When is a Program an Actual Cause?

Bita Banihashemi*[1], Shakil M. Khan*[2] and Mikhail Soutchanski[3]

[1]York University, 4700 Keele St, Toronto, ON M3J 1P3, Canada
[2]University of Regina, 3737 Wascana Pkwy, Regina, SK S4S 0A2, Canada
[3](Former) Ryerson University, 350 Victoria St, Toronto, ON M5B 2K3, Canada

Abstract

Causality plays a central role in reasoning about observations. In many cases, it might be useful to define the conditions under which a non-deterministic program can be called an actual cause of an effect in a setting where a sequence of programs are executed one after another. There can be two perspectives, one where at least one execution of the program leads to the effect, and another where all executions do so. The former captures a "weak" notion of causation and is more general than the latter stronger notion. In this paper, we give the definition of weak potential causes. Our analysis is performed within the situation calculus basic action theories and we consider programs formulated in the logic programming language ConGolog. Within this setting, we show how one can utilize a recently developed abstraction framework to relate causes at various levels of abstraction. In particular, we prove that under some conditions, causes defined at an abstract and a concrete level can be related with each other in a kind of commutative diagram.

Keywords

Actual Cause, Abstraction, The Situation Calculus

1. Introduction

Actual or token causation is concerned with identifying the events or actions in a trace that can be considered as causes of an observed effect. The seminal work of Pearl [1] provided the foundations and served as inspiration for research of actual cause in AI. This research culminated in the book [2] that summarized a number of previously developed definitions concerning when an event can be considered as an actual cause of an effect. These definitions are developed within the framework of structural equations models (SEM), where a simple event is understood as assigning a value to an endogenous variable.

However, this perspective does not facilitate the study of causation for more complex activities such as control flow in programs. It can be interesting and important to define when a non-deterministic program is an actual cause of an effect in a setting where a sequence of programs are executed one after another. This immediately leads to the question when can one intuitively say that a program is an actual cause? One perspective can be that a non-deterministic program is a *weak potential cause*, if at least one execution of the program leads to a situation where the effect holds. Another perspective is that a program is a *strong potential cause* if all executions of the program produce the effect. Note that a strong potential cause is also weak, but not vice versa. Also, in both cases we talk about *potential* causes, since they can manifest only in some of the situations that are produced by the execution of the program sequence.

As an example, imagine *Suzy* buying a lottery ticket that later wins a reward. If one conceptualizes the complex actions of purchasing the ticket as a highly non-deterministic program, then it is reasonable to say that this program was a weak potential cause of the fact that *Suzy* won, since there is an execution of this program that leads to a situation where the effect holds and another to a situation where it doesn't.

Again, imagine a computer system that involves multiple interacting agents. The typical examples of such systems arise in computer security contexts where the behaviour of the agents is specified by non-deterministic protocols due to versatility of possible agent interactions. In this context, one might be interested in determining if all of the executions of a protocol led to the successful handling of a security leak. This corresponds to the case of a strong potential cause.

In this paper, we give a definition of the more inclusive notion of weak cause. We consider programs formulated in the high-level logic programming language ConGolog [3], which is based on action theories specified in the situation calculus (SC) [4, 5]. We build on a recently proposed definition of actual cause in the SC [6], which only considers primitive actions as causes. Since we focus on programs as causes, a natural question that arises then is how these two notions can be related. The programs can be complex, but often they can be conceptualized

*Contributed equally to this paper

Workshop on Causal Reasoning and Explanation in Logic Programming - 37th International Conference on Logic Programming

bita@eecs.yorku.ca (B. Banihashemi*); skhan@cs.uregina.ca (S. M. Khan*); mes@cs.ryerson.ca (M. Soutchanski)
https://www.cs.uregina.ca/~skhan/ (S. M. Khan*); http://www.cs.ryerson.ca/~mes/ (M. Soutchanski)
0000-0002-7147-484X (B. Banihashemi*); 0000-0003-0140-3584 (S. M. Khan*)

at some abstract high-level (HL) as actions. It turns out that the abstraction framework proposed in [7] that can relate programs with primitive actions is also useful for relating a subclass of weak potential causes (in particular, weak causes that are also strong) at different levels of abstraction.

On the semantic level, models of programs can be very complicated, but reasoning about effects of actions that serve as their abstractions can be easier since essential details are encapsulated in a simpler HL model. We argue that HL and low-level (LL) causes can be related in a kind of commutative diagram. Namely, if a HL action is found to be a cause of an effect, this action is associated to a program δ defined over a LL theory that implements it, and this effect is an abstraction of a LL formula ϕ (i.e. ϕ is a refinement of the effect), then at the LL, δ must be a cause of ϕ. This result is one of our main contributions. We focus here on semantics and leave computational issues to future work.

2. Preliminaries

Our base framework for this is the situation calculus (SC) [4] as formalized in [5]. We assume that there is a *finite number of action types* $\mathcal{A}$. Moreover, we assume that the terms of object sort are a countably infinite set $\mathcal{N}$ of standard names for which we have the unique names assumption and domain closure. For simplicity, and w.l.o.g., we assume that there are no functions other than constants and no non-fluent predicates.

A basic action theory (BAT) $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent, (second-order, or SO) axioms of the SC; (first-order, or FO) precondition axioms; (FO) successor state axioms (SSAs) describing how fluents change between situations; (FO) unique names axioms for actions and (FO) domain closure on action types; (SO) unique name axioms and domain closure for object constants; and (FO) axioms describing the initial configuration of the world. A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; precondition axioms characterize this predicate. The abbreviation $Executable(s)$ means that every action performed in reaching situation s was possible in the situation in which it occurred. A precedence relation on situations s and s' denoted by $s \leq s'$ states that s' is a successor situation of s and that every action between s and s' is in fact executable. We write $do([a_1, a_2, \ldots, a_{n-1}, a_n], s)$ as an abbreviation for $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, s)) \ldots))$; for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$.

A SC formula is *uniform* in s iff it does not mention $Poss$, $\sqsubset$, or equality on situations, it does not quantify over situations, and whenever it mentions a term of sort situation then that term is s. Also, we use uppercase Greek letters for situation-suppressed SC formulae and we denote by $\Phi[s]$ the formula obtained from Φ by restoring the situation argument s into all fluents in Φ. To represent and reason about complex actions, various *high-level programming languages* have been defined. Here we concentrate on (a fragment of) ConGolog [3] that includes the following constructs:

$$\delta ::= nil \mid \alpha \mid \Phi? \mid (\delta_1 ; \delta_2) \mid (\delta_1 | \delta_2) \mid (\pi x.\delta(x)) \mid \delta^* \mid (\delta_1 \| \delta_2).$$

Thus, complex actions can be composed using constructs that include the *empty program* (nil), primitive actions (α), waiting for a condition ($\Phi?$), sequence ($\delta_1 ; \delta_2$), nondeterministic branch ($\delta_1 | \delta_2$), nondeterministic choice of arguments ($\pi x.\delta(x)$), nondeterministic iteration (δ^*), and concurrent execution ($\delta_1 \| \delta_2$). Intuitively, $\pi x.\delta(x)$ nondeterministically picks a binding for the variable x and performs the program δ for this binding of x.

The semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates [3]: *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining; and *(ii)* $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ we use are as in [3], except that the test construct $\Phi?$ does not yield any transition, but is final when satisfied. The predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has s' as its legal terminating situation. It is defined as $Do(\delta, s, s') \doteq \exists \delta'.Trans^*(\delta, s, \delta', s') \land Final(\delta', s')$ where *Trans** denotes the reflexive transitive closure of *Trans*. We use $\mathcal{C}$ to denote the axioms defining ConGolog.

Following [8], we say that a ConGolog program δ is *situation-determined* (SD) in s if for every sequence of transitions, the remaining program is determined by the resulting situation, i.e.

$$SituationDetermined(\delta, s) \doteq$$
$$\forall s', \delta', \delta''. Trans^*(\delta, s, \delta', s') \land Trans^*(\delta, s, \delta'', s') \supset \delta' = \delta''.$$

Example. Our running example involves a simple rescue robot Rob that is designed to aid first responders. Initially Rob is at the $Station$ but as an emergency at location $L1$ exists, Rob is expected to go to $L1$ and assist in the rescue operations (by removing rubble or by evacuating people). Action $go_{LL}(r, l)$ takes robot r to location l, and is executable if r is not already at that location. Action $removeRubble(r, l)$ (resp. $evacuate(r, l)$) can be performed by robot r at location l to remove rubble (resp. evacuate people); these actions are executable if r is at location l. Fluent $At_{LL}(r, l, s)$ indicates r's location to be l at situation s. Fluents $Cleared(r, l, s)$ and $Evacuated(r, l, s)$ evaluate to true when the robot r has removed rubble and evacuated people at location l respectively. Robot r is also able to update the software packages it uses by performing action $updateSW_{LL}(r, v)$, where v indicates the version of the software. Fluent

$UpdatedSW_{LL}(r, v, s)$ indicates if software has been updated to version v. Initially, we assume a new version $V2021$ is available.

The BAT for this domain $\mathcal{D}_l^{ex}$ includes the following action precondition axioms (throughout, we assume that free variables are universally quantified from the outside):

$$Poss(go_{LL}(r, l), s) \equiv \neg At_{LL}(r, l, s),$$
$$Poss(updateSW_{LL}(r, v), s) \equiv \neg UpdatedSW_{LL}(r, v, s),$$
$$Poss(removeRubble(r, l), s) \equiv At_{LL}(r, l, s),$$
$$Poss(evacuate(r, l), s) \equiv At_{LL}(r, l, s).$$

Moreover, $\mathcal{D}_l^{ex}$ includes the following SSAs:

$$UpdatedSW_{LL}(r, v, do(a, s)) \equiv$$
$$a = updateSW_{LL}(r, v) \vee UpdatedSW_{LL}(r, v, s),$$
$$At_{LL}(r, l, do(a, s)) \equiv a = go_{LL}(r, l) \vee$$
$$(At_{LL}(r, l, s) \wedge \neg \exists l'.\ l' \neq l \wedge a = go_{LL}(r, l')),$$
$$Cleared(r, l, do(a, s)) \equiv$$
$$a = removeRubble(r, l) \vee Cleared(r, l, s),$$
$$Evacuated(r, l, do(a, s)) \equiv$$
$$a = evacuate(r, l) \vee Evacuated(r, l, s).$$

Thus, e.g., r will be located at l in $do(a, s)$ iff a refers to r going to l, or r was already at l in s and a is not the action of r going to a different location l'.

$\mathcal{D}_l^{ex}$ also includes the following initial state axioms:

$$At_{LL}(Rob, Station, S_0), \neg Evacuated(Rob, L1, S_0),$$
$$\neg Cleared(Rob, L1, S_0), \neg UpdatedSW_{LL}(Rob, V2021, S_0).$$

3. Theoretical Foundations

3.1. Actual Cause

Given a trace of events, *actual achievement causes* are the events that are behind achieving an effect. In this section, we review previous work on achievement causality in the SC [6]. An effect here is a SC formula $\Phi[s]$ that is *uniform in s* and that may include quantifiers over object variables. Given an effect Φ, the actual causes are defined relative to a *causal setting* that includes a BAT $\mathcal{D}$ representing the domain dynamics, and a ground situation σ, representing the "narrative" (i.e. trace of events) where the effect was observed.

Definition 1 (Causal Setting). *A causal setting is a tuple $\langle \mathcal{D}, \sigma, \Phi[s] \rangle$, where $\mathcal{D}$ is a BAT, σ is a ground situation term of the form $do([\alpha_1, \cdots, \alpha_n], S_0)$ with ground action functions $\alpha_1, \cdots, \alpha_n$ such that $\mathcal{D} \models Executable(\sigma)$, and $\Phi[s]$ is a SC formula uniform in s such that $\mathcal{D} \models \neg\Phi[S_0] \wedge \Phi[\sigma]$.*

Since the theory $\mathcal{D}$ does not change, when referring to a causal setting we will often suppress $\mathcal{D}$ and simply write $\langle \sigma, \Phi \rangle$. Also, here Φ is required to hold by the end of the narrative σ, and thus we ignore the cases where Φ is not achieved by the actions in σ, since in that case, the achievement cause truly does not exist.

As all changes in the SC result from actions, the achievement causes of an effect are contained within a set of ground action terms occurring in σ. However, since σ might include multiple occurrences of the same action, one also needs to identify the situations where those actions were executed.

According to [6], if some action α of the action sequence in σ triggers the formula Φ to change its truth value from false to true relative to $\mathcal{D}$, and if there are no actions in σ after α that change the value of Φ back to false, then α is an actual cause of achieving Φ in σ. They showed that using the single-step regression operator ρ (i.e. one-step version of the regression operator defined in [5]), in addition to the primary action that actually brings about the effect of interest, one can recursively compute the chain of actions that build up to the primary achivement cause. The following inductive definition formalizes this intuition. Let $\Pi_{apa}(\alpha, \sigma)$ be the r.h.s. of the precondition axiom for α in σ.

Definition 2 (Achievement Cause). *A causal setting $\mathcal{C} = \langle \mathcal{D}, \sigma, \Phi[s] \rangle$ satisfies the achievement condition of Φ via the situation term $do(\alpha^*, \sigma^*) \sqsubseteq \sigma$ iff there is an action α' and situation σ' such that*

$$\mathcal{D} \models \neg\Phi[\sigma'] \wedge \forall s.\ do(\alpha', \sigma') \sqsubseteq s \sqsubseteq \sigma \supset \Phi[s],$$

and either $\alpha^* = \alpha'$ and $\sigma^* = \sigma'$, or the causal setting $\langle \sigma', \rho[\Phi[s], \alpha'] \wedge \Pi_{apa}(\alpha', \sigma') \rangle$ satisfies the achievement condition via the situation term $do(\alpha^*, \sigma^*)$. Whenever a causal setting $\mathcal{C}$ satisfies the achievement condition via situation $do(\alpha^*, \sigma^*)$, the action α^* executed in situation σ^* is said to be an achievement cause *of $\mathcal{C}$.*

According to [6], the achievement causes of $\mathcal{C}$ form a finite sequence of situation-action pairs, which is called the *achievement causal chain of $\mathcal{C}$.*

Example (Cont'd). Consider causal setting $\mathcal{C}_{ex} = \langle \mathcal{D}_l^{ex}, \sigma_{ex1}, \Phi_{ex1} \rangle$, where $\Phi_{ex1} = \exists r, l.\ Cleared(r, l)$ and $\sigma_{ex1} = do([updateSW_{LL}(Rob, V2021), go_{LL}(Rob, L1), removeRubble(Rob, L1)], S_0)$. Then by Definition 2, the action $removeRubble(Rob, L1)$ performed in situation $S_2 = do([updateSW_{LL}(Rob, V2021), go_{LL}(Rob, L1)], S_0)$ is an achievement cause of $\mathcal{C}_{ex}$. This is the case since $removeRubble(Rob, L1)$ is the first action after which the effect Φ_{ex1} becomes true and it remained true till the end of the trace. Moreover, we can show that $go_{LL}(Rob, L1)$ executed in $S_1 = do(updateSW_{LL}(Rob, V2021), S_0)$ is another achievement cause of $\mathcal{C}_{ex}$, since the causal setting $\langle \mathcal{D}_l^{ex}, \Phi', S_2 \rangle$ satisfies the achievement condition Φ' via the situation term $do(go_{LL}(Rob, L1), S_1)$, where $\Phi' = \rho[\ \Phi_{ex1},\ removeRubble(Rob,\ L1)] \ \wedge$

$\Pi_{apa}(removeRubble(Rob, L1), S_2)$. Finally, these are all the causes, and in particular $updateSW_{LL}$ $(Rob, V2021)$ executed in S_0 is not an achievement cause of $\mathcal{C}_{ex}$.

3.2. Abstraction

We will use the abstraction framework of [7] for reasoning about abstract causes. In this framework, there is a high-level (HL) or abstract action theory $\mathcal{D}_h$ and a low-level (LL) or concrete action theory $\mathcal{D}_l$ representing the dynamics of the domain at different levels of detail. $\mathcal{D}_h$ (resp. $\mathcal{D}_l$) involves a finite set of primitive action types $\mathcal{A}_h$ (resp. $\mathcal{A}_l$) and a finite set of primitive fluent predicates $\mathcal{F}_h$ (resp. $\mathcal{F}_l$). Also, $\mathcal{D}_h$ and $\mathcal{D}_l$ are assumed to share no domain specific symbols except for standard names for objects in $\mathcal{N}$.

Definition 3 (Refinement Mapping). *A* refinement mapping m *is a function that associates each HL primitive action type A in $\mathcal{A}_h$ to a SD ConGolog program δ_A defined over the LL theory that implements the action, i.e. $m(A(\vec{x})) = \delta_A(\vec{x})$. Moreover, m maps each situation-suppressed HL fluent $F(\vec{x})$ in $\mathcal{F}_h$ to a situation-suppressed formula $\Phi_F(\vec{x})$ defined over the LL theory that characterizes the concrete conditions under which $F(\vec{x})$ holds in a situation.*

We extend the notation so that $m(\Phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula Φ by $m(F(\vec{x}))$. Also, we apply m to action sequences with $m(\alpha_1, \ldots, \alpha_n) \doteq m(\alpha_1); \ldots; m(\alpha_n)$ for $n \geq 1$ and $m(\epsilon) \doteq nil$, where ϵ is the empty sequence of actions.

To relate the HL and LL models/theories, a variant of bisimulation [9] is defined as follows.

Definition 4 (m-Bisimulation). *Given M_h a model of $\mathcal{D}_h$, and M_l a model of $\mathcal{D}_l \cup \mathcal{C}$, a relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ (where Δ_S^M stands for the situation domain of M) is an* m-bisimulation relation between M_h and M_l *if $\langle s_h, s_l \rangle \in B$ implies that: (i) $s_h \sim_m^{M_h, M_l} s_l$, i.e. s_h evaluates each HL primitive fluent same as the evaluation of the refinement of the fluent in s_l; (ii) for every HL primitive action type A in $\mathcal{A}_h$, if there exists s_h' s.t. $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$, then there exists s_l' s.t. $M_l \models Do(m(A(\vec{x})), s_l, s_l')$ and $\langle s_h', s_l' \rangle \in B$; and (iii) for every HL primitive action type A in $\mathcal{A}_h$, if there exists s_l' s.t. $M_l \models Do(m(A(\vec{x})), s_l, s_l')$, then there exists s_h' s.t. $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$ and $\langle s_h', s_l' \rangle \in B$.*

M_h is *m-bisimilar* to M_l, written $M_h \sim_m M_l$, iff there exists an m-bisimulation relation B between M_h and M_l such that $(S_0^{M_h}, S_0^{M_l}) \in B$.

Definition 5 (Sound abstraction). *$\mathcal{D}_h$ is a* sound abstraction of $\mathcal{D}_l$ relative to refinement mapping m *iff for all models M_l of $\mathcal{D}_l \cup \mathcal{C}$, there exists a model M_h of $\mathcal{D}_h$ s.t. $M_h \sim_m M_l$.*

With a sound abstraction, if the HL theory *entails* that a sequence of actions is executable and achieves a condition, then the LL must also entail that there exists an executable refinement of the sequence such that the "translated" condition holds afterwards. Also, if the LL considers the executability of a refinement of a sequence of HL actions is satisfiable after which a refinement of a condition holds, then the HL also considers executability of the sequence of HL action satisfiable after which the condition holds as well.

Definition 6 (Complete abstraction). *$\mathcal{D}_h$ is a* complete abstraction of $\mathcal{D}_l$ relative to refinement mapping m *iff for all models M_h of $\mathcal{D}_h$, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ s.t. $M_l \sim_m M_h$.*

With a complete abstraction, if the LL theory entails that some refinement of a sequence of HL actions is executable and achieves a "translated" HL condition, then the HL also *entails* that the action sequence is executable and the condition holds afterwards. Also, if the HL considers the executability of sequence of actions is satisfiable after which a condition holds, then the LL also considers executability of the refinement of the sequence of HL action satisfiable after which a "translated" condition holds as well.

Note that $\mathcal{D}_h$ is a *sound and complete abstraction of $\mathcal{D}_l$ relative to refinement mapping* m iff $\mathcal{D}_h$ is both a sound and a complete abstraction of $\mathcal{D}_l$ relative to m. Also, this approach supports the use of ConGolog programs to specify the possible dynamics of the domain at both the HL and LL; this is done by following [10] and "compiling" the program into the BAT $\mathcal{D}$ to get a new BAT $\mathcal{D}'$ whose executable situations are exactly those that can be reached by executing the program.

Example (Cont'd). In our example, we define a HL BAT $\mathcal{D}_h^{ex}$ that abstracts over some details of $\mathcal{D}_l^{ex}$. At the HL, we abstract over details of rescue actions. Action $rescue(r, l)$ abstracts over the process of either clearing rubble or evacuating people. The fluent $AidedInRescue(r, l, s)$ indicates if robot r has aided in rescue at location l. For simplicity, actions $updateSW_{HL}(r, v)$ and $go_{HL}(r, l)$ are defined similar to $updateSW_{LL}(r, v)$ and $go_{LL}(r, l)$ respectively.

$\mathcal{D}_h^{ex}$ includes the following precondition axioms:

$$Poss(updateSW_{HL}(r, v), s) \equiv \neg UpdatedSW_{HL}(r, v, s),$$
$$Poss(go_{HL}(r, l), s) \equiv \neg At_{HL}(r, l, s),$$
$$Poss(rescue(r, l), s) \equiv At_{HL}(r, l, s).$$

The HL BAT also includes the following SSAs:

$$AidedInRescue(r, l, do(a, s)) \equiv$$
$$a = rescue(r, l) \vee AidedInRescue(r, l, s).$$

At_{HL} and $UpdatedSW_{HL}$ have SSAs similar to their LL counterparts respectively.

$\mathcal{D}_h^{ex}$ contains the following initial state axioms:

$$At_{HL}(Rob, Station, S_0),$$
$$\neg UpdatedSW_{HL}(Rob, V2021, S_0),$$
$$\neg AidedInRescue(Rob, L1, S_0).$$

Refinement Mapping m^{eg} We specify the relationship between the HL and LL BATs through a refinement mapping m^{eg} which is defined as follows:

$$m^{eg}(go_{HL}(r, l)) = go_{LL}(r, l),$$
$$m^{eg}(updateSW_{HL}(r, v)) = updateSW_{LL}(r, v),$$
$$m^{eg}(rescue(r, l)) =$$
$$evacuate(r, l) \mid removeRubble(r, l),$$
$$m^{eg}(At_{HL}(r, l)) = At_{LL}(r, l),$$
$$m^{eg}(UpdatedSW_{HL}(r, v)) = UpdatedSW_{LL}(r, v),$$
$$m^{eg}(AidedInRescue(r, l)) =$$
$$Cleared(r, l) \vee Evacuated(r, l).$$

By using Theorem 9 in [7], it is straightforward to confirm that $\mathcal{D}_h^{ex}$ is a sound abstraction of $\mathcal{D}_l^{ex}$ relative to the mapping m^{eg}.

4. Programs as Actual Causes

We now return to our discussion of abstract causes. As seen in Section 3, Definition 2 appeals to regression, a syntactic notion, and this requires the effect formula $\Phi[s]$ to be uniform in s. However, this is too restrictive for us as it is hard to adapt for abstract causes. Specifically, it is hard to define regression over programs; recall Reiter defined regression over primitive actions.[1] Therefore, we start by introducing the notion of *dynamic effect formulae* in the SC.

Definition 7 (Dynamic Effect Formula). *Let $\vec{x}$ and $\theta_{\vec{a}}$ respectively range over object terms and a sequence of action terms. The class of situation-suppressed dynamic effect formulae ψ is defined inductively using the following grammar:*

$$\psi ::= P(\vec{x}) \mid ExecSeq(\theta_{\vec{a}}) \mid After(\theta_{\vec{a}}, \psi) \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists \vec{x}.\psi.$$

That is, a dynamic effect formula can be a situation-suppressed fluent, a formula that says that some sequence of actions $\theta_{\vec{a}}$ is executable, a formula that indicates some dynamic effect formula holds after some sequence of actions has occurred, or one that can be built from other dynamic effect formulae using the usual connectives. Note that ψ can have quantification over object variables, but must not include quantification over situations or the precedence operator $\sqsubset$. We use lower-case Greek letters for dynamic effect formulae. $\psi[s]$ is the formula obtained from ψ by restoring the appropriate situation argument into all fluents in ψ.

Definition 8.

$$\psi[s] \stackrel{\text{def}}{=} \begin{cases} P(\vec{x}, s) & \text{if } \psi \text{ is } P(\vec{x}) \\ \exists s'.\ Do(\theta_{\vec{a}}, s, s') & \text{if } \psi \text{ is } ExecSeq(\theta_{\vec{a}}) \\ \psi'[do([\theta_{\vec{a}}], s)] & \text{if } \psi \text{ is } After(\theta_{\vec{a}}, \psi') \\ \neg(\psi'[s]) & \text{if } \psi \text{ is } (\neg\psi') \\ \psi_1[s] \wedge \psi_2[s] & \text{if } \psi \text{ is } (\psi_1 \wedge \psi_2) \\ \exists \vec{y}.\ (\psi'[s]) & \text{if } \psi \text{ is } (\exists \vec{y}.\ \psi') \end{cases}$$

We generalize causal settings by allowing effects in our framework to be any dynamic effect formula ψ, i.e. we no longer require the effect to be uniform in s. Also, we do not require the trace to be a ground situation term, so it can now include arbitrary (non-ground) action terms. This allows for the modeling of abstract causes.

Definition 9 (Generalized Causal Setting). *A generalized causal setting is a tuple $\langle \mathcal{D}, \delta, \psi \rangle$, where $\mathcal{D}$ is a BAT, δ is a ConGolog program, and ψ is a dynamic effect formula s.t.:*

$$\mathcal{D} \cup \mathcal{C} \models \neg\psi[S_0] \wedge \exists s'.\ Do(\delta, S_0, s') \wedge \psi[s'].$$

Thus, there is at least one execution of the program δ starting in the initial situation S_0 after which the effect ψ holds.

As discussed in Section 3, the definition of actual achievement cause given by [6] only deals with narratives that are linear sequences of actions. Consequently, their causes are actions (or more precisely, action-situation pairs).[2] To facilitate the modeling of abstract causes, we extend this by allowing narratives to be linear sequences of ConGolog programs. This allows programs to be identified as causes of observed effects. In the following, we progressively define what it means for a ConGolog program to be a weak potential cause, starting with primary causes. Note that, given a generalized causal setting there can be more than one primary potential cause of the effect as the program can have multiple possible executions.

Definition 10. *Given a generalized causal setting $C = \langle \mathcal{D}, (\delta_1; \cdots; \delta_n), \psi \rangle$ and a model M of $\mathcal{D} \cup \mathcal{C}$, a program $\delta_{i+1} \in \{\delta_1, \cdots, \delta_n\}$ is called a primary weak potential cause of ψ relative to C and M if and only if:*

$$M \models \exists s_i, s_{i+1}, s_n.\ Do((\delta_1; \ldots; \delta_i), S_0, s_i) \wedge \neg\psi[s_i]$$
$$\wedge\ Do(\delta_{i+1}, s_i, s_{i+1}) \wedge Do((\delta_{i+2}; \ldots; \delta_n), s_{i+1}, s_n)$$
$$\wedge\ \forall s'.\ s_{i+1} \leq s' \leq s_n \supset \psi[s'].$$

The triple (s_i, s_{i+1}, ψ) is called a witness for this.

[1] Note that, previously [11] has proposed an extension of regression for programs; investigating whether their definition can be adapted for our purpose is future work.

[2] Here and in the sequel, for brevity, we omit the terms *actual* and *achievement* when we talk about causes, since we exclusively consider actual achievement causes in this paper.

That is, a program δ_{i+1} in the scenario $(\delta_1; \ldots; \delta_n)$ is a *primary weak potential cause relative to a model* M *of theory* $\mathcal{D} \cup \mathcal{C}$ *and causal setting* C if and only if there is an execution of the prefix $(\delta_1; \ldots; \delta_i)$ that ends in situation s_i in which ψ is false, situation s_{i+1} can be reached by executing δ_{i+1} starting from s_i, situation s_n can be reached by executing the remaining programs starting from s_{i+1}, and ψ holds in all situations from s_{i+1} up to s_n. Essentially, this is a straightforward generalization of the base case of Definition 2 and ensures that there is an execution of the scenario over which ψ was achieved by some action in δ_{i+1} and ψ persisted till the end of the trace, i.e. it was not later made false by a subsequent action.

Moreover, we define what it means for a program to be a primary weak potential cause relative to a causal setting.

Definition 11 (Primary Weak Potential Cause). *Given a generalized causal setting* $C = \langle \mathcal{D}, (\delta_1; \cdots; \delta_n), \psi \rangle$, *a program* $\delta_i \in \{\delta_1, \cdots, \delta_n\}$ *is called a* primary weak potential cause relative to C *if and only if for all models* M *of* $\mathcal{D} \cup \mathcal{C}$, δ_i *is a primary weak potential cause of* ψ *relative to* C *and* M.

Next, we define weak potential causes in general. These include both primary and non-primary causes reflecting both base and inductive cases of Definition 2.

Definition 12. *Given a generalized causal setting* $C = \langle \mathcal{D}, (\delta_1; \cdots; \delta_n), \psi \rangle$ *and a model* M *of* $\mathcal{D} \cup \mathcal{C}$, *a program* $\delta_i \in \{\delta_1, \cdots, \delta_n\}$ *is called a weak potential cause of* ψ *relative to* C *and* M *if and only if:*

1. δ_i *is a primary weak potential cause wrt* C *and* M *with witness* (s', s'', ψ'), *where* $\psi' = \psi$, *or*
2. δ_j *(where* $i < j \leq n$*) is a weak potential cause relative to setting* C *and* M *with witness* $(s_{j-1}, do([\vec{a_j}], s_{j-1}), \psi_j)$, *and* δ_i *is a primary weak potential cause relative to the setting* $\langle \mathcal{D}, (\delta_1; \cdots; \delta_{j-1}), \psi' \rangle$ *and model* M *with witness* (s', s'', ψ'), *where* $\psi' = ExecSeq(\vec{a_j}) \wedge After(\vec{a_j}, \psi_j)$.

We call (s', s'', ψ') *a witness for* δ_i *being a weak potential cause wrt* C *and* M.

Thus, δ_i is a *weak potential cause relative to model* M *and generalized causal setting* C if and only if it is either a primary weak potential cause wrt C and M, or it is a primary weak potential cause of another weak potential cause δ_j, i.e. it enables δ_j by ensuring that the appropriate execution path $\vec{a_j}$ of δ_j that brought about δ_j's own effect ψ_j is executable (i.e. that $ExecSeq(\vec{a_j})$) and by fulfilling the conditions under which the execution of $\vec{a_j}$ achieved ψ_j (i.e. that $After(\vec{a_j}, \psi_j)$).

Definition 13 (Weak Potential Cause). *Given a generalized causal setting* $C = \langle \mathcal{D}, (\delta_1; \cdots; \delta_n), \psi \rangle$, *a program* $\delta_i \in \{\delta_1, \cdots, \delta_n\}$ *is called a weak potential cause of* ψ *relative to* C *if and only if for all models* M *of* $\mathcal{D} \cup \mathcal{C}$, δ_i *is a weak potential cause of* ψ *relative to* C *and* M.

Moreover, if $\mathcal{D}$ *is initially completely specified, there is only one model; in that case, we call* ψ' *from the witness* (s', s'', ψ') *in Definition 12 a witness to the fact that* δ_i *is a weak potential cause of* ψ *relative to* C.

Thus, we only call a program a *weak potential cause relative to a generalized causal setting* if it is a weak potential cause in all models of the theory.

Example (Cont'd). Consider the setting $\langle \mathcal{D}_l^{ex}, (updateSW_{LL}(Rob, V2021); go_{LL}(Rob, L1); \delta_{rescue}(Rob, L1)), \exists r, l.Cleared(r, l) \rangle$, where $\delta_{rescue}(r, l) = m^{eg}(rescue(r, l))$. Then according to our definitions, $\delta_{rescue}(Rob, L1)$ is the primary weak potential cause relative to the above setting, as in all models, $\mathcal{D}_l^{ex} \cup \mathcal{C} \models \exists s_2, s_3. Do((updateSW_{LL}(Rob, V2021); go_{LL}(Rob, L1)), S_0, s_2) \wedge \neg\exists r, l.Cleared(r, l)[s_2] \wedge Do(\delta_{rescue}(Rob, L1), s_2, s_3) \wedge \exists r, l.Cleared(r, l)[s_3]$, and by the SSA for $Cleared$, the effect persists until the end of scenario.

Note that, as only in some executions of the scenario $\exists r, l.Cleared(r, l)$ is true, $\delta_{rescue}(Rob, L1)$ is considered *weak*. If we instead consider the effect $\exists r, l.Cleared(r, l) \vee Evacuated(r, l)$, then $\delta_{rescue}(Rob, L1)$ can be considered as the primary strong potential cause in the sense that in all executions of the scenario δ_{rescue} achieves the effect.

Moreover, we can also show that $go_{LL}(Rob, L1)$ is a weak potential cause, since it is a primary weak potential cause wrt the setting $\langle \mathcal{D}_l^{ex}, (updateSW_{LL}(Rob, V2021); go_{LL}(Rob, L1)), ExecSeq(removeRubble(Rob, L1)) \wedge After(removeRubble(Rob, L1), \exists r, l.Cleared(r, l)) \rangle$.

On the other hand, $updateSW_{LL}(Rob, V2021)$ cannot be shown to be a weak potential cause.

Our notion of programs as actual cause above is a weak and more inclusive one. We consider a program as a cause if there is at least one execution where the program is a cause. In some cases, it might be useful to consider a stronger version, where a program is considered to be a cause if it is a cause according to all executions of the program. A thorough investigation of such a variant is future work.

When the program δ is finite, terminating, and composed of ground actions only, one can show that the intermediate effects (i.e. $[ExecSeq(\vec{a}) \wedge After(\vec{a}, \phi)]$) can be straightforwardly computed using Reiter's regression. Also, and in particular, when δ is a finite sequence of ground actions, the causes computed using our definition and the definition in [6] are the same.

Theorem 14. *Let $\delta = \alpha_1; \cdots ; \alpha_n$ be a finite sequence of ground actions. Then $(\alpha_i, do([\alpha_1, \cdots, \alpha_{i-1}], S_0))$ is a cause relative to the causal setting $\langle \mathcal{D}, do([\delta], S_0), \Phi \rangle$ according to Definition 2 iff α_i is a potential cause relative to the generalized causal setting $\langle \mathcal{D}, \delta, \Phi \rangle$ according to Definition 13.*[3]

Given the above, it is easy to see that when δ is a finite sequence of ground actions, all properties shown for [6]'s framework also hold in ours. These include the proper handling of preemption and switches.

5. Reasoning about Abstract Causes

We now focus on investigating how reasoning about abstract causes can be simplified. In particular, we will show that under some conditions, a subclass of weak potential causes at various levels of abstraction can be related. This subclass involves weak causes that are also strong in the sense that all executions of the cause achieve the effect (see the corollary below). This reduces reasoning about abstract causes (i.e. programs) at the LL to that of actions as causes at the HL when said conditions are met.

We start by formalizing some of these conditions. First, we assume that every HL action α_i is mapped via m to a LL program δ_i that may take part in a LL scenario; in this way, any abstract scenario can be refined by a concrete one.

Moreover, we assume only action sequences that refine some HL action sequence are executed in the LL BAT:

Assumption 1 (LL behaviours refine HL actions).

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s. Executable(s) \supset \exists \delta. Trans^*(\textsc{anyseqhl}, S_0, \delta, s),$$

where $\textsc{anyseqhl} \overset{\text{def}}{=} (|_{A_i \in \mathcal{A}_h} \ \pi \vec{x}. m(A_i(\vec{x})))^*$,

i.e. do any sequence of refinements of HL actions.

Furthermore, we require that LL effects are non-transient wrt HL actions:

Assumption 2 (Non-transiency of LL Effects).
Suppose the set $F_R^{F_i}$ includes all the fluent literals in a refinement of a HL fluent F_i. We assume that:

$$\begin{aligned}&\mathcal{D}_l \cup \mathcal{C} \models \forall s. Do(\textsc{anyseqhl}, S_0, s) \supset \\ &\quad \textstyle\bigwedge_{A_i \in \mathcal{A}_h} \bigwedge_{F_i \in \mathcal{F}_h} \bigwedge_{F_L \in F_R^{F_i}} \forall s', s'', \vec{x}, \vec{y}. \\ &\quad F_L(\vec{y})[s] \wedge Do(m(A_i(\vec{x})), s, s') \wedge F_L(\vec{y})[s'] \\ &\quad \wedge s < s'' < s' \supset F_L(\vec{y})[s'']\end{aligned}$$

The above essentially requires the LL theory to entail that if a fluent literal F_L that is in a refinement of a HL fluent F_i is true in both the situations before and after execution of the refinement of a HL action $A_i(\vec{x})$, then it should remain true in all intermediate situations of execution of the refinement of $A_i(\vec{x})$ as well. This condition must hold after any sequence of refinements of HL actions, i.e. $Do(\textsc{anyseqhl}, S_0, s)$. To see why this is necessary, consider the following example. Suppose that at the HL, we have the generalized causal setting $\langle \mathcal{D}_h, (\alpha; \beta), F_{hl} \rangle$ in which α is the only primary weak potential cause. Assume the following mapping: $m(\alpha) = a$ and $m(\beta) = b_1; b_2$, and $m(F_{hl}) = F_{ll}$. At the LL, after performing a, F_{ll} becomes true, and after performing b_1 and b_2, F_{ll} becomes false and true respectively. Hence, in the setting $\langle \mathcal{D}_l, m(\alpha; \beta), m(F_{hl}) \rangle$, $m(\beta)$ is considered the only primary weak potential cause if the analysis is done at the LL using Definition 13. To achieve correspondence of potential causes between HL and LL, we need to rule out such cases.

To investigate how causes at the abstract and concrete levels are related, we first consider *sound abstractions.* For this, and when complete information is assumed, we can show that if an action α is a weak potential cause wrt the generalized casual setting $C_h = \langle \mathcal{D}_h, (\vec{\alpha_h}), \Phi_H \rangle$ at the HL, the refinement of α can be considered a weak potential cause wrt the setting $C_m = \langle \mathcal{D}_l, m(\vec{\alpha_h}), m(\Phi_H) \rangle$ at the LL.[4]

Theorem 15. *Suppose that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt some refinement mapping m, and that Assumptions 1 and 2 hold. Then for any ground HL action sequence $\vec{\alpha}$ and for any HL situation suppressed formula Φ such that $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \neg\Phi[S_0] \wedge \Phi[do([\vec{\alpha}], S_0)]$, we have:*

1. $\mathcal{D}_l \cup \mathcal{C} \models \neg m(\Phi)[S_0] \wedge \exists s.\ Do(m(\vec{\alpha}), S_0, s) \wedge m(\Phi)[s]$.
2. *If $\vec{\alpha} = \vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}$ and α_k is the primary weak potential cause wrt the generalized causal setting $C_h = \langle \mathcal{D}_h, (\vec{\alpha}), \Phi \rangle$, then $m(\alpha_k)$ is the unique primary weak potential cause wrt the generalized causal setting $C_m = \langle \mathcal{D}_l, m(\vec{\alpha}), m(\Phi) \rangle$.*
3. *If $\vec{\alpha} = \vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}\alpha_j\vec{\alpha}_{j+1}$, α_j is a weak potential cause wrt the generalized causal setting $C_h = \langle \mathcal{D}_h, (\vec{\alpha}), \Phi \rangle$ with witness Φ_j, and α_k is the primary weak potential cause wrt the setting $C'_h = \langle \mathcal{D}_h, (\vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}), ExecSeq(\alpha_j) \wedge After(\alpha_j, \Phi_j) \rangle$, and moreover, $\mathcal{D}_l$ is initially completely specified, and $m(\alpha_j)$ is a weak potential cause wrt the causal setting $C_m = \langle \mathcal{D}_l, m(\vec{\alpha}), m(\Phi) \rangle$ with witness $m(\Phi_j)$ then, $m(\alpha_k)$ is the unique primary weak potential cause wrt the generalized causal setting $C'_m = \langle \mathcal{D}_l, m(\vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}), \phi'_L \rangle$, where $\mathcal{D}_l \cup \mathcal{C} \models \exists s^*, \vec{a_j}.\ Do(m(\vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}), S_0, s^*) \wedge$*

[3] Note that, a SC formula Φ is also a dynamic effect formula.

[4] In the following, we will quantify over action sequences and so we need to encode sequences as first-order terms as in [3]. For notational simplicity, we suppress this encoding and use sequences as terms directly. Also, we conjecture that these results also hold when the initial state is incomplete; proving this is future work.

$Do(m(\alpha_j), s^*, do([\vec{a_j}], s^*))$ *and* $\phi'_L = ExecSeq(\vec{a_j}) \wedge After(\vec{a_j}, m(\Phi_j))$.

Example (Cont'd). Consider the HL setting $C_h = \langle \mathcal{D}_h^{ex}, (\vec{\alpha}), \phi_e \rangle$, where $\vec{\alpha} = [updateSW_{HL}(Rob, V2021), go_{HL}(Rob, L1), rescue(Rob, L1)]$ and $\phi_e = \exists r, l. AidedInRescue(r, l)$. Using similar reasoning as before, we can show that $rescue(Rob, L1)$ is the primary weak potential cause relative to C_h. Moreover, $go_{HL}(Rob, L1)$ is another cause relative to C_h.

By Theorem 15, we have that $m^{eg}(rescue(Rob, L1)) = \delta_{rescue}(Rob, L1)$ is the primary weak potential cause relative to setting $C_m = \langle \mathcal{D}_l^{ex}, m^{ex}(\vec{\alpha}), m^{ex}(\phi_e) \rangle$, where $m^{ex}(\phi_e) = \exists r, l. Cleared(r, l) \vee Evacuated(r, l)$ and $m^{ex}(\vec{\alpha}) = updateSW_{LL}(Rob, V2021); go_{LL}(Rob, L1); \delta_{rescue}(Rob, L1)$. Moreover, the action $go_{LL}(Rob, L1)$ is considered another weak potential cause relative to C_m.

Notice that since the number of actions and fluents that a reasoner needs to consider are typically higher at the LL, Theorem 15 can yield important efficiency benefits.

In Corollary 5 of [7] ([BDL17]), the authors showed that if $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, then the different sequences of LL actions that are refinements of a given HL action sequence all have the same effects on the HL fluents, and more generally on HL situation-suppressed formulae, i.e. from the HL perspective they are deterministic:

Corollary 16 (from BDL17). *If $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, then for any sequence of ground HL actions $\vec{\alpha}$ and for any HL situation-suppressed formula ϕ, we have:*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'. Do(m(\vec{\alpha}), S_0, s) \wedge Do(m(\vec{\alpha}), S_0, s') \supset (m(\phi)[s] \equiv m(\phi)[s']).$$

This indicates that the weak potential causes in Theorem 15 are in fact strong in the sense that in all executions of the program, the effect is achieved.

With *complete abstractions*, and when complete information is assumed, we can show that if the refinement of a HL action α is a weak potential cause wrt the LL setting $\langle \mathcal{D}_l, m(\vec{\alpha_h}), m(\Phi_H) \rangle$, then α can be considered a weak potential cause wrt the setting $\langle \mathcal{D}_h, (\vec{\alpha_h}), \Phi_H \rangle$ at the HL.

Theorem 17. *Suppose that $\mathcal{D}_h$ is a complete abstraction of $\mathcal{D}_l$ wrt some refinement mapping m. Then for any ground HL action sequence $\vec{\alpha}$ and for any HL situation suppressed formula Φ such that $\mathcal{D}_l \cup \mathcal{C} \models \neg m(\Phi)[S_0] \wedge \exists s.\ Do(m(\vec{\alpha}), S_0, s)\ \wedge m(\Phi)[s]$, we have that:*

1. $\mathcal{D}_h \models \neg\Phi[S_0] \wedge Executable(do([\vec{\alpha}], S_0)) \wedge \Phi[do([\vec{\alpha}], S_0)]$.
2. *If $\vec{\alpha} = \vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}$ and $m(\alpha_k)$ is the primary weak potential cause wrt the generalized causal setting $C_m = \langle \mathcal{D}_l, m(\vec{\alpha}), m(\Phi) \rangle$ then α_k is the unique primary weak potential cause wrt the setting $C_h = \langle \mathcal{D}_h, (\vec{\alpha}), S_0), \Phi \rangle$.*
3. *If $\mathcal{D}_l$ is initially completely specified, $\vec{\alpha} = \vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}\alpha_j\vec{\alpha}_{j+1}$, $m(\alpha_j)$ is a weak potential cause wrt the generalized causal setting $C_m = \langle \mathcal{D}_l, m(\vec{\alpha}), m(\Phi) \rangle$ with witness $m(\Phi_j)$, and $m(\alpha_k)$ is the unique primary weak potential cause wrt the generalized causal setting $C'_m = \langle \mathcal{D}_l, m(\vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}), \phi'_L \rangle$, where $\mathcal{D}_l \cup \mathcal{C} \models \exists s^*, \vec{a_j}.\ Do(m(\vec{\alpha}_{k-1}\alpha_k\vec{\alpha}_{k+1}), S_0, s^*) \wedge Do(m(\alpha_j), s^*, do([\vec{a_j}], s^*)))$ and $\phi'_L = ExecSeq(\vec{a_j}) \wedge After(\vec{a_j}, m(\Phi_j))$, and moreover, α_j is a weak potential cause wrt the causal setting $C_h = \langle \mathcal{D}_h, \vec{\alpha}, \Phi \rangle$, then, α_k is the unique primary weak potential cause wrt the setting $C'_h = \langle \mathcal{D}_h, (\vec{\alpha}_{k-1}\alpha_k \vec{\alpha}_{k+1}), ExecSeq(\alpha_j) \wedge After(\alpha_j, \Phi_j) \rangle$.*

Example (Cont'd). Let $\vec{\alpha} = [updateSW_{HL}(Rob, V2021), go_{HL}(Rob, L1), rescue(Rob, L1)]$ and $\phi_e = \exists r, l.\ AidedInRescue(r, l)$. Suppose at the LL, $\mathcal{D}_l^{ex} \cup \mathcal{C} \models \neg m^{ex}(\phi_e)[S_0] \wedge \exists s.\ Do(m^{ex}(\vec{\alpha}), S_0, s) \wedge m^{ex}(\phi_e)[s]$. Moreover, suppose that $\delta_{rescue}(Rob, L1)$ is the primary weak potential cause wrt setting $C_m = \langle \mathcal{D}_l^{ex}, m^{ex}(\vec{\alpha}), m^{ex}(\phi_e) \rangle$, which brings about the effect $m^{ex}(\phi_e) = \exists r, l. Cleared(r, l) \vee Evacuated(r, l)$. Also, $go_{LL}(Rob, L1)$ is another cause wrt C_m.

Then by Theorem 17, we have that $rescue(Rob, L1)$ is the primary weak potential cause wrt the setting $C_h = \langle \mathcal{D}_h^{ex}, (\vec{\alpha}), \phi_e \rangle$, which brings about the effect ϕ_e. Moreover, the action $go_{HL}(Rob, L1)$ can be considered another weak potential cause wrt C_h.

Depending on requirements of the domain, a modeler can decide among sound, complete, or sound and complete abstractions, each providing efficiency benefits.

6. Discussion

Since we build on a robust approach to computing actual causes in the SC [6] we can also handle correctly tricky cases of preemption and over-determination that were problematic for the previous approaches. For example, in the well-known Switch example, a switch action is not an actual cause as shown in [6]. Moreover, as proved in [12], their own counterfactual based approach turns out to be equivalent to the first principles approach of [6]. Also, the paper [12] argued that an actual cause can be correctly determined in the difficult Bottle example without appealing to physically impossible interventions.

While there has been a lot of work on actual causation, to the best of our knowledge, our account is the first and the only proposal that investigates programs as actual causes. Perhaps the closest to our work is the one by [13], who identified a subset of actions (program steps) of a set of interacting programs as an actual cause for a violation of specific properties in a security domain. Our approach however, focuses on formalizing abstract actual causes as programs in the settings where the actions that led to the observed effect are only incompletely specified. Our framework is based on an expressive first-order logic language for representing and reasoning about dynamic domains. In addition to non-deterministic programs, we allow for incomplete information that is represented through multiple models of a BAT. Furthermore, we investigate how abstraction may be used to facilitate representation and reasoning.

In this paper, we do not study how one can obtain an abstraction given ConGolog programs. Instead, we study causal reasoning that can be accomplished if we are given a sound and/or a complete abstraction of our causal theory. [14] proposed forgetting (of LL fluent and action symbols) to obtain a sound and complete abstraction of a LL BAT for a given mapping. Also, [7] identifed the necessary and sufficient conditions for a (given) HL BAT to be a sound abstraction of a LL BAT under a mapping. For simplicity, we focused on a single layer of abstraction, but the framework supports extending the hierarchy to several levels. In future work, we plan to investigate methodologies for designing abstract theories and refinement mappings with respect to given observed effects, as well as automated synthesis techniques to support this. Extending the current framework to support probabilistic actions [15] and approximate abstractions, and how such extensions facilitate reasoning about causality are important avenues for future research.

References

[1] J. Pearl, Causality: Models, Reasoning, and Inference, Cambridge University Press, 2000.

[2] J. Y. Halpern, Actual Causality, MIT Press, 2016.

[3] G. De Giacomo, Y. Lespérance, H. J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Artif. Intell. 121 (2000) 109–169.

[4] J. McCarthy, P. J. Hayes, Some Philosophical Problems From the Standpoint of Artificial Intelligence, Machine Intelligence 4 (1969) 463–502.

[5] R. Reiter, Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems, The MIT Press, 2001.

[6] V. Batusov, M. Soutchanski, Situation calculus semantics for actual causality, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI Press, 2018, pp. 1744–1752.

[7] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction in situation calculus action theories, in: S. P. Singh, S. Markovitch (Eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI Press, 2017, pp. 1048–1055.

[8] G. De Giacomo, Y. Lespérance, C. J. Muise, On supervising agents in situation-determined ConGolog, in: International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, IFAAMAS, 2012, pp. 1031–1038.

[9] R. Milner, Communication and concurrency, PHI Series in computer science, Prentice Hall, 1989.

[10] G. De Giacomo, Y. Lespérance, F. Patrizi, S. Sardiña, Verifying ConGolog programs on bounded situation calculus theories, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press, 2016, pp. 950–956.

[11] P. Mo, N. Li, Y. Liu, Automatic verification of golog programs via predicate abstraction, in: G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), volume 285 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2016, pp. 760–768.

[12] S. M. Khan, M. Soutchanski, Necessary and sufficient conditions for actual root causes, in: ECAI 2020 - 24th European Conference on Artificial Intelligence, 2020, pp. 800–808.

[13] A. Datta, D. Garg, D. K. Kaynar, D. Sharma, A. Sinha, Program actions as actual causes: A building block for accountability, in: C. Fournet, M. W. Hicks, L. Viganò (Eds.), IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015, IEEE Computer Society, 2015, pp. 261–275. URL: https://doi.org/10.1109/CSF.2015.25. doi:10.1109/CSF.2015.25.

[14] K. Luo, Y. Liu, Y. Lespérance, Z. Lin, Agent abstraction via forgetting in the situation calculus, in: ECAI 2020 - 24th European Conference on Artificial Intelligence, volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 809–816. URL: https://doi.org/10.3233/FAIA200170. doi:10.3233/FAIA200170.

[15] V. Belle, H. J. Levesque, Regression and progression in stochastic domains, Artificial Intelligence 281 (2020) 103247. URL: https://doi.org/10.1016/j.artint.2020.103247. doi:10.1016/j.artint.2020.103247.