On the Semantics of Actual Causality in Situation Calculus Concurrent Game Structures

MohammadHossein Karimian[†], Shakil M. Khan[†,*], Yves Lespérance[‡]
[†] Department of Computer Science, University of Regina
[‡] Department of Electrical Engineering and Computer Science, York University

Abstract

Key to the formalization of rationality is the study of actual causation. Halpern and Pearl's pioneering work on causal models is based on structural-equations models, which assumes an overly simplistic model of action and change. Although much recent work within action-theoretic frameworks has appeared to deal with this, all of these accounts share a common and strong limitation, that the scenario or history of actions in these are assumed to be linear sequences of actions or traces. To deal with this, in this paper we study causation in a synchronous game-theoretic logic framework that allows concurrent moves by multiple agents. Our framework is based on situation calculus concurrent game structures. We show that our formalization has some interesting properties and handles the issues associated with preemption and over-determination well.

Keywords: Actual cause, Concurrent game structures, Situation calculus, Reasoning about action.

1. Introduction

Actual causality, also known as token-level causality, is the problem of identifying the causes of an observed effect from a given history of events or actions (also called, the scenario) [1]. Based on David Hume's original proposal, this problem has been studied extensively both from counterfactual perspectives (e.g., [1–12]) as well as from regularity approaches (e.g., [13–15]). The former involve studying causes by observing what would have happened had some of the events in the original scenario not occurred, while the latter accounts define causation from the observation that causes are regularly followed by their effects (one interpretation of this, among many, states that a cause is an insufficient but necessary part of a sufficient condition that is itself unnecessary for bringing about the observed effect [13]). Others have attempted to combine these two approaches [16, 17].

In recent years researchers have become increasingly interested in studying causation within more expressive action-theoretic frameworks, in particular in that of the situation calculus [16, 18–22]. In contrast to the popular structural equations models-based or SEM-based causal models [23], these are based on a formal theory of action, and thus incorporate important aspects such as action preconditions, effects, and frame conditions, and temporal order of action occurrence into the model, allowing one to capture, e.g., non-persistent change supported by fluents and possible dependencies between events. Moreover, this allows one to formalize causation from the perspective of individual agents by defining a notion of epistemic causation [24] and by supporting causal reasoning about conative effects, which in turn has proven useful for explaining agent behaviour using causal analysis [25] and has the potential for defining important concepts such as responsibility and blame [26].

A major limitation of these proposals, however, is that they take the scenario to be a linear sequence of single-agent actions. In multi-agent settings, this means that they are restricted to turn-taking games. To overcome these limitations, in this paper, we consider causation in *multi-agent synchronous games*. Our account is based on Situation Calculus Synchronous Game Structures (SCSGS) [27], where we have a single action *tick* whose effects

[*] shakil.khan@uregina.ca

depend on the combination of moves selected by the players. Each agent selects its move without knowing which move is selected by the other agents. As we will see, in domains with synchronous concurrency, besides the usual preemption problem,[1] we also face the problem of over-determination, as there may be more than one subset of the moves that are sufficient to cause the effect. In this paper, we extend previous accounts of actual causation in the situation calculus [19, 24] to identify minimal subsets of moves by some of the agents that are causes of the effect, i.e., sufficient to cause it. We also identify causal chains consisting of such minimal sets of moves, and notice that there may be several of them in the scenario for a given effect. We show that our notion of causal chains handles the issues associated with preemption and over-determination well.

In the next section, we start by reviewing the situation calculus (SC) and SCSGS. We also present our running example. In Section 3, we show how previous work on actual causation in the SC can be modified to identify causal chains. In Section 4, we consider minimal sets of agent moves as causes and the associated causal chains. In Section 5, we present some properties of our formalization. We conclude with some discussion in Section 6.

2. Preliminaries

Situation Calculus (SC). The SC is a well-known second-order language for representing and reasoning about dynamic worlds [28, 29]. In the SC, all changes are due to named actions, which are terms in the language. Situations represent a possible world history resulting from performing some actions. The constant S_0 is used to denote the initial situation where no action has been performed yet. The distinguished binary function symbol $do(a, s)$ denotes the successor situation to s resulting from performing the action a. The expression $do([a_1, \cdots, a_n], s)$ represents the situation resulting from executing actions $a_1, \cdots, a_n$, starting with situation s. As usual, a relational/functional fluent representing a property whose value may change from situation to situation takes a situation term as its last argument. There is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s. Also, the special binary predicate $s \sqsubset s'$ represents that s' can be reached from situation s by executing some sequence of actions. $s \sqsubseteq s'$ is an abbreviation of $s \sqsubset s' \vee s = s'$. $s < s'$ is an abbreviation of $s \sqsubset s' \wedge Executable(s')$, where $Executable(s)$ is defined as $\forall a', s'.\ do(a', s') \sqsubseteq s \supset Poss(a', s')$, i.e. every action performed in reaching situation s was possible in the situation in which it occurred. $s \leq s'$ is an abbreviation of $s < s' \vee s = s'$.

In the SC, a dynamic domain is specified using a basic action theory (BAT) $\mathcal{D}$ that includes the following sets of axioms: (i) (first-order or FO) initial state axioms $\mathcal{D}_{S_0}$, which indicate what was true initially; (ii) (FO) action precondition axioms $\mathcal{D}_{ap}$, characterizing $Poss(a, s)$; (iii) (FO) successor-state axioms $\mathcal{D}_{ss}$, indicating precisely when the fluents change; (iv) (FO) unique-names axioms $\mathcal{D}_{una}$ for actions, stating that different action terms represent distinct actions; and (v) (second-order or SO) domain-independent foundational axioms Σ, describing the structure of situations [30]. Although the SC is SO, Reiter [29] showed that for certain type of queries ϕ, $\mathcal{D} \models \phi$ iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \mathcal{R}[\phi]$, where $\mathcal{R}$ is a syntactic transformation operator called *regression* and $\mathcal{R}[\phi]$ is a SC formula that compiles dynamic aspects of the theory $\mathcal{D}$ into the query ϕ. Thus reasoning in the SC for a large class of interesting queries can be restricted to entailment checking w.r.t a FO theory [29].

Synchronous Game Structures (SCSGS). Following [27], we focus on games where there are n players/agents each of whom chooses a move at every time step. All such moves are executed *synchronously* and determine the next state of the game. At each time step, the state of the game is fully observable by all agents, as are all past moves of every agent.

[1] Preemption happens when two competing events try to achieve the same effect, and the latter of these fails to do so, as the earlier one has already achieved the effect.

To represent such multi-player synchronous games, we use a special class of BATs, called *situation calculus synchronous game structures (SCSGSs)*, which are defined as follows.

Agents. A SCSGS $\mathcal{D}$ involves a finite set of n agents, and we use a subsort *agents* of *Objects* which includes these finitely many agents $Ag_1, \ldots, Ag_n$, each denoted by a constant, and for which unique names $Ag_i \neq Ag_j$ for $i \neq j$ and domain closure $agent(x) \equiv x = Ag_1 \vee \cdots \vee x = Ag_n$ hold.

Moves. We also use a second subsort *Moves* of *Objects*, representing the possible moves of the agents. These come in finitely many types, represented by function symbols $M_i(\vec{x})$, which are parameterized by objects $\vec{x}$, with $Move(m) \equiv \bigvee_i \exists \vec{x}.m = M_i(\vec{x})$. Given that the parameters range over *Objects*, each agent may have an infinite number of possible moves at each time step. We have unique name and domain closure axioms (parameterized by objects) for these functions $M_i(\vec{x}) \neq M_j(\vec{y})$ for $i \neq j$, and $M_i(\vec{x}) = M_i(\vec{y}) \supset \vec{x} = \vec{y}$.

Actions. In SCSGSs, there is only *one action type*, $tick(m_1, \ldots, m_n)$, which represents the execution of a joint move by all the agents at a given time step. The action *tick* has exactly n parameters, $m_1, \ldots, m_n$, one per agent, which are of sort *Moves* and corresponds to the simultaneous choice of the move to perform by the n different agents.

Legal moves. The *legal moves* available to each agent in a given situation are specified formally using a special predicate *LegalM*, which is defined by statements of the following form (one for each agent Ag_i and move type M_i): $LegalM(Ag_i, M_i(\vec{x}), s) \doteq \Phi_{Ag_i,M_i}(\vec{x}, s)$, i.e., agent Ag_i can legally perform move $M_i(\vec{x})$ in situation s if and only if $\Phi_{Ag_i,M_i}(\vec{x}, s)$ holds. Technically *LegalM* is an abbreviation for $\Phi_{Ag_i,M_i}(\vec{x}, s)$, which is a uniform formula (i.e., a formula that only refers to a single situation s).

Precondition axioms. The precondition axiom for the action *tick* is fixed and specified in terms of *LegalM* as follows: $Poss(tick(m_1, \ldots, m_n), s) \equiv \wedge_{i=1,\ldots,n} LegalM(Ag_i, m_i, s)$. Thus the joint action by all agents $tick(m_1, \ldots, m_n)$ is executable if and only if each selected move m_i is a legal move for agent Ag_i in situation s. Since we only have one action type *tick*, this is the only precondition axiom in $\mathcal{D}_{poss}$.

Successor state axioms. We have *successor state axioms* $\mathcal{D}_{ssa}$, specifying the effects and frame conditions of the joint moves $tick(m_1, \ldots, m_n)$ on the fluents. Such axioms, as usual in basic action theories, are domain specific, and characterize the actual game under consideration. Within such axioms, the agent moves, which occur as parameters of *tick*, determine how fluents change as the result of joint moves.[2]

Initial situation description. Finally, the initial state of the game is axiomatized in the *initial situation description* $\mathcal{D}_{S_0}$ as usual, in a domain specific way.

Example. We use a variant of the well-known "bottle" example [3], where Suzy and Billy are throwing stones at a bottle. Suzy's stones are smaller and thus she requires two throws to break the bottle while Billy's stone is large and he needs just one throw to break it. The available moves of $ag \in \{Suzy, Billy\}$ can be one of $pick_{ag}$, representing the picking of one or more stones by agent ag (we assume that Suzy can pick both of her stones in one move), $throw_{ag}$, i.e. throwing of a stone by ag, and a catchall $other_{ag}$ move, denoting anything other than picking and throwing. The legality of these moves is specified below.

$$(a).\ LegalM(pick_{ag}, s) \doteq \neg Holding(ag, s),$$
$$(b).\ LegalM(throw_{ag}, s) \doteq Holding(ag, s), \qquad (c).\ LegalM(other_{ag}, s).$$

[2] In many cases, moves don't interfere with each other and the effects are just the union of those of each move. One can also exploit previous work on axiomatizing parallel actions to generate successor state axioms [29, 31].

Thus, e.g., throwing a stone is a legal move for agent ag in situation s is she is holding one or more stones in s. For simplicity, we assume that the $other_{ag}$ move is always possible.

There are three fluents in this domain, $Holding(ag, s)$, $Broken(s)$, and $SuzyThrown(s)$, which means that the agent ag is holding their stones in situation s, the bottle is broken in s, and *Suzy* has already thrown once before in s, respectively. The successor-state axioms of these fluents are as follows.

$$\begin{aligned}
(d).\ Holding(ag, do(a, s)) \equiv\ & [ag = Suzy \wedge \exists m.\ a = tick(pick_{Suzy}, m)] \vee \\
& [ag = Billy \wedge \exists m.\ a = tick(m, pick_{Billy})] \vee \\
& [ag = Suzy \wedge Holding(ag, s) \wedge \neg(\exists m.\ a = tick(throw_{Suzy}, m) \wedge SuzyThrown(s))] \vee \\
& [ag = Billy \wedge Holding(ag, s) \wedge \neg\exists m.\ a = tick(m, throw_{Billy})], \\
(e).\ Broken(do(a, s)) \equiv\ & [\exists m.\ a = tick(m, throw_{Billy})] \vee \\
& [\exists m.\ a = tick(throw_{Suzy}, m) \wedge SuzyThrown(s)], \\
(f).\ SuzyThrown(do(a, s)) \equiv\ & \exists m.\ a = tick(throw_{Suzy}, m) \vee SuzyThrown(s).
\end{aligned}$$

Thus, e.g., (d) states that an agent ag is holding stones after the action a is performed in situation s iff ag is Suzy and a is the *tick* action that involves her move of picking up stones; or if ag is Billy and a is the *tick* action that involves his move of picking up a stone; or ag is Suzy, who was already holding one or more stones in s, and a does not refer to a *tick* action that involves her move of throwing the last stone in hand; or ag is Billy, who already was holding a stone in s, and a is not a *tick* action involving his move of throwing a stone.

Finally, we assume that initially the agents are not holding any stones and the bottle is not broken, as specified by the following initial state axioms:

$$(g).\ \forall ag.\ \neg Holding(ag, S_0), \qquad (h).\ \neg Broken(S_0).$$

We will use $\mathcal{D}_{bt}$ to refer to this axiomatization. □

3. Actual Causation in the SC

Based on Batusov and Soutchanski's original proposal [19], Khan and Lespérance (KL) recently defined achievement cause in the SC [24]. Both of these frameworks study achievement causation, i.e. assume that the effect is false initially and becomes true after the execution of the actions in the scenario. Also, both assume that the scenario is a linear sequence of actions, i.e. these do not allow concurrent actions.

To formalize reasoning about effects, KL [24] introduced the notion of *dynamic formulae*. An effect φ in their framework is thus a situation-suppressed dynamic formula.[3] Given an effect φ, the actual causes are defined relative to a scenario s. When s is ground, the tuple $\langle \varphi, s \rangle$ is often called a *causal setting* [19]. Also, it is assumed that s is executable, and φ was false before the execution of the actions in s, but became true afterwards, i.e. $\mathcal{D} \models Executable(s) \wedge \neg\varphi[S_0] \wedge \varphi[s]$. Here $\varphi[s]$ denotes the formula obtained from φ by restoring the appropriate situation argument into all fluents in φ (see Def. 2).

Note that since all changes in the SC result from actions, the potential causes of an effect φ are identified with a set of action terms occurring in s. However, since s might include multiple occurrences of the same action, we need a way to uniquely identify each action occurrence in the scenario s. To deal with this, KL required that each situation be associated with a time-stamp, which can then be used to uniquely identify an action occurrence. A time-stamp is an integer for their theory. KL assumed that the initial situation starts at time-stamp 0 and each action increments the time-stamp by one. Thus, their action theory

[3] While KL also study epistemic causation, we restrict our discussion to objective causality only.

includes the following axioms:

$$timeStamp(S_0) = 0, \qquad \forall a, s, ts.\ timeStamp(do(a, s)) = ts \equiv timeStamp(s) = ts - 1.$$

With this, causes in their framework is a non-empty set of action-time-stamp pairs.

The notion of *dynamic formulae* is defined as follows:

Definition 1. *Let $\vec{x}$, θ_a, and $\vec{y}$ respectively range over object terms, action terms, and object and action variables. The class of* dynamic formulae *φ is defined inductively using the following grammar:* $\varphi ::= P(\vec{x}) \mid Poss(\theta_a) \mid After(\theta_a, \varphi) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists \vec{y}.\ \varphi$.

That is, a dynamic formula (DF) can be a situation-suppressed fluent, a formula that says that some action θ_a is possible, a formula that some DF holds after some action has occurred, or a formula that can built from other DF using the usual connectives. Note that φ can have quantification over object and action variables, but must not include quantification over situations or ordering over situations (i.e. $\sqsubset$). We will use φ for DF.

$\varphi[\cdot]$ is defined as follows:

Definition 2.

$$\varphi[s] \doteq \begin{cases} P(\vec{x}, s) & \text{if } \varphi \text{ is } P(\vec{x}) \\ Poss(\theta_a, s) & \text{if } \varphi \text{ is } Poss(\theta_a) \\ \varphi'[do(\theta_a, s)] & \text{if } \varphi \text{ is } After(\theta_a, \varphi') \\ \neg(\varphi'[s]) & \text{if } \varphi \text{ is } (\neg\varphi') \\ \varphi_1[s] \wedge \varphi_2[s] & \text{if } \varphi \text{ is } (\varphi_1 \wedge \varphi_2) \\ \exists \vec{y}.\ (\varphi'[s]) & \text{if } \varphi \text{ is } (\exists \vec{y}.\ \varphi') \end{cases}$$

We will now present a variant of KL's definition of causes in the SC. The idea behind how causes are computed is as follows. Given an effect φ and scenario s, if some action of the action sequence in s triggers the formula φ to change its truth value from false to true relative to $\mathcal{D}$, and if there are no actions in s after it that change the value of φ back to false, then this action is a *primary* or *direct* actual cause of achieving φ in s.

Definition 3 (Primary Cause [24])**.**

$$\begin{aligned} CausesDirectly(a, ts, \varphi, s) \doteq \exists s_a.\ & timeStamp(s_a) = ts \wedge (S_0 < do(a, s_a) \leq s) \\ & \wedge \neg\varphi[s_a] \wedge \forall s'.(do(a, s_a) \leq s' \leq s \supset \varphi[s']). \end{aligned}$$

That is, a executed at time-stamp ts is the *primary cause* of effect φ in situation s iff a was executed in a situation with time-stamp ts in scenario s, a caused φ to change its truth value to true, and no subsequent actions on the way to s falsified φ.

Now, note that a (primary) cause a might have been non-executable initially. Also, a might have only brought about the effect conditionally and this context condition might have been false initially. Thus earlier actions in the trace that contributed to the preconditions and the context conditions of a cause must be considered as causes as well. The following definition captures this. It is as in [24], but here we extend it to specify the causal chain that links the cause to the effect. It captures both primary and indirect causes and specifies the causal chains. It defines $CausesByChain(a, ts, cc, \varphi, s)$, meaning that action a at timestamp ts is a cause of an effect φ in scenario s through causal chain cc: [4]

[4] In this, we need to quantify over situation-suppressed DF. Thus we must encode such formulae as terms and formalize their relationship to the associated SC formulae. This is tedious but can be done essentially along the lines of [32]. We assume that we have such an encoding and use formulae as terms directly.

Definition 4 (Actual Cause Through Causal Chain)**.**

$$\begin{aligned}\mathit{CausesByChain}(a, ts, cc, \varphi, s) \doteq \forall P.[&\forall a, ts, s, cc, \varphi.(\mathit{CausesDirectly}(a, ts, \varphi, s) \supset P(a, ts, ((a, ts)), \varphi, s))\\ &\wedge \forall a, ts, cc', s, \varphi.(\exists a', ts', s'.(\mathit{CausesDirectly}(a', ts', \varphi, s)\\ &\qquad \wedge \mathit{timeStamp}(s') = ts' \wedge s' < s\\ &\qquad \wedge P(a, ts, cc', [\mathit{Poss}(a') \wedge \mathit{After}(a', \varphi)], s')\\ &\qquad \wedge cc = \mathit{Append}(cc', (a', ts'))\\ &\quad \supset P(a, ts, cc, \varphi, s))\\ &] \supset P(a, ts, cc, \varphi, s).\end{aligned}$$

Thus, *CausesByChain* is defined to be the least relation P such that if a executed at time-stamp ts directly causes φ in scenario s then (a, ts, cc, φ, s) is in P, where $cc = ((a, ts))$; and if a' executed at ts' is a direct cause of φ in s, the time-stamp of s' is ts', $s' < s$, and $(a, ts, cc', [\mathit{Poss}(a') \wedge \mathit{After}(a', \varphi)], s')$ is in P (i.e. a executed at ts is a direct or indirect cause of $[\mathit{Poss}(a') \wedge \mathit{After}(a', \varphi)]$ in s' through causal chain cc'), then (a, ts, cc, φ, s) is in P, where $cc = \mathit{Append}(cc', (a', ts'))$. Here the effect $[\mathit{Poss}(a') \wedge \mathit{After}(a', \varphi)]$ requires a' to be executable and φ to hold after a'. Also, *Append* is defined as follows.

Definition 5 (Append)**.**

$$\mathit{Append}(((a_1, ts_1), \ldots, (a_n, ts_n)), (a, ts)) \doteq ((a_1, ts_1), \ldots, (a_n, ts_n), (a, ts)).$$

Tick Actions as Causes in the SCSGS. The above formalization of actual causation was formulated for domains specified by BATs in the situation calculus. However, it can be used directly for SCSGS domains, as long as one focuses on identifying the *tick* actions in the scenario that caused the effect, and causal chains consisting of *tick* actions. This is not surprising as SCSGS are special kinds of BATs. We illustrate this in the example below.

Example (cont'd). Consider the scenario σ_1, where: $\sigma_1 = do([tick(pick_{Suzy}, other_{Billy}), tick(throw_{Suzy}, pick_{Billy}), \quad tick(other_{Suzy}, other_{Billy}), \quad tick(throw_{Suzy}, throw_{Billy})], S_0)$. We want to find the actual causes of the effect $\varphi_1 = \mathit{Broken}(s)$. We can show that:[5]

Proposition 1 (Complete Causal Chain in σ_1)**.**

$$\begin{aligned}&\mathcal{D}_{bt} \models \mathit{CausesByChain}(tick(pick_{Suzy}, other_{Billy}), 0, cc, \varphi_1, \sigma_1),\\ &\textit{where } cc = ((tick(pick_{Suzy}, other_{Billy}), 0), (tick(throw_{Suzy}, pick_{Billy}), 1),\\ &\qquad\qquad (tick(throw_{Suzy}, throw_{Billy}), 3)).\end{aligned}$$

Explaining backward in *cc*, the last *tick* action executed at time-stamp 3 is included in the causal chain as (either of the moves in) it directly caused the breaking of the bottle. The second *tick* action executed at 1 is also included because it is a (secondary/indirect) cause as it brought about the preconditions of the last *tick* action (by making Billy's throw legal), besides bringing about the context condition (that *SuzyThrown*) under which Suzy's second throw can brake the bottle. Finally, the first *tick* action is also a cause as it made the second *tick* action executable. □

While the above formalization provides some insight on what *tick* actions are causes and can be used to identify the completely irrelevant *tick* actions, e.g. the one at time-stamp 2, observe that some irrelevant moves might still be included in the discovered causes, such as $other_{Billy}$ at time-stamp 0 in our example. In other words, our formalization of this does not specify what moves within the identified *tick* actions are contributing to the effect. To deal with this, we next propose a formalization of agent moves as causes.

[5]Note that since Definition 4 inductively constructs the causal chain, considering some of the causes, e.g., the primary cause, will only give us a suffix of the complete causal chain *cc*; for simplicity, we thus only show the most indirect cause below, which captures the complete chain *cc*.

4. Agent Moves as Causes in the SCSGS

We now go a step further by pinpointing the moves that actually contributed to the effect within the *tick* actions that are identified as causes. Note that, since unlike actions, agent moves within each *tick* action are concurrently performed, it is possible that more than one alternative chain of subsets of moves in the scenario are each by itself sufficient to bring about the effect. For instance, in our example, either $((pick_{Suzy}, 0), (throw_{Suzy}, 1), (throw_{Suzy}, 3))$ or $((pick_{Billy}, 1), (throw_{Billy}, 3))$ would have been sufficient to break the bottle. Just as with causal chains in the SC, we will identify these refined causal chains in two steps. In the first step, we identify the minimal set of moves in each action that is a direct cause of the effect in some refined chain (e.g., $throw_{Billy}$ in the last *tick* of the second refined causal chain above). We call these sets of direct causes *minimal moves primary causes* since they are minimal sets of moves that are causes. However, we must consider that there might be more than one minimal moves primary cause in one single action. For example in the last tick of our example, we can consider either only Suzy's throwing or Billy's throwing to be a minimal moves primary cause. In the second step, using refined causes, we define the refined chains mentioned above, which we call *minimal moves causal chains*.

In keeping with the formalization of dynamic domains in the SC, we will consider actions (but not moves) as (refined) causes.[6] Thus our formalization of this does not omit the irrelevant moves in each time-stamp altogether, but rather replaces them with the special move *wait* within the *tick* action to remove their effects. *wait* has no effects (the domain modeler must ensure this) and is always legal: $\forall ag, s.\ LegalM(ag, wait, s)$. Thus, for instance, in our example, one such refined action that is a cause is $tick(pick_{Suzy}, wait)$. We collect all of these for all causal chains in our new definition of causes.

We now define minimal moves primary causes:

Definition 6 (Minimal Moves Primary Cause)**.**

$$MinMovCausesDir(a, ts, (a', ts), \varphi, s) \doteq \exists s_a.\ timeStamp(s_a) = ts \wedge (S_0 < do(a, s_a) \leq s) \wedge \neg\varphi[s_a] \wedge \forall s'.(do(a, s_a) \leq s' \leq s \supset \varphi[s']) \wedge MinSuffSubset(a', a, s_a, \varphi, s).$$

The above definition is exactly as the definition of primary causes (Def. 4), but has an additional parameter (a', ts) that returns a tuple consisting of a minimal subset of moves a' of the primary cause a that, when executed in s_a (i.e., the situation where the primary cause was executed in the original scenario), is sufficient to cause the effect φ in s (formalized using *MinSuffSubset* below), and the timestamp ts of a.

$MinSuffSubset(a', a, s', \varphi, s)$ means that the *tick* action a' consists of a sufficient subset of moves of a in s' to achieve φ up to s, and it is minimal.

Definition 7.

$$MinSuffSubset(a', a, s', \varphi, s) \doteq SuffSubset(a', a, s', \varphi, s) \wedge \neg\exists a^*.a^* \neq a' \wedge SuffSubset(a^*, a', s', \varphi, s).$$

a' is a sufficient subset of moves of a in s' to achieve φ up to s, i.e. $SuffSubset(a', a, s', \varphi, s)$, iff a' is a subset of moves of a, and the execution of this subset a' in s' is sufficient to cause φ in s, i.e. φ becomes true after a' is executed in s' and it remains true after the subsequent actions between $do(a, s')$ and s are executed starting in $do(a', s')$.

Definition 8.

$$\begin{aligned} SuffSubset(a', a, s', \varphi, s) \doteq\ & SubsetMovs(a', a) \wedge \exists s''.\ timeStamp(s'') = timeStamp(s) \wedge s' < s'' \\ & \wedge \forall a_1, s_1, a_1', s_1'.[(do(a, s') < do(a_1, s_1) \leq s \wedge do(a', s') < do(a_1', s_1') \leq s'' \wedge \\ & \qquad timeStamp(s_1') = timeStamp(s_1)) \supset (a_1 = a_1')] \\ & \wedge (\forall s_1'.(s' < s_1' \leq s'') \supset \varphi[s_1']). \end{aligned}$$

[6] Note that the action theory specifies how the situation changes when actions, i.e., joint moves, are performed. This allows interfering or synergic effects to be specified.

Here $SubsetMovs(a', a)$, meaning that the *tick* action a' is exactly as a, but with some of the moves possibly replaced with the *wait* move, is defined as follows:

Definition 9 (The Moves of a' Consists of a Subset of the Moves of a)**.**

$$SubsetMovs(a', a) \doteq \exists m'_1, \cdots, m'_n, m_1, \cdots, m_n.\ a' = tick(m'_1, \cdots, m'_n) \\ \wedge a = tick(m_1, \cdots, m_n) \wedge (\forall j.\ 1 \leq j \leq n \supset (m'_j = m_j \vee m'_j = wait)).$$

Using minimal moves primary causes, we next formalize minimal moves causal chains. We do this by defining a variant of *CausesByChain* that inductively constructs the minimal moves chain instead.

Definition 10 (Minimal Moves Cause Through Causal Chain)**.**

$$\begin{aligned} &MinMovesCausesByChain(a, ts, cc, \varphi, s) \doteq \\ &\quad \forall P.[\forall a, ts, cc, a', s, \varphi.(MinMovCausesDir(a, ts, (a', ts), \varphi, s) \supset P(a, ts, ((a', ts)), \varphi, s)) \\ &\qquad \wedge \forall a, ts, cc', s, \varphi.(\exists a'', a', ts', s'.(MinMovCausesDir(a', ts', (a'', ts'), \varphi, s) \\ &\qquad\quad \wedge timeStamp(s') = ts' \wedge s' < s \\ &\qquad\quad \wedge P(a, ts, cc', [Poss(a'') \wedge After(a'', \varphi)], s') \\ &\qquad\quad \wedge cc = Append(cc', (a'', ts')) \\ &\qquad \supset P(a, ts, cc, \varphi, s)) \\ &\quad] \supset P(a, ts, cc, \varphi, s). \end{aligned}$$

Thus *MinMovesCausesByChain* is the smallest set such that if a *tick* action a executed at time-stamp ts directly caused the effect φ in scenario s with the minimal moves primary cause (a', ts), then (a, ts, cc, φ, s) is in that set, where $cc = ((a', ts))$; and if a' executed at ts' is a direct cause of φ in s with minimal moves primary cause (a'', ts'), the time-stamp of s' is ts', $s' < s$, and $(a, ts, cc', [Poss(a'') \wedge After(a'', \varphi)], s')$ is in P (i.e. a executed at ts is a direct or indirect cause of $[Poss(a'') \wedge After(a'', \varphi)]$ in s' through minimal moves causal chain cc'), then (a, ts, cc, φ, s) is in P, where $cc = Append(cc', (a'', ts'))$. This thus incrementally constructs the refined causal chains using *MinMovCausesDir*.

Example (cont'd). We can show the following result about refined causal chains.[7]

Proposition 2 (Complete Refined Causal Chains in σ_1)**.**

$$\begin{aligned} &\mathcal{D}_{bt} \models MinMovesCausesByChain(tick(pick_{Suzy}, other_{Billy}), 0, cc_1, \varphi_1, \sigma_1) \\ &\quad \wedge MinMovesCausesByChain(tick(throw_{Suzy}, pick_{Billy}), 1, cc_2, \varphi_1, \sigma_1),\ \ where \\ &cc_1 = ((tick(pick_{Suzy}, wait), 0), (tick(throw_{Suzy}, wait), 1), (tick(throw_{Suzy}, wait), 3)), \\ &and\ cc_2 = ((tick(wait, pick_{Billy}), 1), (tick(wait, throw_{Billy}), 3)). \end{aligned}$$

Thus, in our example, we have two distinct causal chains, one that stems from the refined primary cause involving Suzy's second throw move and Billy's *wait* move at time-stamp 3, and another originating from Billy's throw and Suzy's *wait* move, again at time-stamp 3. Importantly, as we will show in the next section, there is no causal chain that involves both Suzy and Billy's throw moves at time-stamp 3, avoiding the problem of over-determination (as each of these moves would have been sufficient for breaking the bottle).

Note that in the above, all the minimal move causes involve a single (non-*wait*) move. But this is not always the case. For example, if we replace the effect φ_1 by $\varphi^* = Broken(s) \wedge SuzyThrown(s)$, then a minimal move causal chain is:

$$((tick(pick_{Suzy}, wait), 0), (tick(throw_{Suzy}, pick_{Billy}), 1), (tick(wait, throw_{Billy}), 3)).$$

□

[7] Again, as in Proposition 1, we choose the most indirect causes to show the complete refined causal chains.

5. Properties

We now show that our formalization of refined causes and refined causal chains have some interesting properties. We start with the problem of preemption.

Preemption. Preemption occurs when there are more than one competing contributors (actions) to an effect, but they happen one after another/consecutively. In such cases, only the first of these should be identified as the actual cause. The (effects of the) latter actions are said to be preempted by the actual cause. Our definition of refined causes and causal chains above are based on Khan and Lespérance's formalization of causes [24], which filter out the preempted contributors, and thus our formalization also handles the preemption problem correctly. We illustrate this using the following example.

Example 2. Consider the new scenario σ_2, where

$$\sigma_2 = do([tick(pick_{Suzy}, pick_{Billy}), tick(throw_{Suzy}, other_{Billy}), tick(throw_{Suzy}, other_{Billy}), tick(other_{Suzy}, throw_{Billy})], S_0).$$

In this, we can show the following result.

Proposition 3.

$$\mathcal{D}_{bt} \models \neg\exists cc.\ CausesByChain(tick(other_{Suzy}, throw_{Billy}), 3, cc, \varphi_1, \sigma_2) \wedge \neg\exists cc, m.\ MinMovesCausesByChain(tick(m, throw_{Billy}), 3, cc, \varphi_1, \sigma_2).$$

Thus as expected, Billy's throw is not considered as part of any (refined) causal chain. □

Over-determination. Over-determination happens when the effect is contributed by some events, but a smaller subset of these would have been sufficient for the effect to hold. For example, in a voting scenario, where 6 out of 10 votes are required for a candidate to win, if 7 voters voted for candidate A, saying that all of these 7 votes are the cause of candidate A's winning the ballot would be over-determination as any 6 of these would suffice. An acceptable solution to this is to only identify any 6-vote subsets to be an actual cause [23] (or refined causal chain, in our formalization).

Example (cont'd). Using our first bottle example where the scenario is σ_1, we now argue that our notion of refined causal chains avoids over-determining causes. Note that in this example, our *MinMovesCausesByChain* construct avoids over-determination by inductively specifying all possible sets of refined causal chains that are sufficient to break the bottle. As discussed above, there are only two refined causal chains; the first one only consists of Billy's moves, and the second only consists of Suzy's. The definition *MinMovesCausesByChain* ensures that only these two chains exist, since if we were to consider the chain that includes the time-stamp 3 throw moves of both Suzy and Billy, it will not be a minimal one as either throw would have been sufficient to break the bottle. Hence, we cannot have the moves of both agents involved in the same refined causal chain. Formally, we can show that:

Proposition 4.

$$\mathcal{D}_{bt} \models \neg\exists a, ts, cc.\ MinMovesCausesByChain(a, ts, Append(cc, (tick(throw_{Suzy}, throw_{Billy}), 3)), \varphi_1, \sigma_1).$$

Thus, $tick(throw_{Suzy}, throw_{Billy})$ at time-stamp 3 cannot be a part of any causal chain for any action and timestamp. □

In general, since refined causal chains are constructed to be minimal, if we have two refined chains in the same timestamp, it cannot be the case that one of them is a refined version of the other (i.e., has a subset of moves of the other).

Theorem 5 (No Over-Determination)**.**

$$\mathcal{D} \models \forall a, a_1, a_2, ts, cc, \varphi, s.(\mathit{MinMovesCausesByChain}(a, ts, \mathit{Cons}((a_1, ts), cc), \varphi, s) \wedge \mathit{MinMovesCausesByChain}(a, ts, \mathit{Cons}((a_2, ts), cc), \varphi, s) \supset (a_1 = a_2 \vee (\neg \mathit{SubsetMovs}(a_1, a_2) \wedge \neg \mathit{SubsetMovs}(a_2, a_1))),$$

where $\mathit{Cons}((a, ts), cc)$ *denotes the sequence where* (a, ts) *is added to the front of* cc*.*

Proof Sketch: By induction on the length of the minimal moves causal chain using properties of minimal sufficient subsets of joint moves. □

Persistence. Finally, we study the conditions under which (refined) causal chains persist when the scenario changes.

Theorem 6 (Persistence of the Causal Chain)**.**

$$\mathcal{D} \models \forall s, s', cc, ts, a, \varphi.\ \mathit{CausesByChain}(a, ts, cc, \varphi, s) \wedge s \leq s' \wedge \forall s^*, a^*.(s \leq do(a^*, s^*) \leq s' \supset \varphi[s^*]) \supset \mathit{CausesByChain}(a, ts, cc, \varphi, s').$$

Proof Sketch: By induction on the length of the causal chain. □

That is, if a *tick* action a executed in ts is the cause of an effect φ in scenario s through causal chain cc, then a in ts remains the cause of φ in all subsequent situations/scenarios s' if φ does not change after it was achieved in s. This is because since the situation where φ was achieved does not change in the extended scenario, neither does the causal chain.

A similar result can be shown for refined causal chains.

Theorem 7 (Persistence of Refined Causal Chains)**.**

$$\mathcal{D} \models \forall s, s', cc, ts, a, \varphi.\ \mathit{MinMovesCausesByChain}(a, ts, cc, \varphi, s) \wedge s \leq s' \wedge \forall s^*, a^*.(s \leq do(a^*, s^*) \leq s' \supset \varphi[s^*]) \supset \mathit{MinMovesCausesByChain}(a, ts, cc, \varphi, s').$$

Proof Sketch: By induction on the length of the minimal moves causal chain using properties of minimal sufficient subsets of joint moves. □

6. Discussion and Conclusion

To support casual reasoning in multiagent domains, in this paper we proposed a formalization of actual causation in SCSGSs. We showed that one can adopt the definition of achievement causes in the SC to identify the relevant *tick* actions as causes and the associated causal chain in SCSGSs. However, since the chain can now include multiple sets of causes, each of which are independently sufficient to bring about the effect, to avoid over-determination we also formalized refinements of the identified causal chains. As shown, our account properly handles the problems with preemption and over-determination.

Note that, while the popular SEM-based frameworks of actual causation and their derivatives seem to allow actions by multiple agents, these have other major limitations. For instance, often these consider the occurrence of a set of events without specifying their order of execution or whether they occur concurrently (all events are simply assumed to have happened). Events are also considered to be independent, which is another strong assumption. One cannot distinguish between the occurrence of an event and its effect holding. While recent action-theoretic formalizations of causality are meant to deal with these limitations, as mentioned earlier, these only handle single-agent or turn-taking multi-agent scenarios. Indeed to the best of our knowledge, our proposal is the first to accommodate synchronous concurrent moves by multiple agents while studying actual causes in formal action-theoretic

frameworks. We are currently investigating how various notions of responsibility can be formalized based on this framework.

Acknowledgements

This work is partially supported by the National Science and Engineering Research Council of Canada, by the University of Regina, and by York University.

References

[1] J. Y. Halpern. "Axiomatizing Causal Reasoning". In: *Journal of Artificial Intelligence Research* 12 (2000), pp. 317–337.

[2] D. Lewis. "Causation". In: *Journal of Philosophy* 70.17 (1973), pp. 556–567.

[3] N. Hall. "Two Concepts of Causation". In: *Causation and Counterfactuals*. Ed. by J. Collins, N. Hall, and L. A. Paul. MIT Press, 2004, pp. 225–276.

[4] J. Pearl. *On the Definition of Actual Cause*. Tech. rep. R-259. University of California Los Angeles, 1998.

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

[6] C. Hitchcock. "The Intransitivity of Causation Revealed in Equations and Graphs". In: *The Journal of Philosophy* 98.6 (2001), pp. 273–299.

[7] T. Eiter and T. Lukasiewicz. "Complexity Results for Structure-based Causality". In: *Artificial Intelligence* 142.1 (2002), pp. 53–89.

[8] A. Bochman. "A Logic For Causal Reasoning". In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by G. Gottlob and T. Walsh. Morgan Kaufmann, 2003, pp. 141–146.

[9] M. Hopkins. "The Actual Cause: From Intuition to Automation". PhD thesis. University of California Los Angeles, 2005.

[10] M. Hopkins and J. Pearl. "Causality and Counterfactuals in the Situation Calculus". In: *Journal of Logic and Computation* 17.5 (2007), pp. 939–953.

[11] J. Y. Halpern. "A Modification of the Halpern-Pearl Definition of Causality". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Q. Yang and M. J. Wooldridge. AAAI Press, 2015, pp. 3022–3033.

[12] M. Gladyshev, N. Alechina, M. Dastani, D. Doder, and B. Logan. "Dynamic Causality". In: *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*. Ed. by K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Radulescu. Vol. 372. Frontiers in Artificial Intelligence and Applications. IOS Press, 2023, pp. 867–874.

[13] J. L. Mackie. "Causes and Conditions". In: *American Philosophical Quarterly* 2.4 (1965), pp. 245–264.

[14] R. W. Wright. "Causation in tort law". In: *California Law Review* 73.6 (1985), 1735—1828.

[15] A. Bochman. "Actual Causality in a Logical Setting". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by J. Lang. ijcai.org, 2018, pp. 1730–1736.

[16] S. M. Khan and M. Soutchanski. "Necessary and Sufficient Conditions for Actual Root Causes". In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 800–808.

[17] S. Beckers and J. Vennekens. "A Principled Approach to Defining Actual Causation". In: *Synthese* 195.2 (2018), pp. 835–862.

[18] V. Batusov and M. Soutchanski. "Situation Calculus Semantics for Actual Causality". In: *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, COMMONSENSE 2017, London, UK, November 6-8, 2017*. Ed. by A. S. Gordon, R. Miller, and G. Turán. Vol. 2052. CEUR Workshop Proceedings. CEUR-WS.org, 2017.

[19] V. Batusov and M. Soutchanski. "Situation Calculus Semantics for Actual Causality". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by S. A. McIlraith and K. Q. Weinberger. AAAI Press, 2018, pp. 1744–1752.

[20] A. Mehmood and S. M. Khan. "Towards a Definition of Primary Cause in Hybrid Dynamic Domains". In: *Proceedings of the 37th Canadian Conference on Artificial Intelligence (Canadian AI-24), Guelph, Ontario, Canada*. 2024.

[21] M. Rostamigiv, S. M. Khan, Y. Lespérance, and M. Yadkoo. "A Logic of Actual Cause for Non-Deterministic Dynamic Domains". In: *Proceedings of the 21st European Conference on Multi-Agent Systems (EUMAS-24), August 26-28, Dublin, Ireland*. Springer, 2024.

[22] S. M. Khan, Y. Lespérance, and M. Rostamigiv. "Reasoning about Actual Causes in Non-deterministic Domains". In: *Proceedings of the 39th Annual AAAI Conference on Artificial Intelligence (AAAI-25), February 25 - March 4, 2025, Philadelphia, Pennsylvania, USA*. AAAI Press, 2025.

[23] J. Y. Halpern. *Actual Causality*. MIT Press, 2016. ISBN: 978-0-262-03502-6.

[24] S. M. Khan and Y. Lespérance. "Knowing Why - On the Dynamics of Knowledge about Actual Causes in the Situation Calculus". In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. Ed. by F. Dignum, A. Lomuscio, U. Endriss, and A. Nowé. ACM, 2021, pp. 701–709.

[25] S. M. Khan and M. Rostamigiv. "On Explaining Agent Behaviour via Root Cause Analysis: A Formal Account Grounded in Theory of Mind". In: *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland*. Ed. by K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Radulescu. Vol. 372. Frontiers in Artificial Intelligence and Applications. IOS Press, 2023, pp. 1239–1247.

[26] V. Yazdanpanah, E. H. Gerding, S. Stein, M. Dastani, C. M. Jonker, T. J. Norman, and S. D. Ramchurn. "Reasoning about responsibility in autonomous systems: challenges and opportunities". In: *AI Soc.* 38.4 (2023), pp. 1453–1464.

[27] G. De Giacomo, Y. Lespérance, and A. R. Pearce. "Situation Calculus Game Structures and GDL". In: *ECAI*. 2016, pp. 408–416.

[28] J. McCarthy and P. J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: *Machine Intelligence* 4 (1969), pp. 463–502.

[29] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA, USA: MIT Press, 2001. ISBN: 9780262182188.

[30] H. J. Levesque, F. Pirri, and R. Reiter. "Foundations for the Situation Calculus". In: *Electronic Transactions on Artificial Intelligence (ETAI)* 2 (1998), pp. 159–178.

[31] J. Pinto. "Concurrent Actions and Interacting Effects". In: *KR*. 1998, pp. 292–303.

[32] G. De Giacomo, Y. Lespérance, and H. J. Levesque. "ConGolog, A Concurrent Programming Language based on the Situation Calculus". In: *Artificial Intelligence* 121.1-2 (2000), pp. 109–169.