Explaining Actual Causation via Reasoning about Actions and Change

Emily LeBlanc[1](✉)[0000−0003−1447−412X], Marcello Balduccini[2][0000−0001−5445−3054], and Joost Vennekens[3][0000−0002−0791−0176]

[1] Drexel University, Philadelphia, PA 19104, USA
leblanc@drexel.edu
[2] Saint Joseph's University, Philadelphia, PA 19131, USA
marcello.balduccini@sju.edu
[3] KU Leuven, 2860 Sint-Katelijne-Waver, Belgium
joost.vennekens@cs.kuleuven.be

Abstract. The study of actual causation concerns reasoning about events that have been instrumental in bringing about a particular outcome. Although the subject has long been studied in a number of fields including artificial intelligence, existing approaches have not yet reached the point where their results can be directly applied to explain causation in certain advanced scenarios, such as pin-pointing causes and responsibilities for the behavior of a complex cyber-physical system. We believe that this is due, at least in part, to a lack of distinction between the laws that govern individual states of the world and events whose occurrence cause state to evolve. In this paper, we present a novel approach to reasoning about actual causation that leverages techniques from Reasoning about Actions and Change to identify detailed causal explanations for how an outcome of interest came to be. We also present an implementation of the approach that leverages Answer Set Programming.

Keywords: Causal Reasoning · Reasoning about Actions and Change · Knowledge Representation and Reasoning.

1 Introduction

Actual causation concerns determining how a specified outcome came to be in a given scenario and has long been studied in numerous fields including law, philosophy, and, more recently, computer science and artificial intelligence (AI). Also referred to as *causation in fact*, actual causation is a broad term that encompasses all possible antecedents that have played a meaningful role in producing a consequence [8]. Sophisticated actual causal reasoning has long been prevalent in human society and continues to have an undeniable impact on the advancement of science, technology, medicine, and other important fields. From the development of ancient tools to modern root cause analysis in business and industry, reasoning about causal influence over time in a sequence of events enables us to diagnose the cause of an outcome of interest and gives us insight into how to bring about, or even prevent, similar outcomes in future scenarios.

The ability to automate this kind of reasoning will likely become even more important in the near future due to the ongoing advancement of *deep learning*. Indeed, entrusting important decisions to black-box machine learning algorithms brings with it significant societal risks. To counter these risks, there is a need for Artificial Intelligence systems that are able to explain their behavior in an intuitive way. This is recognized within the scientific community, as witnessed by the emergence of the *explainable AI* domain, and also by the measures enacted by policy makers. For instance, the General Data Protection Regulation (GDPR) has recently come into force in the European Union, a requirement of which is that companies must be able to provide their customers with explanations of algorithmic decisions that affect them.

Explaining the conclusions reached by a single neural network may perhaps not require sophisticated causal reasoning. However, if we consider the behavior of an advanced cyber-physical system, such as a self-driving car, reasoning about causation (e.g. blame or praise) becomes significantly more complex – the car typically contains a large number of software and hardware modules (possibly from different vendors), there may be other cars and pedestrians involved in the scenario of interest, and there may have been wireless communication with other vehicles or a central server, all of which may influence the actions taken by the car's control module over the course of its drive. To reach an intuitively satisfactory explanation of why some outcome of interest came to be in such a domain, the insights that have been produced by the decades-long study of actual causation seem indispensable.

Modern work on actual causation originated in philosophy with the seminal paper by Lewis [29]. His work, like that of other philosophers following him, was of course mainly theoretical and not intended to be put to practical use. The famous Halpern-Pearl (HP) paper [24] initiated interest in this concept within the field of AI and it constitutes a first milestone on the way towards applications of the concept of actual causation. However, neither the HP paper nor the many that have followed it (see also Section 6) have yet reached the point where their results could be directly applied, for example, in the context of a self-driving car as sketched above. We believe that this is due, at least in part, to a lack of distinction between the laws that govern individual states of the world and events whose occurrence cause state to evolve.

The goal of this work is to research and investigate the suitability of techniques from Reasoning about Actions and Change (RAC) for reasoning about and explaining actual causation in domains for which the evolution of the state of the world over time plays a critical role. We utilize the action language $\mathcal{AL}$ [2] to define the constructs of our theoretical framework. While our framework is not strongly tied to this choice of representation language, in this paper we adopt $\mathcal{AL}$ because the language enables us to represent the direct and indirect effects of events on the state of the world, as well as the evolution of state over time in response to their occurrence. $\mathcal{AL}$ also lends itself quite naturally to an automated translation to Answer Set Programming [15,17], using which, reasoning tasks of considerable complexity can be specified and automated.

The organization of the paper is as follows. In the following section, we provide background for the formalization of knowledge and events. Next, we present the technical details of the theoretical framework. Following that, we offer an approach to implementing the framework using Answer Set Programming. We then present an empirical study of the implementation's performance on a number of problem instances. Next, we present a summary of related work, and finally we draw conclusions and discuss directions for future research.

2 Preliminaries

For the representation of the domain and of its evolution over time we rely on action language $\mathcal{AL}$ [2]. $\mathcal{AL}$ is centered around a discrete-state-based representation of the evolution of a domain in response to events. The language $\mathcal{AL}$ builds upon an alphabet consisting of a set $\mathcal{F}$ of *fluents* and a set $\mathcal{E}$ of *elementary events*[4]. Fluents are boolean properties of the domain, whose truth value may change over time. A *(fluent) literal* is a fluent f or its negation $\neg f$. Additionally, we define $\overline{f} = \neg f$ and $\overline{\neg f} = f$. If $f \in \sigma$, we say that f *holds* in σ. A single elementary event is denoted by its element e in $\mathcal{E}$. A *compound event* is a set of elementary events $\epsilon = \{e_1, \ldots, e_n\}$. A statement of the form

$$e \textbf{ causes } l_0 \textbf{ if } l_1, \ldots, l_n \tag{1}$$

is a called a *dynamic (causal) law*. Intuitively, a law of form (1) says that if elementary event e[5] occurs in a state where literals $l_1, \ldots, l_n$ hold, then literal l_0 will hold in the next state. A statement

$$l_0 \textbf{ if } l_1, \ldots, l_n \tag{2}$$

is called a *state constraint* and says that in any state in which literals $l_1, \ldots, l_n$ hold, l_0 also holds. We say that l_0 is the *consequence* of the law. A statement of form (2) allows for an elegant and concise representation of *indirect effects* of events which enhances the expressive power of the language. Finally, a statement of the form

$$e \textbf{ impossible\_if } l_1, \ldots, l_n \tag{3}$$

is called an *executability condition* and states that an elementary event e cannot occur when $l_1, \ldots, l_n$ hold. A set of statements of $\mathcal{AL}$ is called an *action description*.

A set S of literals is *closed under a state constraint* (2) if $\{l_1, \ldots, l_n\} \not\subseteq S$ or $l_0 \in S$. Set S is *consistent* if, for every $f \in \mathcal{F}$, at most one of $\{f, \neg f\}$ is in S. It is *complete* if at least one of $\{f, \neg f\}$ is in S. A *state* σ of an action

[4] For convenience and compatibility with the terminology from RAC, in this paper we use *action* and *event* as synonyms.

[5] We focus on elementary actions for simplicity of presentation. It is straightforward to expand the statements to allow non-elementary actions.

description AD is a complete and consistent set of fluent literals closed under the state constraints of AD.

Given an elementary event e and a state σ, the set of *(direct) effects of* e *in* σ, denoted by $E(e, \sigma)$, is the set that contains a literal l_0 for every dynamic law (1) such that $\{l_1, \ldots, l_n\} \subseteq \sigma$. Given a compound event $\epsilon = \{e_1, \ldots, e_n\}$, the set of direct effects of ϵ in σ, therefore, is given by $E(\epsilon, \sigma) = E(e_1, \sigma) \cup \ldots \cup E(e_n, \sigma)$. Given a set S of literals and a set Z of state constraints, the *set* $Cn_Z(S)$ *of consequences of* S *under* Z is the smallest set of literals that contains S and is closed under every state constraint in Z. Finally, an event e is *non-executable* in a state σ if there exists an executability condition (3) such that $\{l_1, \ldots, l_n\} \subseteq \sigma$. Otherwise, the event is *executable*[6] in σ.

The semantics of an action description AD is defined by its *transition diagram* $\tau(AD)$, a directed graph $\langle N, A \rangle$ such that N is the collection of all states of AD; A is the set of all triples $\langle \sigma, \epsilon, \sigma' \rangle$ where σ, σ' are states, ϵ is an event executable in σ, and σ' satisfies the *successor state equation*:

$$\sigma' = Cn_Z(E(\epsilon, \sigma) \cup (\sigma \cap \sigma')) \tag{4}$$

where Z is the set of all state constraints of AD.

The argument of Cn_Z in (4) is the union of the set of direct effects $E(e, \sigma)$ for all $e \in \epsilon$ with the set $\sigma \cap \sigma'$ of the literals "preserved by inertia". The application of Cn_Z adds the "indirect effects" to this union. A triple $\langle \sigma, \epsilon, \sigma' \rangle \in E$ is called a *transition* of $\tau(AD)$ and σ' is a *successor state of* σ (under ϵ). A sequence $\langle \sigma_1, \epsilon_1, \sigma_2, \ldots, \epsilon_k, \sigma_{k+1} \rangle$ is a *path of* $\tau(AD)$ of length k if every $\langle \sigma_i, \epsilon_i, \sigma_{i+1} \rangle$ is a transition in $\tau(AD)$. We denote the *initial state* of a path ρ by σ_1.

3 Theoretical Framework

In this section we present the constructs of the causal reasoning framework and use them to characterize causal explanations. We then apply the framework to a variant of the well-known Yale Shooting Problem [25].

3.1 Definitions

A *problem* is a tuple $\psi = \langle \theta, \rho, AD \rangle$ where θ is a consistent set of literals we want to explain called an *outcome* and ρ is a path of $\tau(AD)$. We will leverage the framework to identify *causal explanations* for a problem ψ. The first step to explain how an outcome θ came to be in path ρ is to identify the *transition states* of θ in ρ. A transition state indicates the "appearance" of θ in the ρ.

Definition 1. *Given a problem* $\psi = \langle \theta, \rho, AD \rangle$*, a state* σ_j *in* ρ *is a* transition state *of* θ *if* $\theta \not\subseteq \sigma_{j-1}$ *and* $\theta \subseteq \sigma_j$*.*

[6] Note that an event may occur without having an effect on the state of the world, commonly referred to in the literature as a NOP action.

We denote by $T(\psi) = \{\sigma_{j_1}, \ldots, \sigma_{j_m}\}$ the set of transition states with respect to the problem ψ. Intuitively, state σ_j is a transition state of θ if the outcome is satisfied in σ_j but not in the immediately previous state σ_{j-1}. A *causing compound event* ϵ_i of literal l for a transition state σ_j of θ is the most recent compound event to σ_j to result in a transition state σ_{i+1} in ρ.

Definition 2. *Given a problem $\psi = \langle \theta, \rho, AD \rangle$, a state σ_j in ρ, and a literal $l \in \sigma_j$, ϵ_i is a* causing compound event *of l holding in σ_j if σ_{i+1} is a transition state of $\{l\}$ in ρ, $i < j$, and $j - (i+1)$ is minimal.*

Both direct and indirect causes of a literal l holding in a given state σ_j are members of the causing compound event ϵ_i of l holding in σ_j. A *direct cause* $e \in \epsilon_i$ of l is an elementary event in ϵ_i whose occurrence causes l to hold in the subsequent state.

Definition 3. *Given a problem $\psi = \langle \theta, \rho, AD \rangle$, a literal $l \in \sigma_j$ of ρ, and a causing compound event ϵ_i of l, the elementary event $e' \in \epsilon_i$ is a* direct cause *of l for σ_j if l is in the set $E(e', \sigma_i)$.*

We denote by $D(\sigma_j)$ the set containing a tuple $\langle \epsilon_i, e', l \rangle$ for every elementary event e' in each ϵ_i such that e' is a direct cause of a literal $l \in \theta$ for σ_j. Note that direct cause is defined so that multiple events in ϵ_i can be direct causes as long as l is in the corresponding sets of direct effects. An *indirect cause* of literal l is a subset[7] of a causing compound event of l.

Definition 4. *Given a problem $\psi = \langle \theta, \rho, AD \rangle$, state σ_j in ρ, a literal $l \in \sigma_j$, and a causing compound event ϵ_i of l, the compound event $\epsilon' \subseteq \epsilon_i$ is an* indirect cause *of l for σ_j if it is a smallest subset of ϵ_i such that the following conditions are satisfied:*

1. *$l \notin E(\epsilon', \sigma_i)$*
2. *There exists a transition $t = \langle \sigma_i, \epsilon', \sigma'_{i+i} \rangle$ in $\tau(AD)$ such that σ'_{i+1} is a transition state of $\{l\}$ in t*

We denote by $I(\sigma_j)$ a set containing a tuple $\langle \epsilon_i, \epsilon', l \rangle$ for every compound event ϵ' in every ϵ_i in ρ such that ϵ' is a indirect cause of l for σ_j in ρ.

Condition 1 ensures that l is not a direct effect of ϵ'. Condition 2 states that if ϵ' were to hypothetically occur by itself in state σ_i, then it would have caused l. In other words, we know that l does not hold in σ_i and that l is not a direct effect of ϵ'. Therefore, if ϵ' occurs by itself and l holds in the resulting state σ'_{i+1}, then it must be the case that l is an indirect effect of ϵ'. Finally, we require that ϵ' is a *smallest* subset of ϵ_i because we want to rule out any subsets including extraneous elementary events. For example, if ϵ' contains three events and only two are needed to indirectly cause l, then there would indeed

[7] In $\mathcal{AL}$, it is possible that a set of literals must hold simultaneously in order to cause a literal to hold. Consider $AD = \{a$ **causes** $b; c$ **causes** $d; e$ **if** $b, d\}$ of a causing compound event ϵ_i of l.

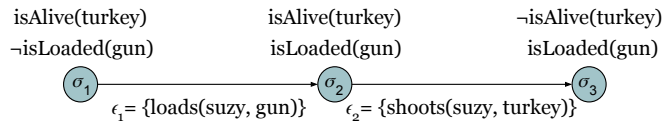

Fig. 1. Path ρ_Y is a representation of the Yale shooting scenario.

be transition $t = \langle \sigma_i, \epsilon', \sigma'_{i+1} \rangle$ in $\tau(AD)$ as required by condition 2. However, we want subsets containing *only* those events that have contributed to causing l.

By now we have defined direct and indirect causation of literals, however, these definitions alone do not provide a comprehensive explanation for an appearance of θ in ρ. Therefore, we define a *causal explanation*, which is a tuple containing the sets of direct and indirect causes of literals in θ in their respective transition state, given by $D(\sigma_j)$ and $I(\sigma_j)$.

Definition 5. *Given a problem $\psi = \langle \theta, \rho, AD \rangle$, a path $\rho \in \tau(AD)$, and a transition state σ_j of θ in ρ, a causal explanation of θ being satisfied in σ_j in path ρ is the tuple $\mathcal{C}(\psi, \sigma_j) = \langle D(\sigma_j), I(\sigma_j) \rangle$.*

Literals that were not caused by any event in ρ are omitted from the causal explanation. This choice is motivated by the idea that no cause can be identified for literals that were not caused.

3.2 Yale Shooting Problem

Here we use the framework defined above to solve a variant of the well-known Yale shooting problem (YSP) from [25]. The scenario is as follows:

> *Shooting a turkey with a loaded gun will kill it. Suzy shoots the turkey. What is the cause of the turkey's death?*

The YSP problem is formalized by $\Psi_Y = \langle \theta_Y, v_Y, AD_Y \rangle$. The outcome of interest is $\theta_Y = \{\neg isAlive(turkey)\}$. The sequence of events is given by $v_Y = \{\epsilon_1, \epsilon_2\}$ where $\epsilon_1 = \{loads(suzy, gun)\}$ and $\epsilon_2 = \{shoots(suzy, turkey)\}$. The action description AD_Y characterizes the events of the YSP domain:

$$\begin{cases} shoots(X, turkey) \textbf{ causes } \neg isAlive(turkey) \textbf{ if } isAlive(turkey) & (5) \\ shoots(X, turkey) \textbf{ impossible\_if } \neg isLoaded(gun) & (6) \\ \quad loads(X, gun) \textbf{ causes } isLoaded(gun) \textbf{ if } \neg isLoaded(gun) & (7) \end{cases}$$

Laws (5) and (7) are straightforward dynamic laws describing the effects of the events in the YSP domain. Law (6) states that the turkey cannot be shot if the gun is not loaded. Consider the path ρ_Y, represented in Figure 1. In the initial state of ρ_Y, the turkey is alive, and the turkey is dead in the

final state of the path after the occurrence of $\epsilon_1 = \{loads(suzy, gun)\}$ and $\epsilon_2 = \{shoots(suzy, turkey)\}$.

It is straightforward to verify for the problem $\psi_Y = \langle \theta_Y, \rho_Y, AD_Y \rangle$ that σ_3 is the only transition state of θ_Y and that ϵ_2 is the causing compound event of $\neg isAlive(turkey)$. The elementary event $shoots(suzy, turkey)$ in ϵ_2 is a direct cause of $\neg isAlive(turkey)$ as per rule (5). The causal explanation for Ψ_Y is therefore $\mathcal{C}(\Psi_Y, \sigma_3) = \langle \{\langle \epsilon_2, shoots(suzy, turkey), \neg isAlive(turkey) \rangle\}, \{\} \rangle$. We have used the framework to identify only Suzy's shooting of the turkey as a direct cause of its death, which corresponds to the intuition about the problem. Moreover, we did not any identify indirect causes of the turkey's death, denoted in the explanation by the empty set for $I(\sigma_3)$. If we want to know why the gun was loaded so that Suzy could kill the turkey, we use rule (6) to formulate the subproblem $\psi'_Y = \langle \{isLoaded(gun)\}, \rho_Y, AD_Y \rangle$ to determine that $loads(suzy, gun)$ directly caused the gun to be loaded. Appendix 2.1 of [28] of this paper presents a novel adaptation of YSP to demonstrate explaining the indirect causation of the turkey's death. Note that it is straightforward to represent and reason about other examples from the causality literature, such as the bottle-shattering example from [20].

4 ASP Implementation of the Framework

In this section, we present an approach to computing causal explanations via Answer Set Programming (ASP) [16,18], a form of declarative programming that is useful in knowledge-intensive applications. In the ASP methodology, problem-solving tasks are reduced to computing answer sets of suitable logic programs. As demonstrated by a substantial body of literature (see, e.g., [1]), $\mathcal{AL}$ lends itself quite naturally to an automated translation to Answer Set Programming [15,17], using which, reasoning tasks of considerable complexity can be specified and executed (see, e.g., [9,11,12]). As such, ASP is well suited to the task of computing causal explanations. We begin this section with a discussion of the syntax and semantics of Answer Set Programming.

4.1 Answer Set Programming

Let Σ be a signature containing constant, function and predicate symbols. Terms and atoms are formed as in first-order logic. A *literal* is an atom a or its strong negation $\neg a$. Literals are combined to form rules that represent both domain knowledge and events in our approach. A *rule* in ASP is a statement of the form:

$$h \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n$$

where h_i's (the head) and l_i's (the body) are literals and *not* is the so-called *default negation*. Intuitively, the meaning of default negation is the following: "if you believe $\{l_1, \ldots, l_m\}$ and have no reason to believe $\{l_{m+1}, \ldots, l_n\}$, then you must believe h". An ASP rule with an empty body is called a fact. In writing facts, the $\leftarrow$ connective is dropped. Rules of the form $\bot \leftarrow l_1, \ldots, \text{not } l_n$ are

abbreviated $\leftarrow l_1, \ldots, \text{not } l_n$, and called *constraints*, intuitively meaning that $\{l_1, \ldots, \text{not } l_n\}$ must not be satisfied. A rule with variables (denoted by an uppercase initial) is interpreted as a shorthand for the set of rules obtained by replacing the variables with all possible variable-free terms. A *program* Π is a set of rules over Σ.

A consistent set S of domain literals is closed under a rule if $h \in S$ whenever $\{l_1, \ldots, l_m\} \subseteq S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$. Set S is an answer set of a *not*-free program Π if S is the minimal set closed under its rules. The reduct, Π^S, of a program Π w.r.t. S is obtained from Π by removing every rule containing an expression "not l" s.t. $l \in S$ and by removing every other occurrence of not l. Finally, set S is an answer set of a program Π if S is the answer set of Π^S.

For a convenient representation of choices, in this paper we also use *constraint literals*, which are expressions of the form $m\{l_1, l_2, \ldots, l_k\}n$, where m, n are arithmetic expressions and l_i's are basic literals. A constraint literal is satisfied w.r.t. S whenever $m \leq |\{l_1, \ldots, l_k\} \cap S| \leq n$. Constraint literals are especially useful to reason about available choices. For example, a rule $1\{p, q, r\}1.$ intuitively states that exactly one of $\{p, q, r\}$ should occur in every answer set.

4.2 Framework Implementation

We begin our approach to computing causal explanations by encoding the elements of a problem $\Psi = \langle \theta, \rho, AD \rangle$.

Problem Translation. For an outcome θ, set $\alpha(\theta)$ contains a fact $outcome(l, theta)$ as well as facts $inOutcome(l, theta)$, $olit(l)$, and $inOutcome(l, olit(l))$ for every $l \in \theta$. We use the $olit(l)$ notation to denote the outcome coinciding with the singleton $\{l\}$.

The elements of a path ρ are given by the sets $\alpha(\rho_1)$, $\alpha(\rho_2)$, and $\alpha(\rho_3)$. The set $\alpha(\rho_1)$ contains a fact $occurs(e, i)$ for every $e \in \epsilon_i$ and a fact $holds(l, i)$ for each literal $l \in \sigma_i$ where $1 < i < k + 1$. The set also contains facts $event(e)$ and $fluent(f)$ for each $e \in \mathcal{E}$ and $f \in \mathcal{F}$, respectively. Next, the set $\alpha(\rho_2)$ contains the facts $subset(\lambda)$, where λ is a unique identifier for C, and $inSubset(e, \lambda)$ for every $e \in C$ for every subset C of $\mathcal{E}$. The subsets of $\mathcal{E}$ will be useful later to identify indirect causation, and are included as elements of ρ because they are specific to the path. The set $\alpha(\rho_3)$ characterizes the transitions of ρ as sequence of steps.

We refer to the steps of a path as *concrete steps*, or *c-steps*, to differentiate them from *hypothetical steps*, which will be discussed later in connection with indirect causes. For related reasons, we also represent the sequence of c-steps $cstep(1), cstep(2), \ldots, cstep(k + 1)$, where k is the length of the path, by means of the rule:

$$next(I1, I2) \leftarrow \quad cstep(I1),\ cstep(I2),\ I2 = I1 + 1. \tag{8}$$

Finally, $\alpha(\rho_3)$ includes the rule $step(I) \leftarrow cstep(I)$ which states that c-steps are types of steps. The set $\alpha(\rho) = \{\alpha(\rho_1) \cup \alpha(\rho_2) \cup \alpha(\rho_3)\}$ represents the path ρ.

We translate laws of $\mathcal{AL}$ to ASP as follows. For dynamic laws, the translation $\alpha(e$ **causes** l_0 **if** $l_1, \ldots, l_n)$ is the collection of atoms $d\_law(d)$, $head(d, l_0)$, $event(d, e)$, $prec(d, 1, l_1), \ldots, prec(d, n, l_n)$, and $prec(d, n+1, nil)$. For state constraints, $\alpha(l_0$ **if** $l_1, \ldots, l_n)$ is the collection of atoms $s\_law(s)$, $head(s, l_0)$, $prec(s, 1, l_1), \ldots, prec(s, n, l_n)$, and $prec(s, n+1, nil)$. Finally, for executability conditions, $\alpha(e$ **impossible_if** $l_1, \ldots, l_n)$ is the collection of atoms $i\_law(\iota)$, $event(\iota, e)$, $prec(\iota, 1, l_1), \ldots, prec(\iota, n, l_n)$, and $prec(\iota, n+1, nil)$.

The semantics of $\mathcal{AL}$ are captured by the rules of program Π, an approach is adapted from [1]. The program describes the effects of dynamic laws and state constraints and enforces executability conditions. It also defines when the preconditions of a translated law are satisfied (i.e. $prec(X, Y)$ atoms from the translation of $\mathcal{AL}$ laws to ASP), denoted by the predicate $prec_h(R, I)$ where R is the identifier of a translated $\mathcal{AL}$ law and I is a step in the ASP representation of ρ[8]. Finally, Π contains rules describing inertia [26] (i.e. things usually stay as they are) and consistency. See Appendix 1.2 of [28] for an expanded description of Π and a full listing of its rules.

Transition and Causing Steps. The rules in set Π_{σ_j} characterize a transition state σ_j of θ in path ρ. We use the term *transition step* in accordance with our representation of states in ρ as c-steps, a type of step.

$$\Pi_{\sigma_j} \begin{cases} transitionStep(OC, J2) \leftarrow & step(J1),\ step(J2), next(J1, J2), \\ & next(J1, J2), outcome(OC), \\ & ocSat(OC, J2), \neg ocSat(OC, J1). \quad (9) \\ \neg ocSat(OC, J) \leftarrow & step(J), inOutcome(OC, L), \\ & \text{not } holds(L, J). \quad (10) \\ ocSat(OC, J) \leftarrow & step(J), \text{not } \neg ocSat(OC, J). \quad (11) \end{cases}$$

Rule (9) states that $J2$ is a transition step of outcome OC if it is satisfied in step $J2$ and not $J1$. (10) and (11) tell us when OC is or is not satisfied in a given step J, respectively. Note that transition steps leverage as *steps* rather than c-steps, allowing flexibility to reason about other types. The rules of Π_{ϵ_i} describe causing steps of a literal holding at step I.

[8] We will use the predicate $prec_h$ when computing both direct and indirect causes.

$$\Pi_{\epsilon_i}\begin{cases} possCausingStep(I,L,J) \leftarrow & cstep(I),\ cstep(J), I<J, \\ & transitionStep(olit(L), I+1), \\ & transitionStep(theta, J). \\ \neg causingStep(I,L,J) \leftarrow & possCausingStep(I,L,J), \\ & possCausingStep(I',L,J), \\ & I<I', I'<J. \\ causingStep(I,L,J) \leftarrow & possCausingStep(I,L,J), \\ & \text{not } \neg causingStep(I,L,J). \end{cases} \quad (12)\ (13)\ (14)$$

Rule (12) states that a c-step I is a *possible* causing step of L holding in J if it occurs prior to c-step J, $I+1$ is a transition step of $outcome(olit(L))$, and J is a transition step for the main outcome $theta$. It is easy to see that rule (12) corresponds to the conditions of Definition 2 for compound causing events. Rule (13) corresponds to condition 3 of Definition 2 by stating that c-step I cannot be a causing step of J if there is another possible causing step I' that closer to J. Again we can use inequalities to determine relative position, this time for two earlier c-steps. Finally, (14) is a straightforward rule stating that a possible causing step I of L in c-step J is a causing step if we have no reason to believe that it is *not* a causing step.

Direct and Indirect Causes. Here we present ASP translations of the definitions of direct and indirect cause. The rules of Π_{D_θ} describe when an event that occurred at causing step I has directly caused L to hold in c-step I.

$$\Pi_{D_\theta}\begin{cases} directEffect(L,E,I) \leftarrow & cstep(I), d\_law(D), event(D,E), \\ & occurs(E,I), prec\_h(D,I), head(D,L). \\ directCause(E,I,L,J) \leftarrow & causingStep(I,L,J), \\ & directEffect(L,E,I). \end{cases} \quad (15)\ (16)$$

Rule (15) states that L is a direct effect of event E occurring at I when all of the preconditions of the dynamic law D are satisfied at I. Rule (16) states that E occurring at I is a direct cause of L holding at c-step J if I is a causing step of L in J and L is a direct effect of E as per rule (15).

Finally, the program Π_{I_θ} contains rules used to identify indirect causation. In the interest of space, we favor discussing the approach at a high level, presenting only the most significant rules of the program to facilitate the presentation. However, the full specification of Π_{I_θ} is given in Appendix 1.3 of [28]. Given a causing step I, we want to know if any subset of events that occurred at I caused the literal under consideration to hold indirectly. Program Π_{I_θ} states that an event subset C (i.e. $subset(C)$) occurring at step I is a possible indirect cause of L holding at step J if I is a causing step for L and we have no reason to believe that the event(s) in C (i.e. all $inSubset(E,C)$ atoms) caused L directly.

Recall that condition 2 of Definition 4 states that if $\epsilon' \in \epsilon_i$ is an indirect cause of l, then a transition t' must exist in $\tau(AD)$ such that if ϵ' occurring by itself in σ_i results in a transition step of $\{l\}$. Given a possible indirect causing subset C occurring at step I, we accomplish this reasoning in ASP by creating a sequence of two hypothetical steps, given by $hstep(\mu(C, I))$ and $hstep(\mu'(C, I))$, whose initial state is identical to I and testing to see if L holds after the occurrence of C's events. Once C has passed the hypothetical reasoning step, we are ready to determine indirect causation. With the use of a rule whose head $\neg smallest(C, L, I)$ becomes true when it is proven that C is a smallest possibly indirectly causing subset, the following rule describes when a subset C is an actual cause:

$$indirectCause(C, I, L, J) \leftarrow \quad hypotheticalPass(C, I, L, J), \quad \text{not } \neg smallest(C, L, I). \tag{17}$$

In short, C occurring at c-step I is an indirect cause of L holding at c-step J if C passes the hypothetical step and there is no reason to believe that it is not a smallest possible causing subset of L at I. Note that this implementation returns information about direct and indirect causes of literals of translated θ, but no comprehensive causal explanation. A causal explanation can be easily extracted from an answer set through the literals formed by relations $directCause(E, I, L, J)$ and $indirectCause(C, I, L, J)$. See appendices 2.2 and 2.3 [28] for ASP encodings of the direct and indirect Yale Shooting Problem adaptations from Section 3.

5 Empirical Study of the Implementation

Although an exhaustive experimental evaluation is beyond the scope of this paper, we present results from a preliminary evaluation aiming to assess the feasibility of the approach. To the best of our knowledge, there is no established set of benchmarks for the type of reasoning presented in this paper and so we have generated a set of novel problem instances that allow us to evaluate the framework's performance with respect to a number of problem features.

Problem instances are defined as follows. Given a number of literals L to explain and an allowed number of events per step EPS, the resulting problem instance's outcome contains L literals caused by E events distributed over $S = \lceil \frac{L}{E} \rceil$ steps of the instance's path. The transition step of the outcome is always $S + 1$. When $S = 1$, we say that the causes are *fully concurrent*. When $S = L$, on the other hand, we say that the causes occur in a *strict sequence*. We use the abbreviations FCDC and FCIC to denote full concurrency for direct causation and indirect causation, respectively. Similarly, we use SSDC and SSIC to denote strict sequences for direct and indirect causation.

Explaining cases of full concurrency and strict sequences. We first compared runtime needed to compute full concurrency and strict sequences for direct

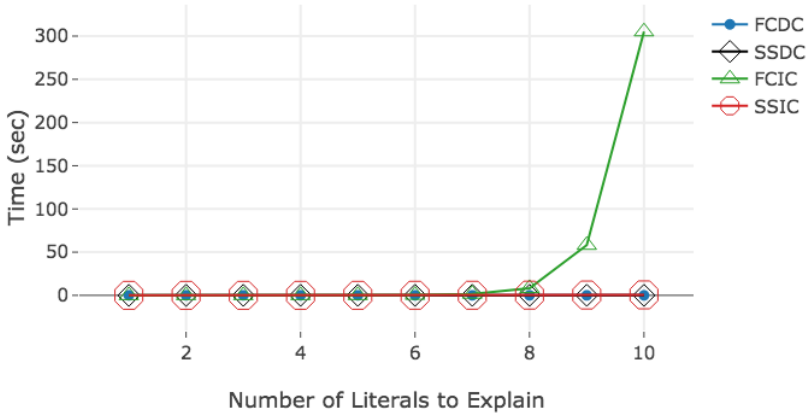

Fig. 2. Relating the explanation of literals and time (direct and indirect causation).

and indirect causes. In this experiment, we varied L from 1 to 10, allowed 10 events per step (EPS) for the fully concurrent cases, and allowed 1 event per step for the strict sequence cases.

The computation times are shown in Figure 2. FCIC is the most challenging type of problem, taking approximately 300 seconds to compute 10 simultaneous indirect causes, overtaking computation of the other cases by a factor [9] of approximately 450. This can be explained by noticing that increasing the number of events occurring in a single step requires the program to perform exhaustive hypothetical reasoning for more and more subsets. Note that at 10 literals to explain there is little discernible difference in the time needed to compute explanations for SSDC, FCDC, and SSIC cases.

We also measured the framework's performance on greater values of L (i.e. larger outcomes). At 50 literals to explain, we found that SSIC overtakes both FCDC and SSDC by a factor of 240, with SSIC taking approximately 190 seconds and SSDC taking approximately 0.8 seconds (see Appendix 3 Figure 2 of [28]). A possible explanation for this is that that the program must initially rule out the possibility that a subset is a direct cause and *then* perform the hypothetical step to confirm indirect causation.

Extending fully concurrent causes towards a strict sequence. Note that in the previous experiment, we were unable to see a significant difference in performance in direct cases for 50 literals. Here, we explore how L causes distributed over $\lceil \frac{L}{EPS} \rceil$ steps affects the performance of the framework for a subset of values for E between 1 and L for direct causes and indirect causes. In the direct causation case, we varied L between 1 and 50 and allowed EPS to take on

[9] The strict sequence indirect cause (SSIC) case takes the second to longest time to explain 10 literals at 0.67 seconds.

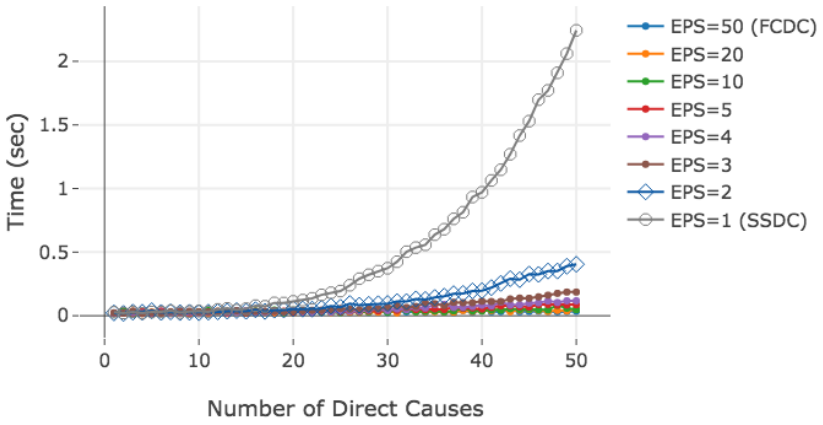

Fig. 3. Varying the number of events per step (EPS) for direct causes.

the values between 1 and 50. The times for this experiment are shown in Figure 3. SSDC, or $EPS = 1$, is the most challenging problem, overtaking $EPS = 2$ by a factor[10] of approximately 6.25. This can be explained by the fact that the program has to reason backward over the path to identify the causing event for all 50 literals in the outcome. In the FCIC case, the program only reasons back over one step for each literal. For the remaining values of $EPS > 2$, explaining 50 literals takes less than 0.25 seconds to compute.

In the case of indirect causation, we varied both L and EPS between 1 and 10. As we saw in Figure 2, FCIC for 10 indirect causes takes approximately 300 seconds to compute (see Appendix 3 Figure 3 of [28]). We observed that FCIC overtakes the second longest computation by a factor of approximately 2.6. While the computation time is large for $EPS = 8, 9, 10$, Figure 3 shows little difference in performance for smaller values of EPS. In order to gain insight into the relationships among times to compute explanations for smaller values of EPS, Figure 4 in Appendix 3 of [28] shows the performance for $EPS \leq 9$, showing that explaining 10 literals that were caused over $\lceil 10/9 \rceil = 2$ steps takes approximately 100 seconds, overtaking $EPS = 8$ by a factor of 5. For smaller values of EPS, the time is at most 5 seconds.

Overall Considerations. A comprehensive evaluation is needed before general claims can be made, but we believe these experiments show that the approach is promising. As we have already stated, the most challenging problem appears to be fully concurrent indirect causes due to an increasing number of event subsets to reason about for each literal that must be explained. However, when the value of EPS is closer to 1 for the indirect case, the literals can be explained in under

[10] The SSDC case takes the second longest time to explain 50 literals at approximately 0.4 seconds

5 seconds. The best performance is seen for larger values of EPS nearing L for direct causation, requiring on average less than half a second to explain 50 directly caused literals.

6 Overview of Related Work

Attempts to mathematically characterize actual causation have largely pursued counterfactual analysis of structural equations [22,24,31,35], neuron diagrams [21], and other logical formalisms [5,27]. Counterfactual accounts of actual causation are typically inspired by the human intuition that if X caused Y, then not Y if not X [29]. It has been widely documented, however, that the counterfactual criteria alone is problematic and fails to recognize causation in a number of common cases such as overdetermination, preemption, and joint causation [10,19,30]. In cases of overdetermination, for example, removing one of the multiple sufficient causes from the scenario will not prevent the outcome from occurring. Therefore, if X and Y are both sufficient to cause Z, the counterfactual definition of cause may not identify X or Y as an actual cause because removing one or the other will not prevent Z. Similarly, it is straightforward to verify that the counterfactual approach may fail to identify causation in cases of preemption and joint causation.

More recent approaches such as [27,23,34] have addressed some of the shortcomings associated with the counterfacual criterion by modifying the existing definitions of actual cause or by modeling change over time with some improved results. However, there is still no widely agreed upon counterfactual definition of actual cause in spite of a considerably large body of work aiming to find one. This suggests that alternate approaches should be explored to explain why an outcome of interest has come to be in a scenario.

In [4], the authors depart from the counterfactual approach, using a similar insight to our own that actual causation can be determined by inspecting a specific scenario rather than hypothesizing strictly about counterfactual worlds. Although the conceptual approach is similar, the technical approaches differ significantly. Leveraging the Situation Calculus (SC) to formalize knowledge, the approach identifies a sequence of event(s) that caused an SC formula ϕ to become true in a scenario. Our framework is capable of explaining a set of causal explanations for an outcome identifying not only causing events (or a sequence of events using multiple problems on the same path), but details about *how* each event influenced the outcome. There are also ramifications due to the choices for the formalization of the domain. Compared to $\mathcal{AL}$ formalizations, SC formalizations incur limitations when it comes to the representations of indirect effects of actions, which play an important role in our work, and the elaboration tolerance of the formalization. Additionally, SC relies on First-Order Logic, while $\mathcal{AL}$ features an independent and arguably simpler semantics.

A number of other interesting approaches exist linking causality and logic programming (LP) with varying goals (e.g. encoding the HP approach for LP [3,6], explaining answer sets of ASP programs [7,33], reasoning about causal

information [14,13,32]). Research relating these topics is steadily advancing, prompting interdisciplinary discussion and exploration of the role and placement of causal reasoning and LP in the landscape of modern computer theory and the software industry.

7 Conclusions and Future Work

The aim of the work presented here is to lay the foundations of actual causal explanation from a representation and reasoning standpoint, leveraging techniques from Reasoning about Actions and Change to represent scenarios and identify actual causal explanations for an outcome of interest. We believe that we have demonstrated that our approach to representing and reasoning about actual causation is promising and practically feasible. In addition to further evaluating the implementation, an important next step will be to conduct a comparative analysis with related approaches to reasoning about actual causation. Another open problem is to investigate extensions of the framework to support the representation of time-delayed effects, probabilities, and triggered events.

References

1. Balduccini, M., Gelfond, M.: Diagnostic reasoning with A-Prolog. Journal of Theory and Practice of Logic Programming (TPLP) **3**(4–5), 425–461 (Jul 2003)
2. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. In: Logic-based artificial intelligence, pp. 257–279. Springer (2000)
3. Baral, C., Hunsaker, M.: Using the probabilistic logic programming language p-log for causal and counterfactual reasoning and non-naive conditioning. In: IJCAI. pp. 243–249 (2007)
4. Batusov, V., Soutchanski, M.: Situation calculus semantics for actual causality. In: 13th International Symposium on Commonsense Reasoning. University College London, UK. Monday, November. vol. 6 (2017)
5. Beckers, S., Vennekens, J.: A general framework for defining and extending actual causation using cp-logic. International Journal of Approximate Reasoning **77**, 105–126 (2016)
6. Bochman, A., Lifschitz, V.: Pearl's causality in a logical setting. In: AAAI. pp. 1446–1452 (2015)
7. Cabalar, P., Fandinno, J., Fink, M.: Causal graph justifications of logic programs. Theory and Practice of Logic Programming **14**(4-5), 603–618 (2014)
8. Carpenter, C.E.: Concurrent causation. University of Pennsylvania Law Review and American Law Register **83**(8), 941–952 (1935)
9. Dix, J., Kuter, U., Nau, D.: Planning in answer set programming using ordered task decomposition. In: Annual Conference on Artificial Intelligence. pp. 490–504. Springer (2003)
10. Dobbs, D.B.: Rethinking actual causation in tort law (2017)
11. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Answer set planning under action costs. Journal of Artificial Intelligence Research **19**, 25–71 (2003)
12. Erdem, E., Gelfond, M., Leone, N.: Applications of answer set programming. AI Magazine **37**(3) (2016)

13. Fandinno, J.: Deriving conclusions from non-monotonic cause-effect relations. Theory and Practice of Logic Programming **16**(5-6), 670–687 (2016)
14. Fandinno, J.: Towards deriving conclusions from cause-effect relations. Fundamenta Informaticae **147**(1), 93–131 (2016)
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. vol. 88, pp. 1070–1080 (1988)
16. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of ICLP-88. pp. 1070–1080 (1988)
17. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New generation computing **9**(3-4), 365–385 (1991)
18. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9**, 365–385 (1991)
19. Glymour, C., Danks, D.: Actual causation: a stone soup essay. Synthese **175**(2), 169–192 (2010)
20. Hall, N.: Two concepts of causation. Causation and counterfactuals pp. 225–276 (2004)
21. Hall, N.: Structural equations and causation. Philosophical Studies **132**(1), 109–136 (2007)
22. Halpern, J.Y.: Axiomatizing causal reasoning. Journal of Artificial Intelligence Research **12**, 317–337 (2000)
23. Halpern, J.Y.: Actual causality. MIT Press (2016)
24. Halpern, J.Y., Pearl, J.: Causes and explanations: A structural-model approach. part i: Causes. The British journal for the philosophy of science **56**(4), 843–887 (2005)
25. Hanks, S., McDermott, D.: Nonmonotonic logic and temporal projection. Artificial intelligence **33**(3), 379–412 (1987)
26. Hayes, P.J., McCarthy, J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 4, pp. 463–502. Edinburgh University Press (1969)
27. Hopkins, M., Pearl, J.: Causality and counterfactuals in the situation calculus. Journal of Logic and Computation **17**(5), 939–953 (2007)
28. LeBlanc, E., Balduccini, M., Vennekens, J.: Appendices of explaining actual causation via reasoning about actions and change (JELIA 2019). `http://eleblanc.ai/files/lbv-jelia2019-appendices.pdf`
29. Lewis, D.: Causation. The journal of philosophy **70**(17), 556–567 (1973)
30. Menzies, P.: Counterfactual theories of causation. The Stanford Encyclopedia of Philosophy (2001)
31. Pearl, J.: On the definition of actual cause. Tech. rep., University of California (1998)
32. Pereira, L.M., Saptawijaya, A.: Counterfactuals, logic programming and agent morality. R. Urbaniak, G. Payette (eds.), Applications of Formal Philosophy: The Road Less Travelled, Springer Logic, Argumentation & Reasoning series (20187)
33. Pontelli, E., Son, T.C., Elkhatib, O.: Justifications for logic programs under answer set semantics. Theory and Practice of Logic Programming **9**(1), 1–56 (2009)
34. Vennekens, J.: Actual causation in cp-logic. Theory and Practice of Logic Programming **11**(4-5), 647–662 (2011)
35. Weslake, B.: A partial theory of actual causation. British Journal for the Philosophy of Science (2015)