‘Put the Car on the Stand’: SMT-based Oracles for Investigating Decisions

Samuel Judson
samuel.judson@yale.edu
Yale University
USA

Matthew Elacqua
matt.elacqua@yale.edu
Yale University
USA

Filip Cano
filip.cano@iaik.tugraz.at
Graz University of Technology
Austria

Timos Antonopoulos
timos.antonopoulos@yale.edu
Yale University
USA

Bettina Könighofer
bettina.koenighofer@iaik.tugraz.at
Graz University of Technology
Austria

Scott J. Shapiro
scott.shapiro@yale.edu
Yale Law School & Yale University
USA

Ruzica Piskac
ruzica.piskac@yale.edu
Yale University
USA

Abstract

Principled accountability in the aftermath of harms is essential to the trustworthy design and governance of algorithmic decision making. Legal theory offers a paramount method for assessing culpability: putting the agent ‘on the stand’ to subject their actions and intentions to cross-examination. We show that under minimal assumptions automated reasoning can rigorously interrogate algorithmic behaviors as in the adversarial process of legal fact finding. We use the formal methods of symbolic execution and satisfiability modulo theories (SMT) solving to discharge queries about agent behavior in factual and counterfactual scenarios, as adaptively formulated by a human investigator. We implement our framework and demonstrate its utility on an illustrative car crash scenario.

CCS Concepts

• **Applied computing** → **Law**; • **Theory of computation** → **Automated reasoning**; **Logic and verification**.

Keywords

algorithmic decision making, algorithmic accountability, formal methods, SMT solving, symbolic execution

ACM Reference Format:
Samuel Judson, Matthew Elacqua, Filip Cano, Timos Antonopoulos, Bettina Könighofer, Scott J. Shapiro, and Ruzica Piskac. 2024. ‘Put the Car on the Stand’: SMT-based Oracles for Investigating Decisions. In *Symposium on Computer Science and Law (CSLAW ’24), March 12–13, 2024, Boston, MA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3614407.3643699

CSLAW ’24, March 12–13, 2024, Boston, MA, USA

ACM ISBN 979-8-4007-0333-1/24/03
https://doi.org/10.1145/3614407.3643699

1 Introduction

Our lives are increasingly impacted by the automated decision making of AI. We share roads with autonomous vehicles, as healthcare providers use algorithms to diagnose diseases and prepare treatment plans, employers to automate hiring screens, and even judges to analyze flight and recidivism risks. Though the creators of AI often intend it to improve human welfare, it is a harsh reality that algorithms often fail. Automated decision makers (ADMs) are now deployed into roles of immense social responsibility even as their nature means they are not now, and likely will never be, trustworthy enough to do no harm. When autonomous vehicles drive on open roads they cause fatal accidents [Smiley 2022]. Classification and scoring algorithms perpetuate race- and gender-based biases in hiring and recidivism evaluations, both characteristics legally protected from discrimination in many countries [Angwin et al. 2016; Dastin 2018; Kroll et al. 2017]. Both the rule of law and a more ordinary sense of justice demand that society hold accountable those responsible and answerable for harms. In this work, we investigate how formal methods can aid society and the law in providing accountability and trust in a clear, rigorous, and efficient manner through SMT-based automated reasoning.

Ideally, computer scientists would verify decision making algorithms to confirm their correctness before deployment. Using the techniques of program verification discrimination and other social harms would be automatically detected and eliminated by engineers before ADMs appear in the field. Unfortunately, the formal verification of these algorithms is undecidable in the general case, and even when theoretically possible will often require computational power that can make the task uneconomical, if not practically infeasible. Additionally, writing specifications for algorithmic decision making often treads onto contentious questions of law and policy with no universally agreed upon, let alone formalizable, answers [Kroll et al. 2017]. Nevertheless, assessing responsibility for harms remains vital to the safe use of ADMs.

To better understand the concept of accountability, consider a case in which one autonomous car hits another. We can ask: Which car is responsible for the accident? Which made the error, and

to what end? When human drivers crash, lawyers investigate the drivers' reasoning and actions. Did the drivers intend to hit the other car? Did the drivers know that an accident would occur? To infer drivers' intentions, lawyers engage in direct and indirect examination to uncover the decision logic that lead to the accident. Standard notions of due process view this opportunity to mount a defense as essential to justice. A person must be permitted to explain and justify themselves through arguing the facts of the case [Edwards 2021]. For example, if a human driver can convince a jury the crash was an unforeseeable accident, then they may be subject to lesser penalties than had they acted intentionally or negligently. A self-driving car cannot do likewise. Like a person, an ADM makes unsupervised decisions in complex environments which can lead to harm. But very much unlike a person, an algorithm cannot simply walk into a courtroom and swear to tell the whole truth.

ADMs leave us with the traditional need for explanation but – as a program and not a mind – without the traditional means for acquiring one. Still, a program can be translated into logic, and logic can be rigorously reasoned about. While an algorithm may not be able to defend itself under interrogation, we show the right formal method can absolutely advocate on its behalf. And at least in one sense, a formal method makes for a better witness than a human – while the human can lie, the car cannot. The provable rigor of our approach guarantees that whatever answers we get from the decision algorithm are both accurate and comprehensive.

1.1 Contribution

We developed a method and tool, soid, for applying automated reasoning to 'put the algorithm on the stand' in cases where its correct behavior cannot be reduced to a practically verifiable formal specification. Using soid, an *investigator* can pose tailored *factual* and *counterfactual* queries to better understand the functional intention of the decision algorithm, in order to distinguish accidents from honest design failures from malicious design consequences. Our method also generates counterexamples and counterfactuals to challenge flawed conclusions about agent behavior – just as would a human assisting in their own defense. We assume only access to a program A implementing a decision making algorithm $\mathcal{A}$ as granted by its *controller*. Our method supports three types of queries: *factuals* ('did the agent do...'), *might counterfactuals* ('might the agent have possibly done...'), and *would counterfactuals* ('would the agent have necessarily done...') [Lewis 2013]. We formally define each in §3. The distinction between 'would' and 'might' counterfactuals is foundational to their treatment in philosophy [Lewis 2013]. Logically, we implement 'might' counterfactuals using the $\exists$ operator, and 'would' counterfactuals using $\forall$.

We have also implemented an example of a domain-specific graphical user interface (GUI) that allows operation of soid without requiring technical expertise, but here describe how it works directly. The investigator starts from the logs of A, the factual information within capturing the state of the world as the agent perceived it. These logs can be easily translated into a first-order formalism as a sequence of equalities. For example, in the left panel of the car crash diagrammed in Figure 1 the information in the logs of A (blue at bottom) about the other car (red at left) might be

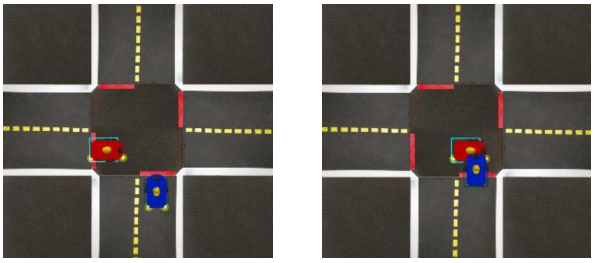


Figure 1: A broadside car crash rendered in the soid GUI.

encodable as a system of equalities

$$\varphi \equiv \texttt{agent1\_pos\_x} = 1.376 \wedge \cdots \wedge \texttt{agent1\_signal} = \text{RIGHT} \wedge \cdots . \quad (1)$$

In the GUI all such statements are translated into formal logic automatically. Using the automated reasoning method of symbolic execution, we can answer queries about what the car did under these constraints by checking formula entailment. Posing counterfactual queries requires manipulating the state of the world since such queries ask how would the agent react if the situation were different. Counterfactuals can be encoded by substituting constraint values

$$\varphi' \equiv \varphi[(\texttt{agent1\_pos\_x} = 1.376) \mapsto (\texttt{agent1\_pos\_x} = 1.250)]. \quad (2)$$

This formalism also allows us to reason about whole families of counterfactuals, which can be defined by relaxing the constraints and negating the original factual state as a valid model

$$\varphi'' \equiv \varphi[(\texttt{agent1\_pos\_x} = 1.376) \mapsto (1.0 \leq \texttt{agent1\_pos\_x} \leq 1.5)] \wedge \neg\varphi. \quad (3)$$

In this way, hypothetical-but-similar scenarios of interest to the investigator ('what if that car was outside instead of inside the intersection?', 'what if the car was signaling a turn instead of straight?') can be rigorously formalized to enable automated analysis of the agent's behavior.

We use SMT-based automated reasoning [Baldoni et al. 2018; Cadar et al. 2008; Moura and Bjørner 2008] for the oracle that answers the factual and counterfactual questions the investigator asks. Each query and its answer helps the expert investigator to build a body of knowledge, the Facts, about the decision logic used in the autonomous agent. The investigator then decides when to terminate the investigation when they have enough information for the final judgement of an agent's culpability.

1.2 Related Work

Formal methods for accountability are a burgeoning research topic, both in general [Baier et al. 2021a; Chockler and Halpern 2004; Datta et al. 2015; Feigenbaum et al. 2020; Halpern and Pearl 2005a,b; Küsters et al. 2010] and focused on specific domains including automated and economic decision making [Baier et al. 2021b,c; Ghosh and Meel 2019; Kroll et al. 2017; Su et al. 2015] and security [Feigenbaum et al. 2011; Künnemann et al. 2019]. In particular, a recent work uses such methods to investigate similar questions to ours under stronger modeling assumptions [Cano Córdoba et al. 2023]. Trustworthy algorithmic decision making is now a major focus

of classical formal methods research as well [Alshiekh et al. 2018; Christakis et al. 2021; Dreossi et al. 2019; Garcıa and Fernández 2015; Gehr et al. 2018; Katz et al. 2019; Singh et al. 2019]. Intention and its relationship to responsibility is a central focus of law and the philosophy of action, with a cross-discipline history dating back millennia [Beebee and Menzies 2019; Lewis 2013; Moore 2019; Starr 2021; Wachter et al. 2017], including an extensive modern focus on (often symbolic) automated decision making in particular [Bratman 1987; Cohen and Levesque 1990; Rao and Georgeff 1991]. Counterfactual reasoning is accordingly a significant topic in Explainable AI (XAI) [Adadi and Berrada 2018; Arrieta et al. 2020; Guidotti et al. 2018; Padovan et al. 2023], using both logical [Chockler and Halpern 2004; Halpern and Pearl 2005a,b] and statistical methods [Wachter et al. 2017]. SMT solving [Barrett et al. 2021; Moura and Bjørner 2008] and symbolic execution [Baldoni et al. 2018; Cadar et al. 2008], are both foundational topics in automated reasoning.

Technically, our approach differs from the already significant body of work in counterfactual analysis of algorithmic decision making in two significant ways: in our analysis of executable code 'as it runs' – rather than just of a partial component (such as an ML model in isolation) or of some higher-level mathematical abstraction of the system – and in our reliance on formal verification. By analyzing the code itself, rather than an idealized abstraction or particular model, we can capture behaviors of the entire software system: preprocessing, decision making, postprocessing, and any bugs and faults therein. This makes our analysis more complete. In §6.2, for example, we consider a hypothetical case study where an instance of API misusage – rather than any mistakes of logic within the code itself – undermines a machine learning decision. The decision and its consequences are not analyzable by considering only the (correct on its own terms) decision model alone.

Meanwhile, verification technologies allow us to analyze *all possible executions* obeying *highly expressive pre- and post-conditions*. SMT-based methods in particular provide the full expressiveness of first-order logic. As such, our approach can encode entire families of counterfactuals in order to provide a broad and thorough picture of agent decision making, so as to better interpret responsibility. Prevailing, often statistical methods, commonly focus more on gathering explanations prioritized on informal measures, such as minimality or diversity criteria, in order to demonstrate causality, see *e.g.*, [Mothilal et al. 2020; Wachter et al. 2017]. Informally, in computer science terms, this distinction is analogous to that in automated reasoning between methods for verification – which emphasize overall safety and the correctness of the set of all program traces provably meeting some logically expressed property – and methods like testing or fuzzing that focus on finding or excluding representative executions believed to exemplify that property [Abebe et al. 2022]. Of course (SMT-based) verification does have costs – it carries substantially more computational complexity than testing approaches, which can increase compute costs and limit scalability. Accordingly, we implement and benchmark the empirical efficacy of our method in a laboratory environment in §6. In human terms, our approach is analagous to enabling asking broad, positive questions about agent behavior under a coherent family of scenarios, rather than asking questions aimed primarily at generating or falsifying a particular claimed explanation for a (factual or counterfactual) decision. The work of [Cano Córdoba et al. 2023] and VerifAI [Dreossi et al. 2019] are the related approaches of which we are aware most similar to our own in goals and method, although the former requires stronger modeling assumptions while the latter sacrifices some formal guarantees for scalability.

2 Motivation

To illustrate the purpose of soid, we continue with the crash from Figure 1. In the left panel the autonomous vehicle $\mathcal{A}$ (blue at bottom) perceived that the other car (red at left) had its right turn signal on. Call this time t^*. When $\mathcal{A}$ entered the intersection – believing the action to be safe, even though it lacked the right-of-way – the other car proceeded straight, leading to a collision. Because it did not possess the right-of-way, $\mathcal{A}$ is culpable for the crash. This scenario forms the basis for our benchmarks in §6, where the specific question we investigate with soid is 'with what purpose did $\mathcal{A}$ move, and so to what degree is it culpable for the crash?'

Note that the actions of $\mathcal{A}$ are consistent with with three significantly different interpretations:

i a **reasonable** (or **standard**) $\mathcal{A}$ drove carefully, but proceeded straight as is common human driving behavior given the (perception of an) indicated right turn;
ii an **impatient** $\mathcal{A}$ drove with reckless indifference to the risk of a crash; and
iii a **pathological** $\mathcal{A}$ drove to opportunistically cause crashes with other cars, without unjustifiable violations of traffic laws such as weaving into an oncoming lane.

Even for the same act, these different interpretations will likely lead to drastically different liabilities for the controller under criminal or civil law. Interestingly, the natural language explanations for the i) *reasonable* and iii) *pathological* cars are identical: 'moving straight would likely not cause a crash, so proceeding would bring me closer to my goal'. Nonetheless, counterfactual queries (notated $\Box\!\!\rightarrow$) can help distinguish between these candidate interpretations.

at t^*

$\Box\!\!\rightarrow$ Could a different turn signal have led $\mathcal{A}$ to remain stationary?

$\Box\!\!\rightarrow$ If $\mathcal{A}$ had arrived before the other car, and that other car was not signaling a turn, would $\mathcal{A}$ have waited? (*e.g.*, to 'bait' the other car into passing in front of it?)

Interpretation i) is consistent with (*yes*, *no*), ii) with (*no*, *no*), and iii) with (*no*, *yes*). Note the adaptive structure of our questions, where the second query can be skipped based on the answer to the first. The goal of soid is to enable efficient and adaptive investigation of such queries, in order to distinguish the computational reasoning underlying agent decisions and support principled assessment of responsibility. Although autonomous vehicles provide an insightful example, soid is not limited to cyberphysical systems. In §6.2 we use soid to analyze a buggy application of a decision tree leading to a health risk misclassification.

2.1 Legal Accountability for ADMs

Before presenting the technical details of soid, we also overview how the philosophy and practice of legal accountability might apply to ADMs, and so motivate the analysis soid is designed to enable. We work in broad strokes as the legal liability scheme for ADMs is still

being developed, and so we do not want to limit our consideration to a specific body of law. This in turn limits our ability to draw specific conclusions, as disparate bodies of law often place vastly different importance on the presence of intentionality, negligence, and other artifacts of decision making.

A core principle of legal accountability is that the 'why' of a wrongful act is almost always relevant to evaluating how (severely) liable the actor is. In the words of the influential United States Supreme Court justice Oliver Wendell Holmes, 'even a dog distinguishes between being stumbled over and being kicked.' As every kick has its own reasons bodies of law often distinguish further – such as whether the 'why' is an active intent to cause harm. Though holding algorithmic agents accountable raises the many technical challenges that motivate soid, once we understand the 'why' of an algorithmic decision we can still apply the same framework of our ethical and legal practices we always use for accountability [Hallevy 2013; Kroll et al. 2017]. The algorithmic nature of a harmful decision does not invalidate the need for accountability: the locus of Holmes' adage lies in the harm to the victim, being justifiably more aggrieved to be injured on purpose or due to a negligent disregard of the risk of a kick than by an accidental contact in the course of reasonable behavior. In practice, even though criminal law and civil law each place different emphasis on the presence of attributes like intention and negligence in a decision, intention in particular almost always matters to – and often intensifies – an agent's liability. Any time the law penalizes an unintentional offense it will almost always punish an intentional violation as well, and should intention be present, the law will usually apply the greatest possible penalties authorized for the harm. Given the importance of recognizing intention, soid is designed to support rigorous and thorough *findings of fact* about algorithmic decisions from which a principled assessment of their 'why' can be drawn.

Taking a step back, it is deeply contentious whether ADMs now and in the future can, could, or should possess agency, legal personhood, or sovereignty, and whether they can ever be morally and legally responsible [Müller 2023]. Even the basic nature of computational decision making is a significant point of debate in artificial intelligence and philosophy, with a long and contentious history [Bratman 1987; Cole 2020]. For the moment, ADMs are not general intelligences. They will likely not for the foreseeable future possess cognition, agency, values, or theory of mind, nor will they formulate their own goals and desires, or be more than 'fancy toasters' that proxy the decision making agency and responsibility of some answerable controller. An algorithm is no more than a computable function implemented by symbolic manipulation, statistically-inferred pattern matching, or a combination thereof. Nonetheless, even working off the most stringent rejection of modern ADMs forming explicit knowledge or intentional states, following Holmes we can see there is still value in grading the severity of a harmful decision. It is deeply ingrained in our governing frameworks for legal and moral accountability that when acting with the purpose of harm an agent (or its controller) has committed a greater transgression than in the case where the harm was unintended.

In this work we sidestep whether and how computers can possess intentionality by viewing intention through a functionalist lens. Even for conscious reasoning, it is impossible to replay a human being's actual thought process during a trial. So in practice, legal definitions refer instead to an *ex post* rationalization of the agent's decisions made by the accountability process through the finding of fact. A person is assessed to have, *e.g.*, purposely caused harm if the facts show they acted in a way that is consistent with purposeful behavior. We can approach computational reasoning in much the same way, with an investigator making an *ex post* descriptive rationalization capturing their understanding of an ADM's decision making. This understanding then justifies a principled assessment of the controller's responsibility. For example, a controller can be assessed to have released into the world an ADM that the facts show acted in a way that is consistent with a purposeful attempt to cause harm. The design and algorithmic processes of the agent are otherwise irrelevant. How the ADM actually decision makes – whether through statistical inference or explicit goal-oriented decision logic or otherwise – is relevant only with respect to our ability to interrogate its decision making. This approach is consistent with soid, which is capable of analyzing arbitrary programs.

An investigator using soid to label an ADM as 'reasonable' or 'reckless' or 'pathological' or similar is, however, only the start. How such an assessment should then be interpreted and used by an accountability process is, ultimately, a policy question. The unsettled nature of the laws, policies, and norms that govern ADMs, both for now and into the future, means there are many open questions about the relevance of the intent of an ADM and its relationship to the intent of the controller. But we can consider the ramifications in broad strokes. For individuals harmed by ADMs (whether as consumers, other end-users, or just unlucky 'bystanders'), the situation seems little different than for human misconduct: the finding of intent amplifies the harm, and the victim can reasonably expect the accountability process to penalize the transgressor appropriately. More specific questions are harder. Should apparent intent in both the controller and ADM be assessed more harshly than in one or the other alone? Or would apparent intent in the controller render the actions of the ADM relevant only in how successfully the intent of the controller was carried out? How should an emergent 'algorithmic intent' traceable to software faults interact with any documented, contrary evidence of the intent of the controller? These questions lay beyond the scope of this work, but they are each dependent on our capacity to first recognize and distinguish the functional intent of the ADM, motivating our research goals.

For the controllers of ADMs (whether as programmers, vendors, owners, or sovereign states), it is a natural starting point to view them as responsible for the actions of their computational agents, just as they would (most often) be responsible for human agents acting on their behalf. With reward comes responsibility. If a controller profits from deploying an ADM, so must they bear the costs of its harms. Legal concepts governing humans acting on behalf or through each other or organizations are well-founded throughout, *e.g.*, agency and criminal law [Hallevy 2013; Legal Information Institute 2023]. These mechanisms may be either directly applicable or can form the basis for analogous systems governing algorithmic accountability. For example, just as a business is expected to adequately prepare (i.e., train) a human agent to operate on their behalf without causing harm, a controller can be expected to adequately prepare (*i.e.*, design or train) a computational agent. What standard the controller sets internally *ex ante* before deploying the

ADM is primarily relevant insofar as it provides confidence to the controller the ADM will not be found *ex post* to have operated in a way consistent with an intent to harm – and so carry with it a corresponding increase in liability.

Grounding our approach in the functionalist perspective also helps us manage difficult questions about the validity of anthropomorphizing algorithmic systems through the use of language like 'intent', 'beliefs', or 'reasonableness', as we ourselves have done throughout §2. It is not immediately clear such language is intrinsically confusing or harmful: the use of such labels in characterizing automated decision making is decades-old, to the extent that consideration of whether and how machines can form intentional states has informed how prevailing approaches in the philosophy of action now capture whether and how humans form them [Bratman 1987; Bratman et al. 1988]. Moreover, as accountability processes begin to wrestle with algorithmic decision making some anthropomorphization is perhaps unavoidable, due to the often heavily analogical nature of legal reasoning [Levi 1947]. We ourselves invoked the analogy of Holmes to frame our discussion. The validity of some such analogies are in some cases already contentious. For example, whether the 'creativity' required to earn authorship under copyright law must necessarily be human is under active consideration in litigation and scholarship concerning generative models [tha [n. d.]; Dornis 2020]. On the other hand, the negative consequences of anthropomorphizing ADMs has been itself widely recognized in scholarship and science fiction dating back decades: it can cause us to, *e.g.*, ascribe to machines and their actions non-existent morality and common sense, or grow attached to them in ways that cause us to disregard their harms or cloud our judgement of their true capabilities and limitations.

To avoid conflation, perhaps machine analogues to terms like 'intention' will arise. But wherever the legal and policy language settles, the core philosophical principle – that a functional interpretation of the 'why' of a decision matters for accountability – will hold. So long as the philosophical (and computational) principles remain, the goals of our research should likewise remain applicable no matter what norms of language develop.

3 Technical Background

In this section we present some relevant foundations for soid from formal and automated reasoning.

3.1 Programs and Traces

Let A be the program instantiating a decision algorithm $\mathcal{A}$. A operates over a finite set of program variables $\mathsf{var}(A) = V = \{v_1, \ldots, v_n\}$. We view $\mathsf{var}(A)$ as a union of disjoint subsets $V = I \cup D$. The set $D = \{vd_1, \ldots, vd_{n_D}\}$ is the set of internal *decision* variables. The set of input variables $I = E \cup S$ is itself partitioned into sets of *environment* variables $E = \{ve_1, \ldots, ve_{n_E}\}$ and *state* variables $S = \{vs_1, \ldots, vs_{n_S}\}$. Therefore $n = n_E + n_S + n_D$. E is composed of variables encoding input sources external to the agent, while S is composed of variables encoding internal state.

Every v_i is associated with a domain $\mathcal{D}_{v_i}$. A *state* is the composition of the variable assignments at that point of the execution $\sigma \in \mathcal{D} = (\mathcal{D}_{v_1} \times \cdots \times \mathcal{D}_{v_n})$. Given $\sigma = (\ell_1, \ldots, \ell_n) \in \mathcal{D}$, we denote the restriction to only environment variables as $\sigma|_E = (\ell_1, \ldots, \ell_{e_{n_E}}) \in \mathcal{D}_{e_1} \times \cdots \times \mathcal{D}_{e_{n_E}}$, and similarly for $\sigma|_S$, $\sigma|_I$, and $\sigma|_D$. A *trace* $\tau = \sigma_1\sigma_2\sigma_3 \ldots$ is a (possibly infinite) sequence of states. We access states by $\tau(t) = \sigma_t$, and values of variables at states by $\sigma(v_i) = \ell_i$. The set of possible traces is governed by a transition relation $R \subseteq \mathcal{D} \times \mathcal{D}$, so that $\tau(t) = \sigma$ and $\tau(t+1) = \sigma'$ may occur within some τ only if $(\sigma, \sigma') \in R$. The program A encodes a partial transition relation, R_A, with the constraint that $(\sigma, \sigma') \in R_A$ requires that $\forall i \in [n_E].\ \sigma(ve_i) = \sigma'(ve_i)$. That is, by definition A cannot define how the environment $\mathcal{E}[\mathcal{A}]$ updates the ve_i, as that capacity is exactly the distinction between a program and the environment it runs within.

We work with statements over the program variables in the logic `QF_FPBV`. The available domains are those of floating points and bitvectors. An expression e is built from variables composed with the constants and function symbols defined over those domains, *e.g.*, $(\text{fp.to\_real b011}) + 2.34 \cdot v_{14}$. A formula φ is built from expressions and relation symbols composed using propositional operators, *e.g.*, the prior expression could extend to the formula $(\text{fp.to\_real b011}) + 2.34 \cdot v_{14} \geq -1.87$. If φ is a formula over $\mathsf{var}(A) = V$, notated $\varphi(V)$, and $\sigma = (\ell_1, \ldots, \ell_n) \in \mathcal{D}$, then we write $\sigma \models \varphi(V)$ if the constant formula that results from substituting each ℓ_i for v_i evaluates to True. We use φ and ψ when writing formulas over I that represent scenarios, β when writing formulas over D representing decisions made, and Π when writing formulas over V representing whole program executions.

A symbolic state $\hat{\sigma} = \hat{\mathcal{D}} = (\hat{\mathcal{D}}_{\hat{v}_1} \times \cdots \times \hat{\mathcal{D}}_{\hat{v}_n}, \pi_{\hat{\sigma}})$ is defined over a set of symbolic variables $\mathsf{symvar}(A) = \hat{V} = \{\hat{v}_1, \ldots, \hat{v}_n\}$, with $\hat{I}$, $\hat{E}$, $\hat{S}$, and $\hat{D}$ defined analogously. Each $\hat{\mathcal{D}}_{\hat{v}_i}$ augments the concrete domain $\mathcal{D}_{v_i}$ by allowing $\hat{v}_i$ to reference an expression e_i over a set of symbolic values $\{\alpha_i\}_{i \in [k]}$, *e.g.*, $e_i = \alpha_i$ or $e_i = 2\alpha_j + 3.0$. We write $\hat{\sigma} \models \varphi(\hat{V})$ for formulas over $\mathsf{symvar}(A)$ analogously to the concrete case, still in `QF_FPBV`. The *path constraint* $\pi_{\hat{\sigma}}(\alpha_1, \ldots, \alpha_n)$ is such a formula over the α_i, which captures their possible settings.

3.2 SMT-based Program Analysis

We overview SMT solving and symbolic execution, and refer the reader to [Moura and Bjørner 2008] and [Baldoni et al. 2018; Cadar et al. 2008] respectively for greater detail.

SMT Solving. Satisfiability modulo theory (SMT) solving is a form of automated theorem proving that computes the satisfiability (and, by duality, validity) of formulas in certain fragments of first-order (FO) logic. SMT solvers – we use the state-of-the-art Z3 [Moura and Bjørner 2008] – are also able to return satisfying models when they exist. In the case of validity queries, these models are concrete counterexamples to the disproven theorem. An SMT formula Φ is a FO-formula over a decidable theory T. In this work, we set T = `QF_FPBV`, the combination of quantifier-free formulas in the theories of floating-points and bitvectors [Barrett et al. 2021]. We require support for floating-point statements due to their centrality in machine learning.

Symbolic Execution. One of the great successes of SMT-based program analysis, symbolic execution explores the reachable paths of a program P when executed over $\hat{V}$. Concrete values are computed exactly, assignments to or from symbolic-valued variables update

their expressions, and branching conditions update the path constraints. For a branch condition $b(\hat{V})$ reached at symbolic state $\hat{\sigma}_j$, such as a guard for an if-statement or while-loop, an SMT solver is invoked to check which branches are feasible under π_j, *i.e.*, whether $\Phi \equiv b(\hat{V}) \wedge \pi_j$ and/or $\Phi' \equiv \neg b(\hat{V}) \wedge \pi_j$ are satisfiable. If only one is, the execution continues along it and its path constraints are updated. For example, if only Φ' is satisfiable then $\pi_j \leftarrow \pi_j \wedge \neg b(\hat{V})$. If both are satisfiable, the execution can fork in order to explore all reachable paths and produce a set of constraint formulas $\{\pi_i\}_{i\in[ct]}$ encoding each path at termination. By setting initial constraints on the input variables, symbolic execution can narrow the search space to only the paths of executions meeting preconditions.

3.3 Counterfactual Reasoning

Counterfactuals are essential to modern theories of causation and responsibility in philosophy and law [Beebee and Menzies 2019; Lewis 2013; Moore 2019; Starr 2021; Wachter et al. 2017], and are are already quite prominent in formal methods for accountability [Baier et al. 2021a,b,c; Chockler and Halpern 2004; Datta et al. 2015; Feigenbaum et al. 2020; Halpern and Pearl 2005a,b; Wachter et al. 2017]. Causation refers to the influence an input of some process has on its output, *e.g.*, in an MDP how a choice of action influences the resultant distribution on (some property of) the next state, or for a program how changes in the inputs influence the outputs. In the simplest possible case, an action a is an actual cause of an outcome, such as a harm h, if under every counterfactual a is both necessary and sufficient for the harm to occur, notated as $a \;\square\!\!\rightarrow\; h$ and $\neg a \;\square\!\!\rightarrow\; \neg h$. Here a and h are some domain-specific formal objects, while the modal notation $x \;\square\!\!\rightarrow\; y$, popularized by Lewis [Lewis 2013], means *if x had happened, then y would have happened.* The canonical unified logical and computational treatment of counterfactuals are the works of Pearl and Halpern [Halpern and Pearl 2005a,b], which provide a directed acyclic graph-based formalism able to inductively model causal effects in far more complicated dependency structures.

Responsibility is a higher-order property than causality. As raised in §2, counterfactuals of decision making are essential to interrogating *intention* and with it responsibility. Put simply, counterfactuals enable challenging and verifying proposed explanations for an agent's decisions. Importantly, counterfactual analysis is well-defined independent of any specific decisions, or the inferences about agent intention made on the basis of them. So although we often frame our discussion in terms of behaviors that match common understanding of human decision making, we stress our formal approach would generalize to future models of algorithmic decision making interested in very different attributes and behaviors than those we apply to humans.

Formalism. Working *ex post*, we formalize counterfactual algorithmic decision making by starting from a set of *factual* traces $T^f = \{\tau^f_1, \ldots, \tau^f_k\}$ encoding a history of A's harmful or otherwise relevant decisions. A *counterfactual* $\tau^{cf} = (\tau^f, \tau^{pp}, t^*)$ is a tuple of a factual trace τ^f, a *past possibility* trace τ^{pp}, and an integer $t^* \in \mathbb{N}$ that we call the *keyframe*. We write $\tau^{cf}.\mathsf{fst} = \tau^f$ and $\tau^{cf}.\mathsf{snd} = \tau^{pp}$. Intuitively, we want counterfactuals to represent the decisions that $\mathcal{A}$ *would have* made in revealing alternate circumstances. What makes a counterfactual 'revealing' is a deep and nuanced question, but the philosophy of action highlighs the importance of particular attributes for counterfactual scenarios to be meaningful. We enforce these tenets as predicates, in order to guarantee that our method works for counterfactuals possessing them.

(1) *Non-backtracking*: Counterfactuals should encode scenarios with a meaningful relationship to observed events, and should not require us to 'replay' the evolution of the world under significant changes to past history. Formally, both τ^f and τ^{pp} must be defined at t^*, and must agree up until it:

$$\mathsf{nbt}(\tau^{cf}) \equiv \forall t'.\, t^* < |\tau^f| \wedge t^* < |\tau^{pp}| \wedge \big(t' < t^* \rightarrow \tau^f(t') = \tau^{pp}(t')\big).$$

Every past possibility trace forms a non-backtracking counterfactual for $t^* = 1$, so usually choice of keyframe will come first from some *a priori* understanding the investigator has about the critical decision moments leading to a harm.

(2) *Scope of Decisions*: In order to clarify the purpose of an agent's actions, what an agent *might have done* is less important than what it *would have decided to try to do.* In complex systems the former is often contaminated by the decisions of other agents and the evolution of the environment, as agents rarely have complete control over outcomes. To clarify this distinction, we enforce a scope to the decision making of $\mathcal{A}$ by limiting past possibility traces to internal reasoning. No transition after t^* may update the valuations of E.

$$\mathsf{scope}(\tau,\, t^*) \equiv \forall t' \forall i \in [n_E].\, t^* < t' \rightarrow \tau(t'-1)(ve_i) = \tau(t')(ve_i).$$

This scope constraint can be interpreted as formalizing that we do not require or use access to an environmental model $\mathcal{E}[\mathcal{A}]$.

An *admissible* counterfactual is both non-backtracking and limited in scope:

$$\mathsf{admit}(\tau^{cf}) \equiv \mathsf{nbt}(\tau^{cf}) \wedge \mathsf{scope}(\tau^{pp},\, t^*).$$

In order to use automated reasoning to interrogate $\mathcal{A}$'s decision making history (in the form of T^f), we need to formalize the semantics of two different families of trace properties:

factuals: $\tau^f \overset{?}{\models} \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \beta(\hat{D})$
when faced with φ at time t, did $\mathcal{A}$ do β at time ℓ?

counterfactuals: $\tau^{cf} \overset{?}{\models} \varphi(\hat{I}) \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D})$
if faced with φ at time t^*, would $\mathcal{A}$ have done β at time ℓ?

We begin with factuals. In order to formulate a useful semantics for this predicate, we need a reasonable interpretation of the subsequence $\tau^f(t) \ldots \tau^f(\ell)$ that the property implicitly analyzes. Working after-the-fact justifies one: as a *window of agency*, during which either $\mathcal{A}$ made a decision or failed to do so as a harm played out. If the window of agency was still open, we could not be working *ex post.* We can then formulate a semantic definition in which $\varphi(\hat{I})$ specifies preconditions on the inputs to A, and $\beta(\hat{D})$ then specifies post-conditions on its decision variables, limited in scope and to

the window of agency.

$$\tau^f \models \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \beta(\hat{D}) \quad \text{if scope}(\tau^f, t) \text{ and } \tau^f(t)\Big|_I \models \varphi(\hat{I}) \text{ and } \tau^f(\ell)\Big|_D \models \beta(\hat{D}).$$

On the contrary, for counterfactuals it is not obvious that we can assume a known and finite window. As $\tau^{pp}(t^*)$ may have never been observed it could lead to A looping forever, and without an $\mathcal{E}[\mathcal{A}]$ we cannot know how long the window would last. However, as counterfactuals are objects of our own creation, we will assume that the investigator can conjecture a reasonable window $[t^*, \ell]$ within which the decision of $\mathcal{A}$ must be made in order to be timely, with responsibility attaching to the agent if it is unable to make a decision within it. This assumption guarantees termination.

With this philosophically distinct but mathematically equivalent assumption, we are able to define the semantics of the counterfactual operator as

$$\tau^{cf} \models \varphi(\hat{I}) \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}) \quad \text{if admit}(\tau^{cf}) \text{ and } \tau^f(t^*)\Big|_I \not\models \varphi(\hat{I}) \text{ and } \tau^{pp}(t^*)\Big|_I \models \varphi(\hat{I}) \text{ and } \tau^{pp}(\ell)\Big|_D \models \beta(\hat{D}).$$

In practice, our use of symbolic execution will abstract away these details by framing the scope and the window of agency so that $\text{admit}(\tau^{cf})$ is true by construction. We discuss this formally in §5.1.

Lastly, we will oftentimes discuss *families of counterfactuals*, which are sets of counterfactuals which share a factual trace

$$T^{cf} = \{\tau_1^{cf}, \ldots, \tau_{k'}^{cf} \mid \forall i, j \in [k']. \tau_i^{cf}.\text{fst} = \tau_j^{cf}.\text{fst}\}.$$

Families of counterfactuals can naturally be defined implicitly by a tuple $\text{ctx} = (\tau^f, t^*, \varphi)$ as

$$T^{cf}_{\text{ctx}} = \{\tau^{cf} \mid \text{admit}(\tau^{cf}) \text{ and } \tau^{cf}.\text{fst}(t^*) \not\models \varphi(\hat{I}) \text{ and } \tau^{cf}.\text{snd}(t^*) \models \varphi(\hat{I})\}.$$

Choice of context ctx will be our usual way of delineating families, especially as φ then provides a descriptive representation.

4 Formal Reasoning for Accountability

Given a program A and log of factual executions T^f, soid aims to provide an interactive, adaptive procedure for the investigator to refine a set Facts of trace properties capturing how A behaves in T^f and related counterfactuals, just as in a legal finding of fact. We call this *counterfactual-guided logic exploration*. Our end goal is for soid to is to enable continuous refinement of a *formal representation* of $\mathcal{A}$'s decision making:

$$\text{Facts} = \{\ldots, \big(\tau_i^f, \varphi_i(I) \rightarrow_{\mathcal{A},t,\ell} \beta_i(V)\big), \ldots\} \cup \{\ldots, \big(\tau_j^{cf}, \varphi_j(I) \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta_j(V)\big), \ldots\}.$$

Each fact in Facts is composed of a (counter)factual trace and a property that holds over it, as proven by an SMT solver. Since we do not assume access to some overarching property $P(A)$ that we aim to prove, Facts is the ultimate product of the counterfactual-guided logic exploration. The human investigator is trusted to take Facts and use it to assess $\mathcal{A}$'s responsibility for a harm.

Our method relies on an *oracle* interface, $O_A(\cdot)$, into the decision logic of A. We specify factual queries as $q = (\varphi, \beta)$, pairing an input constraint φ and a behavior β. Such a query asks whether the factual program execution starting from the program state encoded by φ results in the agent behavior encoded by β, or more formally...

for τ^f s.t. $\tau^f(t) \models \varphi(\hat{I})$ uniquely, does $\tau^f \models \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \beta(\hat{D})$?

We specify counterfactual queries as $q = (\mathbf{1}_\exists, \varphi, \beta)$, composed of a 'might'/'would' (existential/universal) indicator bit $\mathbf{1}_\exists$, input constraint φ, and behavior β. Each such query asks whether there exists a program execution (a might counterfactual) starting from a program state encoded by φ that results in the agent behavior encoded by β, more formally...

if $\mathbf{1}_\exists$, for $\text{ctx} = (\tau^f, t^*, \varphi)$ does there exist $\tau^{cf} \in T^{cf}_{\text{ctx}}$ such that $\tau^{cf} \models \varphi(\hat{I}) \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D})$;

or similarly, but now whether for all executions starting from the program states encoded by φ (a would counterfactual)...

if $\neg\mathbf{1}_\exists$, for $\text{ctx} = (\tau^f, t^*, \varphi)$, whether for all $\tau^{cf} \in T^{cf}_{\text{ctx}}$ it follows that $\tau^{cf} \models \varphi(\hat{I}) \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D})$?

This quite minimal information is sufficient for the oracle to resolve the information needed to improve Facts. Note that as T^{cf}_{ctx} excludes the factual trace as a valid continuance from t^*, it is not possible for a counterfactual query to resolve (positively or negatively) on the basis of the factual execution – only counterfactuals are considered.

An Example. Consider an investigator trying to understand the facts under which the car in Figure 1 did, would, or might enter the intersection. If φ of Equation 1 represents the critical moment at which the car moved into the intersection, then the investigator could query

$$q_1 = (\varphi, \texttt{move} = 1)$$

where `move` is a decision variable. If, for example, $r_1 = (1, \_\_)$, then Algorithm 1 will set

$$\text{Facts} = \{(\tau^f, \varphi \rightarrow_{\mathcal{A},t,\ell} \texttt{move} = 1)\}$$

to capture the now confirmed fact that the car chose to move into the intersection (rather than say, had a brake failure). Note that to do so the investigator needs only to know the input constraints φ and the specific decision variable `move`. All other aspects of the self-driving car's decision logic is hidden by the oracle interface, and the output is clear and interpretable answer to exactly the question posed. Adaptively, the investigator might then decide to skip Equation 2, and instead move on to querying using φ'' from Equation 3, *e.g.*,

$$q_2 = (\mathbf{1}_\exists, \varphi'', \texttt{move} = 0)$$

to ask whether under the family of counterfactuals T^{cf}_{ctx} defined by φ'' there exists a circumstance where the car would not have entered the intersection. If then, for example, $r_2 = (1, \mathcal{M})$, where the model $\mathcal{M}$ encodes a concrete counterfactual scenario, the investigator can update

$$\text{Facts} \leftarrow \text{Facts} \cup \{(\mathcal{M}, \varphi'' \;\square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \texttt{move} = 0)\}$$

and continue on from there.

Algorithm 1 Counterfactual-Guided Logic Exploration

$T^f \leftarrow \{\tau^f_1, \ldots, \tau^f_k\}$, Facts $\leftarrow \{\}$
while *not done?* **do**
  if *factual?* **then**
    $(\tau^f, t), \varphi \leftarrow$ *start?*(T^f, I)
    $\beta \leftarrow$ *behavior?*(V)
    $r_i = (b, \_) \leftarrow O_A(q_i = (\varphi, \beta))$
    if $b = 0$
      then Facts $\leftarrow$ Facts $\cup \{(\tau^f(t),\ \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \neg\beta(\hat{D}))\}$
      else Facts $\leftarrow$ Facts $\cup \{(\tau^f(t),\ \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \beta(\hat{D}))\}$
  else
    ctx $= (\tau^f, t^*, \varphi)$, $F \leftarrow cf(T^f, I)$
    $\beta, \mathbf{1}_\exists \leftarrow$ *behavior?*(V)
    $r_i = (b, \mathcal{M}) \leftarrow O_A(q_i = (\mathbf{1}_\exists, \varphi, \beta, F))$
    if $\mathbf{1}_\exists = 0$ **then**
      if $b = 0$
        then Facts $\leftarrow$ Facts $\cup \{(\mathcal{M},\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \neg\beta(\hat{D}))\}$
        else Facts $\leftarrow$ Facts $\cup \{(\tau^{cf}_j(t^*),\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}))\}_{\tau^{cf}_j \in T^{cf}_{\text{ctx}}}$
    else
      if $b = 0$
        then Facts $\leftarrow$ Facts $\cup \{(\tau^{cf}_j(t^*),\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \neg\beta(\hat{D}))\}_{\tau^{cf}_j \in T^{cf}_{\text{ctx}}}$
        else Facts $\leftarrow$ Facts $\cup \{(\mathcal{M},\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}))\}$

The Method. We define the counterfactual-guided logic exploration loop and oracle interface that together underlie soid in Algorithms 1 and 2, where calls in *italics?* indicate manual interventions that must be made by the investigator. The investigatory procedure starts from the set of factual traces $T^f = \{\tau^f_1, \ldots, \tau^f_k\}$ observed from A's executions. At each iteration of the loop, the investigator adaptively formulates and poses the next question in a sequence Query $= \langle q_1, \ldots, q_i, \ldots\rangle$. The responses Resp $= \langle r_1, \ldots, r_i, \ldots\rangle$ are then used to build up the set Facts of trace properties regarding A's decision making under both the T^f and the set of counterfactual scenarios, $T^{cf} = \{\tau^{cf}_1, \ldots, \tau^{cf}_{k'}\}$, defined within the q_i by the investigator. Each entry in Facts is rigorously proven by the verification oracle $O_A(\cdot)$, with access to the logical representation Π of $\mathcal{A}$ as expressed by A. We leave to the investigator the decision to terminate the investigatory loop, as well as any final judgement as to the agent's culpability. In §5 we explain the encodings Φ used within Algorithm 2 in detail, and in our technical report [Judson et al. 2023] prove that they correctly implement the semantics of $\rightarrow_{\mathcal{A},t,\ell}$ and $\boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell}$ as defined in §3.3.

Design Goals. We briefly highly how soid meets some critical design goals to support principled analysis for legal accountability.

(1) The oracle design pushes the technical details of how $\mathcal{A}$ works 'across the veil', so that an investigator needs to know no more than the meaning of the input/output API exposed by A (over the variables in I and some subset of D, respectively) in order to construct a query q_i and interpret the response $r_i \leftarrow O_A(q_i)$. To this end, we designed the oracle query to place as minimal a possible burden on the investigator.

(2) The method emphasizes *adaptive* construction of Facts, so that the investigator may shape the ith query not just by considering the questions $\langle q_1, \ldots, q_{i-1}\rangle$ asked, but also using the responses $\langle r_1, \ldots, r_{i-1}\rangle$ already received. We aim to put the agent on the stand, not just send it a questionnaire. Crucial to this goal is to return concrete traces from counterfactual queries, so that their corresponding facts can help guide the construction of the next. Using the ability of SMT solvers to return models for satisfiable formulas, when $\mathbf{1}_\exists = 1$ and there exists $\tau^{cf} \in T^{cf}_{\text{ctx}}$ such that $\tau^{cf} \models \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D})$, we are able to explicitly inform the investigator of the fact $(\tau^{cf},\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}))$. Conversely, when $\mathbf{1}_\exists = 0$ and $\varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D})$ is not true for all $\tau^{cf} \in T^{cf}_{\text{ctx}}$, we also can explicitly return the fact $(\overline{\tau^{cf}},\ \varphi(\hat{I}) \boxdot\!\!\rightarrow_{\mathcal{A},t^*,\ell} \neg\beta(\hat{D}))$ for some counterexample $\overline{\tau^{cf}} \in T^{cf}_{\text{ctx}}$ encoded by the output model $\mathcal{M}$.

(3) The method is *interpretable*. When an investigator poses a question, soid pushes everything 'smart' the method does across the oracle interface to the verification, so that even non-technical users can understand the relationship between query and response. In a sense, our method benefits from a simple and straightforward design, so that its process is direct and interpretable to the investigators using it. As with a human on the stand, we just want the answer to the question that was asked, no more and no less. As this design goal describes what *not to do* rather than what *to do*, we informally meet it by not introducing unnecessary automation.

In general, we balance automation against interpretability, in order to minimize the burden on the investigator: we want them to pick

Algorithm 2 Oracle

Require: $q_i = (\varphi, \beta)$ or $q_i = (\mathbf{1}_\exists, \varphi, \beta, F)$
1: $(ct, \{\pi_{ab}\}) \leftarrow \mathsf{SymExec}(A, \varphi(\hat{I}))$ ▷ ct is the branch count
2: $\Pi \leftarrow \bigvee_{a\in[n], b\in[ct]} \pi_{ab}$
3: **if** $|q| = 2$ **then**
4: $\quad \Phi \leftarrow \neg\big(\big(\varphi(\hat{I}) \wedge \Pi(\hat{V})\big) \rightarrow \beta(\hat{D})\big)$
5: $\quad (b, \mathcal{M}) \leftarrow \mathsf{SMT.isValid?}(\Phi)$
6: **else**
7: $\quad$ **if** $q.\mathsf{fst} = 0$ **then**
8: $\quad\quad \Phi \leftarrow \neg\big(\big(\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})\big) \rightarrow \beta(\hat{D})\big)$
9: $\quad\quad (b, \mathcal{M}) \leftarrow \mathsf{SMT.isValid?}(\Phi)$
10: $\quad$ **else**
11: $\quad\quad \Phi \leftarrow \big(\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})\big) \wedge$
12: $\quad\quad\quad\quad \big(\big(\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})\big) \rightarrow \beta(\hat{D})\big)$
13: $\quad\quad (b, \mathcal{M}) \leftarrow \mathsf{SMT.isSat?}(\Phi)$
14: **return** $r_i = (b, \mathcal{M})$

a critical moment and (family of) counterfactual scenarios, define a behavior as a post-condition, and get push-button execution.

5 Representations and Queries

We discuss how we represent (counter)factual scenarios and queries logically so that soid can resolve them using an SMT solver. For an extended discussion of how see our full technical report [Judson et al. 2023].

We specify a factual query as a tuple (φ, β), the former logical formula specifying the inputs to the program at a critical decision moment (as in Equation 1), the latter encoding a description of the possible agent decision being investigated. Implicitly, φ defines a factual scenario (τ^f, t, φ), where φ encodes the program state at that critical moment $\tau^f(t)$. Counterfactual queries are encoded similarly, but with the additional of the existential indicator bit $\mathbf{1}_\exists$. They are also able to encode many different possible program executions, captured by the notion of a family of counterfactuals T^{cf}_{ctx}. The last necessary statement required to invoke an SMT solver is $\Pi(\hat{V})$, the decision logic of A constrained to the scenario(s) implied by φ. We generate $\Pi(\hat{V})$ dynamically given a (counter)factual query using symbolic execution.

5.1 Representing Agents and Scenarios

We represent (counter)factual scenarios by formulas on the variables in $\hat{I}$ so that soid can resolve them using an SMT solver.

Factuals. Factual scenarios are naturally defined such that every input variable is constrained by an equality, together encoding some factual state $\tau^f(t)$.

Definition 5.1. *A **factual scenario** is a tuple (τ^f, t, φ) s.t.*
i) $\tau^f(t)\Big|_I \models \varphi(\hat{I})$; and
ii) for all $\sigma|_I \neq \tau^f(t)\Big|_I$, $\sigma|_I \not\models \varphi(\hat{I})$.

In practice, a factual query is specified by a φ that has a unique satisfying model over I. Evaluating a factual scenario is functionally equivalent to a concrete execution, since τ^f is the only possible program trace. We tie factual analysis into our framework for completeness, and because unlike traditional 'opaque' assertion-based testing soid supports writing complex behavioral conditions on all of V, including both internal and output variables. Additionally, our factual representations also naturally generate circuits for (zero-knowledge) proofs-of-compliance, another promising tool for algorithmic accountability [Kroll et al. 2017; Ozdemir et al. 2022].

Counterfactuals. A counterfactual is a formula which removes the original factual τ^f as a valid model, $\varphi(\hat{I}) \equiv \varphi(\hat{E}) \wedge \neg F(\hat{I})$.

Definition 5.2. *1. A **counterfactual scenario** is a tuple $(\tau^{cf} = (\tau^f, \tau^{pp}, t^*), \varphi, F)$ such that*
i) $\tau^{pp}(t^*)|_I \models \varphi(\hat{I}) \wedge \neg F(\hat{I})$;
ii) $\tau^f(t^*)\Big|_I \models F(\hat{I})$; and
*2. A **family of counterfactual scenarios** is a tuple $(T^{cf}_{ctx}, \varphi, F)$ where the set T^{cf}_{ctx} contains every $\tau^{cf} = (\tau^f, \tau^{pp}, t^*)$ such that $\tau^{pp}|_I \models \varphi(\hat{I}) \wedge \neg F(\hat{I})$ and (τ^{cf}, φ, F) is a counterfactual scenario.*

In practice, a counterfactual query is specified by a $(\mathbf{1}_\exists, \varphi, F)$ tuple where $\tau^f(t^*)$ is excluded as a model by the negation of the formula $F(\hat{I})$ tightly encoding it.

Behaviors. A behavior is just an arbitrary formula over $\hat{D}$.

Definition 5.3. *A **behavior** is a formula $\beta(\hat{D})$.*

Decision Logic. We leave the more involved definition of the **decision logic** $\Pi(\hat{V})$ to our technical report [Judson et al. 2023]. Roughly, it is a formula representing the possible executions of A under the preconditions specified in the (family of counter)factual scenarios, and is generated by symbolic execution of A.

5.2 Resolving (Counter)factual Queries

Given these representations, we can encode the semantics of our factual $(\rightarrow_{\mathcal{A},t,\ell})$ and counterfactual $(\square\!\!\rightarrow_{\mathcal{A},t^*,\ell})$ operators as SMT queries in the logic of `QF_FPBV` [Barrett et al. 2021]. To conclude the following theorem we assume correctness of symbolic execution, *i.e.*, that $\Pi(\hat{V})$ exactly represents the possible executions of A under $\varphi(\hat{I})$ up to some step $\ell \leq \ell_{\mathsf{max}}$. For proofs of the results in this section see our technical report [Judson et al. 2023].

Theorem 5.4. *Let $q_i = (\varphi, \beta)$ be a factual query, and (τ^f, t, φ) be a corresponding factual scenario. Then*

$$\Phi \equiv \big(\varphi(\hat{I}) \wedge \Pi(\hat{V})\big) \rightarrow \beta(\hat{D})$$

is valid iff $\tau^f \models \varphi(\hat{I}) \rightarrow_{\mathcal{A},t,\ell} \beta(\hat{D})$.

A similar result holds for both types of counterfactual query.

Theorem 5.5. *Let $q_i = (\mathbf{1}_\exists, \varphi, \beta)$ be a counterfactual query, and $(T^{cf}_{ctx}, \varphi, F)$ be a corresponding family of counterfactual scenarios. Then*
i) for $\neg\mathbf{1}_\exists$,

$$\Phi \equiv \big(\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})\big) \rightarrow \beta(\hat{D})$$

is valid iff

$$\forall \tau^{cf} \in T^{cf}_{ctx}.\ \tau^{cf} \models \varphi(\hat{I}) \wedge \neg F(\hat{I}) \ \square\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}).$$

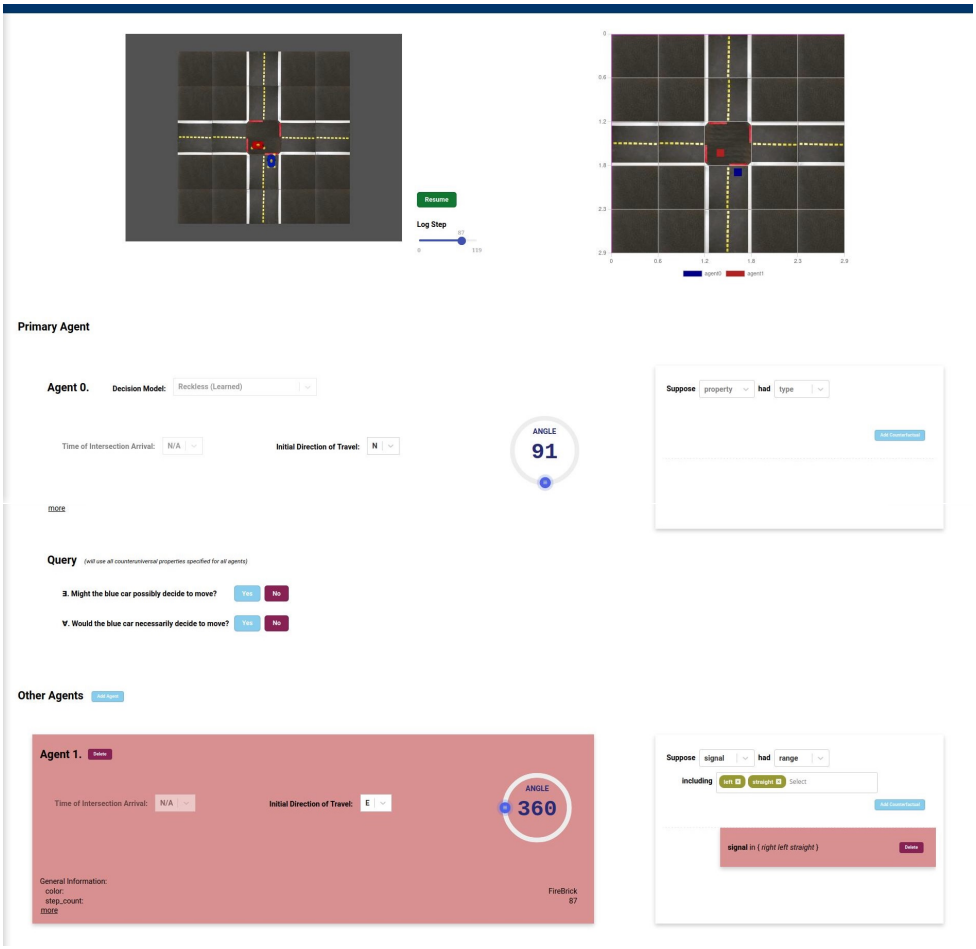


Figure 2: Still of the soid GUI (with a small section cut out for brevity). At top left is the critical moment from the program logs as chosen by the investigator. At bottom right are the counterfactual conditions the investigator has specified.

ii) for $\mathbf{1}_\exists$,

$$\Phi \equiv \big(\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})\big) \wedge \big((\varphi(\hat{I}) \wedge \neg F(\hat{I}) \wedge \Pi(\hat{V})) \rightarrow \beta(\hat{D})\big)$$

is satisfiable iff

$$\exists \tau^{cf} \in T^{cf}_{ctx}.\ \tau^{cf} \models \varphi(\hat{I}) \wedge \neg F(\hat{I}) \ \Box\!\!\rightarrow_{\mathcal{A},t^*,\ell} \beta(\hat{D}).$$

6 The soid Tool: Architecture and Case Studies

We implemented the counterfactual-guided logic exploration loop in our tool soid, for **S**MT-based **O**racle for **I**nvestigating **D**ecisions. The soid tool is implemented in Python, and invokes the Z3 SMT solver [Moura and Bjørner 2008] for resolving queries. To begin, and outside of the scope of soid, the investigator uses their knowledge of the harm under investigation to extract the factual trace τ^f from the logging infrastructure of A. Note that our tool assumes that both the τ^f and A used in the analysis correspond to the real-world execution. Accountable logging [Yoon and Shao 2019] and verifiable computation [Parno et al. 2013] can bolster confidence in these assumptions, and further that the program execution pathways being analyzed by soid are those applicable in deployment and are not being manipulated by a 'defeat device' [Contag et al. 2017]. At present soid also assumes deterministic programs, though symbolic execution of randomized programs is an active area of formal methods research with developing tooling that could in the future be used to extend our method [Susag et al. 2022]. After extracting the trace the investigator specifies the (counter)factual query $\varphi(I)$ and behavior $\beta(\hat{D})$ using a Python library interface. Upon invocation, soid symbolically executes A to generate $\Pi(\hat{V})$. After the symbolic execution completes, soid formulates Φ as per §5 and invokes Z3 to resolve the query. It then outputs the finding, as well as any model $\mathcal{M}$ in the event one exists due to a failed verification or successful counterfactual generation.

6.1 Three Cars on the Stand: A Case Study

In this section, we evaluate soid on the crash example from §2 (and Figure 1). We pose and resolve the queries from the example:

model	output	timings (avg. $n = 10$) symbolic (s)	solving (s)	total (s)	paths
$\rightarrow_{\mathcal{A},t,\ell}$	φ_{fact}, *moved?*				
standard	✔	3.575	4.290e-03	4.162	1
impatient	✔	3.607	4.317e-03	4.193	1
pathological	✔	3.626	4.249e-03	4.212	1
$\square\!\!\rightarrow_{\mathcal{A},t^*,\ell}$	$\varphi^* \equiv \varphi_{fact}[(\texttt{agent1\_signal\_choice} = 2) \mapsto (\texttt{agent1\_signal\_choice} \in \{0, 1, 2\})]$, *ever not move?*				
standard	✔	4.015	2.428	7.849	3
impatient	✘	3.919	2.334	7.673	3
pathological	✘	3.966	2.352	7.718	3
$\square\!\!\rightarrow_{\mathcal{A},t^*,\ell}$	$\varphi^*[(\texttt{agent1\_pos\_x} = 1.376) \mapsto (1.0 \leq \texttt{agent1\_pos\_x} \leq 1.5)]$, *ever not move?*				
standard	✔	133.0	54.40	195.2	19
impatient	✘	126.0	4.648	138.5	19
pathological	✔	254.2	17.47	279.5	19

model	output	timings (avg. $n = 10$) symbolic (s)	solving (s)	total (s)	paths
$\rightarrow_{\mathcal{A},t,\ell}$	φ_{fact}, *low risk?*				
dt	✔	0.746	4.896e-03	0.812	1
$\square\!\!\rightarrow_{\mathcal{A},t^*,\ell}$	$\varphi^* \equiv \varphi_{fact}[(\texttt{weight} = 249.973) \mapsto \top]$, *ever high risk?*				
dt	✔	2.277	1.655	4.009	2

Table 1: Experimental results for our (top) car crash and (bottom) decision tree misclassification case studies.

at t_1^*

$\square\!\!\rightarrow$ Could a different turn signal have led $\mathcal{A}$ to remain stationary?

$\square\!\!\rightarrow$ If $\mathcal{A}$ had arrived before the other car, and that other car was not signaling a turn, would $\mathcal{A}$ have waited? (*e.g.*, to 'bait' the other car into passing in front of it?)

in a simulated driving environment, and show that soid is able to produce Facts that distinguish between three different machine-learned self-driving car agents.

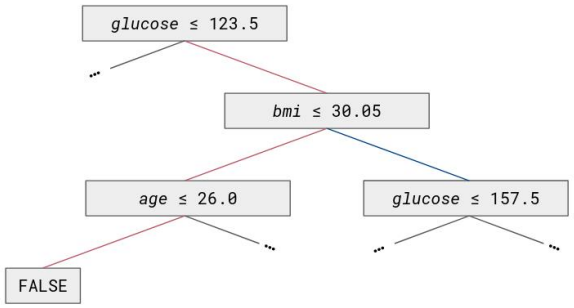
For our environment we employ Gym-Duckietown [Chevalier-Boisvert et al. 2018] with a simple intersection layout. A rendering of our example crash in our environment is given in Figure 1. For our three agents, we used the same general C codebase, but used reinforcement learning – specifically Q-learning [Watkins and Dayan 1992] – to train three different versions of the decision model it invokes, each based on a different reward profile. Informally we deemed these reward profiles 'standard', 'impatient' and 'pathological'. The 'standard' profile is heavily penalized for crashing, but also rewarded for speed and not punished for moving without the right of way, so long as it is 'safe'. The 'impatient' profile is only rewarded for speed. The 'pathological' profile is rewarded significantly for crashes, and minimally for speed to promote movement over nothing. The simulation environment is completely invisible to soid, which only analyzes program executions on the basis of its code and logs.

On top of Gym-Duckietown we designed and implemented a web GUI to enable non-expert interaction with soid. GUIs that automatically generate representations of the driving environment are already deployed into semi-autonomous vehicles, such as those produced by Tesla. While simulating the environment, a drag-and-drop and button interface allows the user to manipulate the environment. by, *e.g.*, introducing new cars, manipulating a car's position or angle, or changing a car's destination or which car possesses the right of way. After a factual trace plays out, a slider allows the investigator to select a step of the execution, before a drop-down and button interface allows specifying a counterfactual family and behavior (whether or not the car moved). We provide still images of the GUI's interfaces in Figure 2.

Results. We provide a selection of the results of our benchmarks, summarized in Table 1. We refer to our technical report [Judson et al. 2023] full the full set of benchmarks. All statistics were gathered on an Intel Xeon CPU E5-2650 v3 @ 2.30GHz workstation with 64 GB of RAM. Each heading in Table 1 specifies a set of constraints $\varphi(\hat{I})$, and implicitly a behavior $\beta(\hat{D})$. The rows list the trained model invoked within A, the output of the evaluations, average timings, and the total number of feasible paths. Note that the **symbolic** and **solving** timings do not exactly sum to the **total** timing, due to some overhead. In our full benchmarks we find most of our queries resolved within $< 20s$, providing effective usability, though as Table 1 shows the inclusion of a floating-point range query to notably increase the cost of solving, with a ~6x increase in the number of feasible program paths and a ~20-30x increase in the time required.

6.2 Health Risk Decision Tree Misclassification

To demonstrate that soid is more general in application than to just cyberphysical systems, we also consider a second motivating




```c
int traverse(Node *N, double *fv) {
  if (N->class >= 0) return N->class;
  return (fv[N->tidx] <= N->test)
    ? traverse(N->tchild, fv)
    : traverse(N->fchild, fv);
}

int classify(Node *root, double *data) {
  double bmi = data[6] / pow(data[5], 2);
  double fv[8] = {
    data[0], data[1], data[2], data[3],
    data[4],      bmi, data[7], data[8]
  };

  return traverse(root, fv);
}
```

Figure 3: Our decision tree example. At top, the relevant decision subtree for a misclassification based on health data, with the incorrect path taken in red – and the correct branch missed in blue – as the unit conversion bug leads to a significantly smaller BMI input than is correct. At bottom, the (otherwise correct) decision tree inference logic in C.

example of incorrect statistical inference. We train a decision tree to infer the health risk status of individuals using the Pima Indians dataset, a classic example in counterfactuals due to [Wachter et al. 2017]. Notably, we consider a program A with an implicit unit conversion bug: A computes the BMI input to the decision tree using the height and weight parameters from its input. However, it is written to expect metric inputs in kg and m, while the inputs are instead provided in the imperial in and lb. This is a flaw of the software system in general. Both the decision tree and program themselves are correct, but end-to-end the system misclassifies many inputs, as for the same quantities $(kg/m^2) \gg (lb/in^2)$.

Unlike statistical counterfactual methods like those of [Mothilal et al. 2020; Wachter et al. 2017] which only analyze the (correct) decision model, the end-to-end nature of soid allows it to analyze everything, including the conversion bug. Figure 3 displays the inference code and incorrect decision due to the conversion error.

We then ran a small case study on this decision tree health risk misclassification example. The results of our benchmarks are summarized in Table 1, and were gathered on the same Intel Xeon CPU E5-2650 v3 @ 2.30GHz workstation with 64 GB of RAM. In additional to a simple factual verification query as a baseline, we posed a single counterfactual query:

at t^*

> $\square\!\!\rightarrow$ Does there exist a weight for which the instance is classified as high risk?

The results show soid is able to efficiently resolve the counterfactual in the positive.

Acknowledgments

The authors thank Gideon Yaffe for many helpful conversations, Man-Ki Yoon for his assistance in implementing an earlier simulated driving environment, and Cristian Cadar and Daniel Liew for their guidance on successfully using Klee-Float for symbolic execution of our experiments. This work was supported by the Office of Naval Research (ONR) of the United States Department of Defense through a National Defense Science and Engineering Graduate (NDSEG) Fellowship, by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark, by EPSRC grant no EP/R014604/1, and by NSF awards CCF-2131476, CCF-2106845, and CCF-2318974. The authors would also like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programme Verified Software where work on this paper was undertaken.

References

[n. d.]. Memorandum Opinion, Thaler v. Shira Perlmutter *et al.* (2023) (No. 1:22-cv-01564-BAH) https://storage.courtlistener.com/recap/gov.uscourts.dcd.243956/gov.uscourts.dcd.243956.24.0_2.pdf.

Rediet Abebe, Moritz Hardt, Angela Jin, John Miller, Ludwig Schmidt, and Rebecca Wexler. 2022. Adversarial Scrutiny of Evidentiary Statistical Software. In *ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*. 1733–1746.

Amina Adadi and Mohammed Berrada. 2018. Peeking Inside the Black-Box: a Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '18)*, Vol. 32.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. May 23rd, 2016. Machine Bias. *ProPublica* (May 23rd, 2016). https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *Information Fusion* 58 (2020), 82–115.

Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Rupak Majumdar, Jakob Piribauer, and Robin Ziemek. 2021a. From Verification to Causality-based Explications. *arXiv preprint arXiv:2105.09533* (2021).

Christel Baier, Florian Funke, and Rupak Majumdar. 2021b. A Game-Theoretic Account of Responsibility Allocation. *arXiv preprint arXiv:2105.09129* (2021).

Christel Baier, Florian Funke, and Rupak Majumdar. 2021c. Responsibility Attribution in Parameterized Markovian Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11734–11743.

Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. 2018. A Survey of Symbolic Execution Techniques. *ACM Comput. Surv.* 51, 3, Article 50 (2018).

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2021. The SMT-LIB Standard: Version 2.6.

Helen Beebee and Peter Menzies. 2019. Counterfactual Theories of Causation. In *Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.). Stanford University.

Michael Bratman. 1987. Intention, Plans, and Practical Reason.

Michael E. Bratman, David J. Israel, and Martha E. Pollack. 1988. Plans and Resource-Bounded Practical Reasoning. *Computational intelligence* 4, 3 (1988), 349–355.

Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*. 209–224.

Filip Cano Córdoba, Samuel Judson, Timos Antonopoulos, Katrine Bjørner, Nicholas Shoemaker, Scott J. Shapiro, Ruzica Piskac, and Bettina Könighofer. 2023. Analyzing Intentional Behavior in Autonomous Agents under Uncertainty. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. 372–381.

Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, and Liam Paull. 2018. Duckietown Environments for OpenAI Gym. https://github.com/duckietown/gym-duckietown.

Hana Chockler and Joseph Y. Halpern. 2004. Responsibility and Blame: A Structural-Model Approach. *Journal of Artificial Intelligence Research)* 22 (2004), 93–115.

Maria Christakis, Hasan Ferit Eniser, Holger Hermanns, Jörg Hoffmann, Yugesh Kothari, Jianlin Li, Jorge A Navas, and Valentin Wüstholz. 2021. Automated Safety Verification of Programs Invoking Neural Networks. In *International Conference on Computer Aided Verification (CAV '21)*. Springer, 201–224.

Philip R. Cohen and Hector J. Levesque. 1990. Intention is Choice with Commitment. *Artificial Intelligence* 42, 2-3 (1990), 213–261.

David Cole. 2020. The Chinese Room Argument. In *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.). Stanford University.

Moritz Contag, Guo Li, Andre Pawlowski, Felix Domke, Kirill Levchenko, Thorsten Holz, and Stefan Savage. 2017. How They Did It: An Analysis of Emission Defeat Devices in Modern Automobiles. In *IEEE Symposium on Security and Privacy (Oakland '17)*. IEEE, 231–250.

Jeffrey Dastin. 2018. Amazon scraps secret AI recruiting tool that showed bias against women. *Reuters* (2018). https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G.

Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. 2015. Program Actions as Actual Causes: A Building Block for Accountability. In *2015 IEEE 28th Computer Security Foundations Symposium (CSF '15)*. IEEE, 261–275.

Tim W. Dornis. 2020. Artificial Creativity: Emergent Works and the Void in Current Copyright Doctrine. *Yale JL & Tech.* 22 (2020), 1.

Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2019. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-based Systems. In *Intentional Conference on Computer Aided Verification (CAV '19)*. Springer, 432–442.

James Edwards. 2021. Theories of Criminal Law. In *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.). Stanford University.

Joan Feigenbaum, Aaron D. Jaggard, and Rebecca N. Wright. 2011. Towards a Formal Model of Accountability. In *Proceedings of the 2011 New Security Paradigms Workshop*. 45–56.

Joan Feigenbaum, Aaron D. Jaggard, and Rebecca N. Wright. 2020. Accountability in Computing: Concepts and Mechanisms. *Foundations and Trends® in Privacy and Security* 2, 4 (2020), 247–399.

Javier Garcıa and Fernando Fernández. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI^2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (S&P '18)*. 3–18.

Bishwamittra Ghosh and Kuldeep S. Meel. 2019. IMLI: An Incremental Framework for MaxSAT-based Learning of Interpretable Classification Rules. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES '19)*. 203–210.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–42.

Gabriel Hallevy. 2013. *When Robots Kill: Artificial Intelligence Under Criminal Law*. UPNE.

Joseph Y. Halpern and Judea Pearl. 2005a. Causes and Explanations: A Structural-Model Approach. Part I: Causes. *The British Journal for the Philosophy of Science* 56, 4 (2005), 843–887.

Joseph Y. Halpern and Judea Pearl. 2005b. Causes and Explanations: A Structural-Model Approach. Part II: Explanations. *The British Journal for the Philosophy of Science* 56, 4 (2005), 889–911.

Samuel Judson, Matthew Elacqua, Filip Cano Córdoba, Timos Antonopoulos, Bettina Könighofer, Scott J Shapiro, and Ruzica Piskac. 2023. 'Put the Car on the Stand': SMT-based Oracles for Investigating Decisions. *arXiv preprint arXiv:2305.05731* (2023).

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *International Conference on Computer Aided Verification (CAV '19)*. 443–452.

Joshua A. Kroll, Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu. 2017. Accountable Algorithms. *University of Pennsylvania Law Review* 165, 3 (2017), 633–705.

Robert Künnemann, Ilkan Esiyok, and Michael Backes. 2019. Automated Verification of Accountability in Security Protocols. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF '19)*. IEEE, 397–39716.

Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*. 526–535.

Legal Information Institute. 2023. Respondeat Superior. https://www.law.cornell.edu/wex/respondeat_superior.

Edward H Levi. 1947. An Introduction to Legal Reasoning. *U. Chi. L. Rev.* 15 (1947), 501.

David Lewis. 2013. *Counterfactuals*. John Wiley & Sons. Originally published in 1973.

Michael Moore. 2019. Causation in the Law. In *Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.). Stanford University.

Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining Machine Learning Classifiers Through Diverse Counterfactual Explanations. In *ACM Conference on Fairness, Accountability, and Transparency (FAT* '20)*. 607–617.

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*. 337–340.

Vincent C. Müller. 2023. Ethics of Artificial Intelligence and Robotics. In *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta and Uri Nodelman (Eds.). Stanford University.

Alex Ozdemir, Fraser Brown, and Riad S Wahby. 2022. CirC: Compiler Infrastructure for Proof Systems, Software Verification, and more. In *IEEE Symposium on Security and Privacy (Oakland '22)*. 2248–2266.

Paulo Henrique Padovan, Clarice Marinho Martins, and Chris Reed. 2023. Black is the New Orange: How to Determine AI Liability. *Artificial Intelligence and Law* 31, 1 (2023), 133–167.

Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 238–252.

Anand S. Rao and Michael P. Georgeff. 1991. Modeling Rational Agents Within a BDI-Architecture. *KR* 91 (1991), 473–484.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

Lauren Smiley. 2022. 'I'm the Operator': The Aftermath of a Self-Driving Tragedy. *Wired Magazine* (2022). https://www.wired.com/story/uber-self-driving-car-fatal-crash/.

William Starr. 2021. Counterfactuals. In *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.). Stanford University.

Guolong Su, Dennis Wei, Kush R. Varshney, and Dmitry M. Malioutov. 2015. Interpretable Two-Level Boolean Rule Learning for Classification. *arXiv preprint arXiv:1511.07361* (2015).

Zachary Susag, Sumit Lahiri, Justin Hsu, and Subhajit Roy. 2022. Symbolic Execution for Randomized Programs. *Proceedings of the ACM on Programming Languages* 6, OOPSLA (2022), 1583–1612.

Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law & Technology* 31 (2017), 841.

Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8 (1992), 279–292.

Man-Ki Yoon and Zhong Shao. 2019. ADLP: Accountable Data Logging Protocol for Publish-Subscribe Communication Systems. In *International Conference on Distributed Computing Systems (ICDCS '19)*. IEEE, 1149–1160.