t), pitch(t), power–lag(t), and the actions of the autopilot system for $t > 0$, which include changing the throttle $\delta_t(t)$ and adjusting the angle of the elevators $\delta_e(t)$ to control the pitch (nose up or down). In this experiment, we investigate the actions ($\delta_t(t)$ and/or $\delta_e(t)$) that determine whether the plane succeeds or fails in reaching the desired checkpoint, achieving the desired speed, or violating aircraft limits, such as upward acceleration, AoA, or minimum airspeed, that could lead to stalling.

2) Second Scenario: Here, we place the aircraft in a critical position near the ground to evaluate its collision avoidance system. This scenario involves using a larger set of variables, thereby increasing the dimensionality of our problem compared to the previous scenario. These critical moments involve high degrees of pitch, roll, and yaw, as well as low airspeed near the ground, which may lead to failures, such as ground collision and violations of the aircraft's aerodynamic limits. Our exogenous variables are the initial settings for altitude(0), airspeed(0), pitch(0), $\alpha(0)$, yaw(0), roll(0), and power–lag, while the endogenous variables are altitude(t), airspeed(t), pitch(t), yaw(t), roll(t), for $t > 0$ and the actions of the autopilot system. These actions include adjusting the degree of the rudder $\delta_r(t)$ to change the yaw of the plane, changing the degree of the aileron $\delta_a(t)$ to modify the roll of the plane, and controlling the throttle $\delta_t(t)$ and elevator $\delta_e(t)$. As in the previous scenario, we are examining the autopilot decisions that influence whether the aircraft can successfully recover from a potential collision or avoid violating aerodynamic constrains. Additionally, we aim to identify the actual cause of the failures.

F. Performance Analysis

Fig. 6(f)–(h) illustrate the results of our experiments for the mountain car, Lunar Lander, and both F-16 simulation

TABLE I
EXPERIMENT ON 1000 TRACES

Case Study	Algorithm	α	Refinement steps	Time (ms)
Mountain Car	Abs_DA	0.01	28	1024
		0.05	7	475
		0.1	2	102
	Abs_Z3	0.01	22	9793
		0.05	6	4757
		0.1	2	1306
Lunar Lander	Abs_DA	0.01	24	494
		0.05	7	396
		0.1	3	150
	Abs_Z3	0.01	19	2809
		0.05	4	794
		0.1	3	239

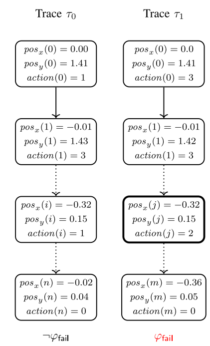

Fig. 7. Simulated traces in Lunar Lander and causal effect of decision by the main engine.

scenario, respectively. Indeed all graphs show a similar profile in terms of the behavior of the four modes of experiments mentioned in Section V-A.

As shown in the graphs, the abstraction algorithms (*Abs_DA* and *Abs_Z3*) demonstrate significantly better performance by orders of magnitude than the conventional solvers (*Only_DA* and *Only_Z3*), with the latter exhibiting exponential growth in runtime with an increasing number of traces. This demonstrates the effectiveness of our abstraction-refinement technique: it identifies the actual causes of failures while running much faster than techniques on concrete traces. As shown in Fig. 6(f), our technique processes up to 80 000 traces in under 250 s, whereas *Only_Z3* times out with threshold 1200 s at 20 000 traces and *Only_DA* at 55 000 traces.

Notably, *Abs_DA* outperforms *Abs_Z3*, and *Only_DA* shows better performance than *Only_Z3*. This observation can be attributed to the fundamental differences between SMT solvers, which focus on logical consistency, and the searching methods developed in data analysis libraries, which are tailored for efficient searching in large datasets.

In Table I, we present a comparison between different valuations of the parameter α, which represents the subset size of $\check{h}$. We conducted an experiment to find an optimal value for α. Our findings indicate that a very small α may require numerous refinements, as it needs to add more traces to identify the cause, which is inefficient. On the other hand, large values of α needs fewer refinements, but the under-approximation function has to process a larger amount of data, which increases the processing time. Therefore, there is a tradeoff between the number of refinements and the total time spent on them. We note that for row that have equal α, we shuffle the trace set, which impact computing the under-approximation.

G. *Causality Analysis*

This section demonstrates an important aspect of this research in investigating the actual cause of safety failures in CPS to *explain* the underlying reason. Our case studies involve simulations that specifically focus on the intersection of AI-enabled decision-making (mountain car and Lunar Lander), environmental dynamics feedback, and the correctness of a non-AI controller within an F-16 aircraft simulation.

1) Mountain Car: In Example 3 (see Fig. 4), we prove that making a poor decision to accelerate to the right (i.e., $\text{action}(0) = 1$) leads to failing in reaching the mountain top (i.e., formula φ_{fail}). Instead, in the counterfactual scenario we observe that it is necessary to accelerate to the left to gain momentum in order to climb the mountain. This not only shows the earliest bad decision by the controller but also identifies the "but-for" scenario, meaning what would have happened if a different action was taken. Additionally, counterfactual reasoning demonstrates how to fix the bad decision made by the neural network.

2) Lunar Lander: We observe that when there is a strong wind from left to right, some controllers tend to overuse the right engine, resulting in $\text{action} = 3$ during the initial steps. This causes the lander to drift to the left. However, we observe that even in this situation where the lander is positioned to the left of the landing pad, the controller can use its left engine, $\text{action} = 1$, to move the lander to the right and land safely. However, some controllers use their main engine, $\text{action} = 2$, resulting in the lander not reaching the landing pad. This results in $\text{pos}_x(t) < 0 - \epsilon$, constituting a failure.

To illustrate this further, Fig. 7 shows two traces starting from the same point but taking different actions in the first step. Dotted transitions means the next state is not the immediate next time step. The final state of the traces τ_0 and τ_1 is the nth and mth state, respectively. In trace τ_0, $\text{action}(0) = 1$, while in trace τ_1, $\text{action}(0) = 3$. However, in both traces, the controllers overuse the right engine in the initial steps (both controllers in τ_0 and τ_1 use the right engine $\text{action}(1) = 3$), causing the lander to drift far to the left, resulting in $\text{pos}_x(i) = -0.32$ in τ_0 and $\text{pos}_x(j) = -0.32$ in τ_1. At this state, where in both scenarios the lander has the same position and setting, the controller in τ_1 decides to use the main engine $\text{action}(j) = 2$, while the controller in τ_0 opts to use the left engine $\text{action}(i) = 1$ to move the lander to the right. These decisions under similar conditions lead to the failure of τ_1 (i.e., $\text{pos}_x(m) = -0.36 < 0 - \epsilon$) and the success of τ_0 (i.e., $0 - \epsilon < \text{pos}_x(n) = -0.02 < 0 + \epsilon$). This finding indicates that the failure in τ_1 using decision $\text{action}(j) = 2$, while the counterfactual scenario in τ_0 succeeds with a different decision $\text{action}(i) = 1$, highlighting that $\text{action}(j) = 2$ in τ_1 is the actual cause of the failure.

3) F-16 Autopilot Simulation: Here, we identify the cause of failures and analyze counterfactual scenarios (alternative actions) under the same conditions that could lead to success. When the aircraft needs to gain altitude at *low speed*, some traces show the controller lowering the nose to gain speed and

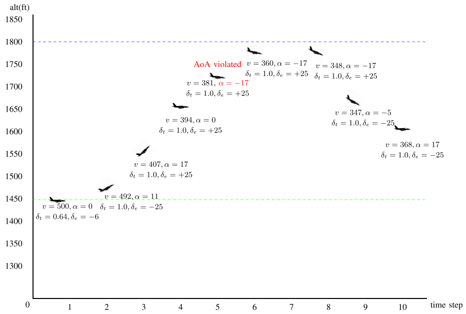

Fig. 8. F-16 scenario leads to failure due to a violation of the AoA limit.

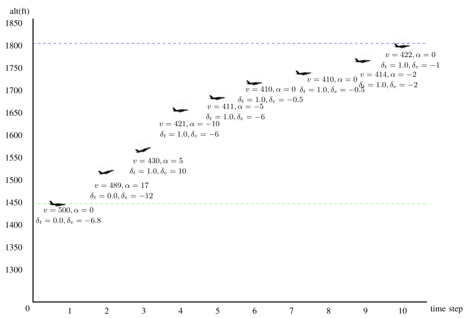

Fig. 9. F-16 counterfactual scenario leads to success.

avoid stalling before attempting to climb. This approach results in a loss of altitude and insufficient time to reach the desired altitude within the specified time frame, leading to failure. However, in counterfactual scenarios, the controller opts to gain speed by using more throttle and then gradually raises the nose using the elevators, eventually reaching the desired altitude. This demonstrates that the decision to lower the nose is the actual cause of the failure to reach the desired altitude within the specified time frame.

In another scenario, when transitioning from a lower to a higher altitude, some traces show controllers using excessive elevator and throttle, which places the aircraft in a danger zone and violates the AoA limits, leading to catastrophic failure. However, in alternative counterfactual scenarios with the same starting conditions, the controller gradually uses the throttle and adjusts the elevator more cautiously. This approach allows the aircraft to reach the desired altitude without violating its aerodynamic limits.

To illustrate the latter scenario in detail, Figs. 8 and 9 show two flight real paths starting from the same altitude, altitude(1) $= 1450$, and the same speed, airspeed(1) $= 500$, with the goal of reaching an altitude of altitude(n) $= 1800$. This process should occur within a specified time frame while not violating aircraft limits. In Fig. 8, the controller starts by using throttle $\delta_t(1) = 0.64$ and setting the elevator to a negative position, $\delta_e(1) = -6$, to achieve a positive pitch angle. This decision continues in subsequent steps in a more extreme manner, with $\delta_t(2) = 1.0$ (full throttle) and $\delta_e(2) = -25$, resulting in nearly a 45-degree pitch. Next, to counteract this situation, the controller attempts to use $\delta_e(3) = 25$ and $\delta_e(4) = 25$ to stabilize the aircraft's sharp nose-up attitude, leading to a negative AoA, $\alpha(5) = -17$. Since the aircraft's maximum negative AoA limit is -15, $\alpha(5) = -17$ violates this limit, and the controller fails to achieve its objective. On the contrary, in the counterfactual scenario (see Fig. 9), the controller starting with less aggressive throttle and elevator adjustments, such as $\delta_t(1) = 0$ and $\delta_e(1) = -6.8$, resulting in a slight pitch. This strategy continues similarly with $\delta_t(2) = 0$ and $\delta_e(2) = -12$, avoiding harsh climbs to reach the destination. By examining this scenario, we find that in the first time step, Fig. 8 makes the decisions $\delta_t(1) = 0.64$ and $\delta_e(1) = -6.8$, while Fig. 9 makes $\delta_t(1) = 0.0$ and $\delta_e(1) = -6.8$ under the same conditions (same altitude, speed, etc.). This counterfactual example shows that an alternative decision by the controller leads to success, providing sufficient evidence that the initial decision is the actual cause of failure.

VI. Related Work

There is a wealth of research on causality analysis in the context of embedded and component-based systems from different perspectives. In [2], [3], [4], [6], [7], [8], and [9], a new structure of formal causal analysis is proposed that can serve as a substitute for the HP causal model. This approach is distinct from our work, which utilizes a framework of causal analysis to identify the cause of a specific effect. Recently, there has been great interest in using temporal logics to reason about causality and explaining bugs [10], [11], [12], [13]. However, these lines of work either focus on only modeling aspects of causality or do not address the problem of scalability in automated reasoning about causality, which inherently involves a combinatorial blow up for counterfactual reasoning. In the CPS domain, using causality to repair AI-enabled controllers has recently gained interest [15]. This work explored the construction of HP models on AI-enabled controllers, the search for the cause of failure using a search algorithm, and the verification of these causes using HP constraints. In contrast, our work focuses on identifying the cause of failure efficiently in traces using HP constraints and proposes an efficient method for doing so. In [31], causal analysis is performed on system models and system execution traces. In contrast, our algorithm is designed to efficiently identify the cause of any potential failure. Additionally, our work is focused on systems, such as CPS, that interact with their environment.

Although the idea of abstracting causal models in terms of structural equations has been studied in [18], [19], and [20], these works do not attempt to establish a relation between *actual* causes in the abstract and concrete causal models. In the studies [18], [19], [20], the concept of abstraction in causal models was introduced, along with the preliminaries required to construct an abstraction function that maps low-level variables to high-level variables. The work in [18] presents a more general form of abstraction, while [19], [20] focus on the concept of intervention in causal models and how to build an abstraction that preserves them. The distinction between our work and these studies lies in our objective; we are not aiming to construct causal models, but rather, we are utilizing abstraction to identify the cause of an effect in a more efficient manner.

In [10] and [32], the concept of explaining counterexamples returned from the model checker is proposed, with one focusing on specifications in LTL format and the other in HyperLTL format. However, in our work, we aim to efficiently identify the cause of failure in an embedded system.

VII. Conclusion

We concentrated on designing an efficient technique to reason about actual causality. We proposed an SMT-based formulation to determine whether for an input transition system or a set of traces and a state formula (the effect), there exists an actual cause. Since identifying an actual cause involves counterfactual reasoning and, hence, a combinatorial blow up, we also introduced an efficient heuristic based on abstraction refinement. We evaluated our techniques on three case studies from the CPS domain: AI-enabled controllers for a 1) mountain car; 2) Lunar Lander [24]; and 3) an MPC controller for an F-16 autopilot simulator [25].

One natural extension is to consider *probabilistic* actual causality, where either occurrence of events in the system are associated with probabilities, or, data points follow some distribution. Another important direction is causal models where the system is partially observable.

References

[1] Y. Zhou and E. A. Lee, "Causality interfaces for actor networks," *ACM Trans. Embedd. Comput. Syst.*, vol. 7, no. 3, pp. 1–35, 2008.

[2] M. Broy, "Time, causality, and realizability: Engineering interactive, distributed software systems," *J. Syst. Softw.*, vol. 210, Apr. 2024, Art. no. 111940.

[3] G. Goessler and L. Astefanoaei, "Blaming in component-based real-time systems," in *Proc. Int. Conf. Embedd. Softw.(EMSOFT)*, 2014, pp. 1–10.

[4] G. Gössler and J.-B. Stefani, "Causality analysis and fault ascription in component-based systems," *Theor. Comput. Sci.*, vol. 837, pp. 158–180, Oct. 2020.

[5] S. Cherrared, S. Imadali, E. Fabre, and G. Gößler, "SAKURA a model based root cause analysis framework for vIMS," in *Proc. 17th Annu. Int. Conf. Mobile Syst., Appl., Services(MobiSys)*, 2019, pp. 594–595.

[6] G. Gößler, O. Sokolsky, and J. Stefani, "Counterfactual causality from first principles?" in *Proc. 2nd Int. Workshop Causal Reason. Embedd. Safety-Critical Syst. Technol. (CREST)*, 2017, pp. 47–53.

[7] S. Wang, Y. Geoffroy, G. Gößler, O. Sokolsky, and I. Lee, "A hybrid approach to causality analysis," in *Proc. 6th Int. Conf. Runtime Verif. (RV)*, 2015, pp. 250–265.

[8] G. Gößler and D. L. Métayer, "A general trace-based framework of logical causality," in *Proc. 10th Int. Symp. Formal Aspects Compon. Softw. (FACS)*, 2013, pp. 157–173.

[9] G. Gößler, D. L. Métayer, and J. Raclet, "Causality analysis in contract violation," in *Proc. 1st Int. Conf. Runtime Verif. (RV)*, 2010, pp. 270–284.

[10] N. Coenen et al., "Explaining hyperproperty violations," in *Proc. 34th Int. Conf. Comput. Aided Verif. (CAV)*, 2022, pp. 407–429.

[11] B. Finkbeiner and A. Kupriyanov, "Causality-based model checking," in *Proc. 2nd Int. Workshop Causal Reason. Embedd. Safety-Critical Syst. Technol. (CREST)*, 2017, pp. 31–38.

[12] N. Coenen, B. Finkbeiner, H. Frenkel, C. Hahn, N. Metzger, and J. Siber, "Temporal causality in reactive systems," in *Proc. 20th Int. Symp. Autom. Technol. Verif. Anal. (ATVA)*, 2022, pp. 208–224.

[13] B. R. B. Finkbeiner, H. Frenkel, and J. Siber, "Checking and sketching causes on temporal sequences," in *Proc. 21st Int. Symp. Autom. Technol. Verif. Anal. (ATVA)*, 2023, pp. 314–327.

[14] P. Lu, M. Cleaveland, O. Sokolsky, I. Lee, and I. Ruchkin, "Repairing learning-enabled controllers while preserving what works," in *Proc. 15th Int. Conf. Cyber-Phys. Syst. (ICCPS)*, 2024, pp. 1–11.

[15] P. Lu, I. Ruchkin, M. Cleaveland, O. Sokolsky, and I. Lee, "Causal repair of learning-enabled cyber-physical systems," in *Proc. IEEE Int. Conf. Assured Auton. (ICAA)*, 2023, pp. 1–10.

[16] J. Y. Halpern, *Actual Causality*. Cambridge, MA, USA: MIT Press, 2016.

[17] G. Aleksandrowicz, H. Chockler, J. Y. Halpern, and A. Ivrii, "The computational complexity of structure-based causality," *J. Artif. Intell. Res.*, vol. 58, pp. 431–451, Feb. 2017.

[18] P. K. Rubenstein et al., "Causal consistency of structural equation models," in *Proc. 33rd Conf. Uncertain. Artif. Intell. (UAI)*, 2017, pp. 1–15.

[19] S. Beckers, F. Eberhardt, and J. Y. Halpern, "Approximate causal abstractions," in *Proc. 35th Conf. Uncertain. Artif. Intell. (UAI)*, 2019, pp. 606–615.

[20] S. Beckers and J. Y. Halpern, "Abstracting causal models," in *Proc. 33rd AAAI Conf. Artif. Intell. (AAAI)*, 2019, pp. 2678–2685.

[21] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. 14th Int. Conf. Tools Algorithms Constr. Anal. Syst. (TACAS)*, 2008, pp. 337–340.

[22] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[23] W. McKinney, "Data structures for statistical computing in python," in *Proc. 9th Python Sci. Conf.*, 2010, pp. 56–61.

[24] G. Brockman et al., "OpenAI Gym," 2016, *arXiv:1606.01540*.

[25] P. Heidlauf, A. Collins, M. Bolender, and S. Bak, "Verification challenges in F-16 ground collision avoidance and other automated maneuvers," in *Proc. 5th Int. Workshop Appl. Verif. Continuous Hybrid Syst. (ARCH)*, 2018, pp. 208–217.

[26] A. Rafieioskouei and B. Bonakdarpour, "Efficient discovery of actual causality using abstraction-refinement," 2024, *arXiv:2407.16629*.

[27] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods Syst. Design*, vol. 19, no. 1, pp. 7–34, 2001.

[28] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. 12th Int. Conf. Comput. Aided Verif. (CAV)*, 2000, pp. 154–169.

[29] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," 2018, *arXiv:1811.01828*.

[30] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] F. Leitner-Fischer and S. Leue, "Towards causality checking for complex system models," in *Proc. 8th Dagstuhl-Workshop Modellbasierte Entwicklung eingebetteter Systeme (MBEES) Model-Based Develop. Embedd.Syst.*, 2012, pp. 71–80.

[32] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. J. Trefler, "Explaining counterexamples using causality," in *Proc. 21st Int. Conf. Comput. Aided Verif. (CAV)*, 2009, pp. 94–108.