Causality & Control flow

Abstract

Causality has been the issue of philosophic debate since Hippocrates. It is used in formal verification and testing, e.g., to explain counterexamples or construct fault trees. Recent work defines actual causation in terms of Pearl's causality framework, but most definitions brought forward so far struggle with examples where one event preempts another one. A key point to capturing such examples in the context of programs or distributed systems is a sound treatment of control flow. We discuss how causal models should incorporate control flow and discover that much of what Pearl/Halpern's notion of contingencies tries to capture is captured better by an explicit modelling of the control flow in terms of structural equations and an arguably simpler definition. Inspired by causality notions in the security domain, we bring forward a definition of causality that takes these control-variables into account. This definition provides a clear picture of the interaction between control flow and causality and captures these notoriously difficult preemption examples without secondary concepts. We give convincing results on a benchmark of 34 examples from the literature.

Introduction

A growing body of literature is concerned with notions of accountability in security protocols, as in many scenarios, e.g., electronic voting, certified e-mail, online transactions, or when personal data is processed within a company, not all agents can be trusted to behave according to some established protocol [4]. Thus, in order to specify accountability, i.e., the ability of a protocol to detect misbehaviour, but also to reason about test coverage [6], or explain attacks [3], the information security domain needs a reliable notion of what it means for a protocol event to *cause* a security violation in a given scenario. The security domain has proposed causal notions specifically for network traces, which, in contrast to traditional notions of causality, capture actions sufficient to cause an event and put a focus on control flow [8]. In this work, we investigate these ideas in a more general setting; generalising, improving and validating them to provide a sound basis for causal reasoning in protocols and beyond.

The problem we investigate is called *actual causation*, as opposed to *type causation*, which aims at deriving general statements, e.g., "smoking causes cancer" not linked to a specific scenario. Starting from Lewis' 'closest-world concept' [23], philosophers have largely accepted *counter-factual* reasoning, i.e., investigating causal claims by regarding hypothetical scenarios of the form 'had A not occurred, B would not have

occurred', as a means to determine actual causation. Pearl's causality framework [25] provides a basis for such reasoning.

So far, control flow has largely been ignored in causal reasoning. This is not very surprising, considering that the causation literature typically treats real-life examples inspired from criminal law. Control flow is simply not a well-defined notion there. Albeit, precisely those scenarios where the order of events is relevant, e.g., an event might prevent another event from happening, turn out to be notoriously difficult to explain using counterfactual reasoning. Once we consider each potential course of events as a control-flow path consisting of events that may enable or prevent each other, they become easy to handle.

To accommodate preemption, Pearl and Halpern's very influential notion of causation has been modified several times [13], complemented with secondary notions [16] and ad-hoc modifications have been proposed [16, p. 26]. Neither of these solutions provides a satisfying answer as to how these examples should be handled in general.

In this work, we provide an account of the relation between control flow and causation. This gives us the means to adequately capture preemption in cases where we can speak of control flow. We propose that control flow variables should be modelled explicitly, as they can capture the course of events that lead to a certain outcome. Once they are made explicit, we can capture these difficult examples and provide a notion of actual causation that is simple and intuitive, captures joint as well as independent causes, gets by without secondary notions of normality and defaults and readily applies to Pearl's causality framework. Our contributions are the following:

1. We explain how control flow should be incorporated in causal models and propose a formalism that makes control flow explicit.
2. We show how control flow helps to handle preemption without resorting to secondary notions like defaults and normality or ad-hoc modifications to the model. Preemption examples are notorious for being difficult to handle [18, 20, 16],
3. We relate control flow to *structural contingencies* introduced by Halpern and Pearl [17], providing evidence that what this notion achieves is very similar to a simple fixing of control flow variables, and that it provides unintuitive results when applied outside control flow.
4. Finally, we validate our proposal, control flow preserving sufficient causation, on 34 examples, including all scenarios discussed in [28] and [13].

Notation We write $\vec{t}$ for a sequence $t_1, \ldots, t_n$ if n is clear from the context and use $(a_1, \ldots, a_n) \cdot (b_1, \ldots, b_m) = (a_1, \ldots, a_n, b_1, \ldots, b_m)$ to denote concatenation. We filter a sequence l by a set S, denoted $l|_S$, by removing each element that is not in S.

Causality framework (Review)

We review the causality framework introduced by Pearl [25], also known as the *structural equations model*. The causality framework models how random variables influence each other. The set of random variables, which we assume discrete, is partitioned

into a set $\mathcal{U}$ of *exogenous* variables, variables that are outside the model, e.g., in the case of a security protocol, the scheduling and the attack the adversary decides to mount, and a set $\mathcal{V}$ of *endogenous* variables, which are ultimately determined by the value of the exogenous variables. A *signature* is a triple consisting of $\mathcal{U}$, $\mathcal{V}$ and function $\mathcal{R}$ associating a range, i.e., a set, to each variable $Y \in \mathcal{U} \cup \mathcal{V}$. A causal model on this signature defines the relation between endogenous variables and exogenous variables or other endogenous variables in terms of a set of equations.

Definition 1 (Causal model)**.** *A* causal model M *over a signature* $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ *is a pair of said signature* $\mathcal{S}$ *and a set of functions* $\mathcal{F} = \{ F_X \}_{X \in \mathcal{V}}$ *such that, for each* $X \in \mathcal{V}$*,*

$$F_X : (\bigtimes_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\bigtimes_{Y \in \mathcal{V} \setminus \{ X \}} \mathcal{R}(Y)) \to \mathcal{R}(X).$$

Each causal model induces a *causal network*, a graph with a node for each variable in $\mathcal{V}$, and an edge from X to Y iff F_Y depends on X. (Y depends on X iff there is a setting for the variables in $\mathcal{V} \cup \mathcal{U} \setminus \{ X, Y \}$ such that modifying X changes the value of Y.) If the causal graph associated to a causal model M is acyclic, then each setting $\vec{u}$ of the variables in $\mathcal{U}$ provides a unique solution to the equations in M. Throughout this paper, we only consider causal models that have this property. We call a vector setting the variables in $\mathcal{U}$ a *context*, and a pair $(M, \vec{u})$ of a causal model and a context *a situation*. All modern definitions of causality follow a counterfactual approach, which requires answering 'what if' questions. It is not always possible to do this by observing actual outcomes. Consider the following example.

Example 1 (Wet ground)**.** *The ground in front of Charlie's house is slippery when wet. Not only does it become wet when it rains; if the neighbour's sprinkler is turned on, the ground gets wet, too. The neighbour turns on the sprinkler unless it rains. Let* $R \in \mathcal{U}$ *be 1 if it rains, and 0 otherwise, and consider the following equations for a causal model on* R *and endogenous variables* W*,* S *and* F *with range* $\{0, 1\}$*.*

$$\begin{aligned} S &= \neg R && \textit{(The sprinkler is on if it does not rain.)} \\ W &= R \vee S && \textit{(The sprinkler or the rain wets the ground.)} \\ F &= W && \textit{(Charlie falls when the ground is slippery.)} \end{aligned}$$

Clearly, the ground being wet is a cause to Charlie's falling, but we cannot argue counterfactually, because the counterfactual case never actually occurs: the ground is always wet. We need to intervene on the causal model.

Definition 2 (Modified causal model)**.** *Given a causal model* $M = ((\mathcal{U}, \mathcal{V}, \mathcal{R}), \mathcal{F})$*, we define the* modified causal model $M_{\vec{X} \leftarrow \vec{x}}$ *over the signature* $\mathcal{S}_{\vec{X}} = (\mathcal{U}, \mathcal{V} \setminus \vec{X}, \mathcal{R}|_{\mathcal{U} \cup \mathcal{V} \setminus \vec{X}})$ *by replacing each endogenous variable* $X \in \vec{X}$ *in* $\mathcal{F}$ *with the corresponding* $x \in \vec{x}$*, obtaining* $\mathcal{F}_{\vec{X} \leftarrow \vec{x}}$*. Then,* $M_{\vec{X} \leftarrow \vec{x}} = (\mathcal{S}_{\vec{X}}, \mathcal{F}_{\vec{X} \leftarrow \vec{x}})$*.*

We can now define how to evaluate queries on causal models w.r.t. interventions on a vector of variables, which allows us to answer 'what if' questions.

Definition 3 (Causal formula)**.** *A* causal formula *has the form* $[Y_1 \leftarrow y_1, \ldots, Y_n \leftarrow y_n]\varphi$ *(abbreviated* $[\vec{Y} \leftarrow \vec{y}]\varphi$*), where*

- *φ is a boolean combination of primitive events, i.e., formulas of the form $X = x$ for $X \in \mathcal{V}$, $x \in \mathcal{R}(X)$,*
- *$Y_1, \ldots, Y_n \in \mathcal{V}$ are distinct,*
- *$y_i \in \mathcal{R}(Y_i)$.*

We write $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}]\varphi$ ($[\vec{Y} \leftarrow \vec{y}]\varphi$ is true in a causal model M given context $\vec{u}$) if the (unique) solution to the equations in $M_{\vec{Y} \leftarrow \vec{y}}$ in the context $\vec{u}$ satisfies φ.

Sufficient causes

We define a causality notion based on sufficiency. In order to allow for comparison with existing notions of causation, we chose to formulate this notion in Pearl's causation framework, as opposed to formalisms that already incorporate temporality, e.g., Kripke structures, but would obscure this comparison. This simplistic notion of causality, which we will later extend to models with explicit control flow, captures the causal variables that, by themselves, guarantee the outcome.

Definition 4 (Sufficient cause)**.** *$\vec{X} = \vec{x}$ is a* sufficient cause *of φ in $(M, \vec{u})$ if the following three conditions hold.*

SF1. $(M, \vec{u}) \models (\vec{X} = \vec{x}) \wedge \varphi$.

SF2. For all $\vec{z}$, $(M, \vec{u}) \models [(\mathcal{V} \setminus \vec{X}) \leftarrow \vec{z}]\varphi$.

SF3. $\vec{X}$ is minimal: No strict subset $\vec{X}'$ of $\vec{X}$ satisfies SF1 and SF2.

We say $\vec{X}$ is a sufficient cause for φ if this is the case for some $\vec{x}$. Any non-empty subset of $\vec{X}$ is part *of the sufficient cause $\vec{X}$.*

Sufficient causes are well-suited for establishing joint causation, i.e., several factors that independently would not cause an injury, but do so in combination. Consider the following scenario:

Example 2 (Forest fire, conjunctive)**.** *Person A drops a canister full of gasoline in the forest, which soaks a tree. An hour later, B smokes a cigarette next to that tree. A and B have joint responsibility for the resulting forest fire ($FF = A \wedge B$). The above definition yields $(A, B, FF) = (1, 1, 1)$ as the sufficient cause.*

In contrast, the traditional but-for test (or *condicio sine qua non*), formulates a necessary condition. A and B on their own are necessary causes, despite the fact that the fire was jointly caused. Sufficient causation can distinguish joint causation, like in this case, from independent causation, like in the case where the forest was dry and both A and B dropped cigarettes independently.

Remark: Similar to how the set of necessary causes for any $O = o$ contains the singleton cause $O = o$, $O = o$ is also a part of each sufficient cause for $O = o$. Hence, it can be filtered out for most purposes.

Modelling control flow

In this section, we discuss how control flow should be incorporated into causal models. Consider the famous *late preemption* example, where Suzy and Billy both throw stones at a bottle. Suzy's stone hits the bottle first and shatters it, while Billy would have hit, had the bottle still been there [13, Example 3.2]. We will discuss several models of this example.

Example 3 (Control flow in equations)**.** *Exogenous variables ST and BT are 1 if Suzy, respectively Billy, throws. The endogenous variable BS is 1 (bottle shatters) if $ST \vee BT$.*

Pearl and Halpern propose a slightly modified model of this situation which captures the relationship between the bottle being shattered and the bottle being hittable by Billy explicitly [17]. However, the relationship between the two is fixed in the model. Datta, Garg, Kaynar and Sharma observe this and therefore introduce exogenous variables to determine which stone reaches the bottle first depending on the context [7].

Example 4 (Control flow in context)**.** *Exogenous variables ST and BT are 1 if Suzy, respectively Billy throws, and R is 1 if Suzy's throw reaches the bottle first. Then $BS \in \mathcal{V}$ is ST if $R = 1$, and otherwise BT.*

Pearl and Halpern's solution can be transferred to the case where the order is not fixed a priori by explicitly representing the temporal order in distinguished variables.

Example 5 (Control flow in variables)**.** *Exogenous variables T_1 to T_n with range $\{ S, B, N \}$ model whether Suzy, Billy or no-one throws a stone at point i. Endogenous variable BS_i is 1 if $BS_{i-1} = 0$ and $T_i \neq N$. $BS = 1$ if $BS_i = 1$ for any i.*

BS_i models whether the bottle is available for hitting at point $i + 1$, similar to the concept of control flow variables in programming. In programming, control flow is the order in which statements are evaluated. Interpreting the above model as a program, BS_i controls whether BS is assigned 1, similar to n nested if-statements surrounding an assignment $BS := 1$.

Control flow should be modelled within variables. Among these three solutions (Examples 3, 4 and 5), only the second and third solutions are able to capture preemption, as preemption is about the temporal relation between events, and control flow captures just that. Without examining the equations themselves, it is not possible to distinguish this order, as Example 3 demonstrates. The second solution is not always sufficient, since control flow is often determined by data flow, e.g., when branching on a variable. Hence, preemption can be accounted for by modelling control flow and data flow separately, at least in scenarios where these concepts are meaningful, e.g., in programs. Even for distributed systems, once the scheduler is made explicit, the system can be adequately modelled as a program and hence control flow can be distinguished. In the following, we will demonstrate this point on a number of other examples (which admittedly have little to do with programs, but we want to stay close to the literature.

They can be recast easily, imagine, e.g., Suzy's and Billy's stone being illegal instructions in a message queue). Note that there might be ways of modelling preemption which do not fall in any of these three classes, however, all examples we are aware of follow one of these three paradigms. We formalise our assumptions on control flow as follows.

Definition 5 (Causal model with control flow)**.** *We say an acyclic causal model $M = (S, \mathcal{F})$ over a signature $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ has control flow variables $\mathcal{V}_{rch}$ if*

1. *$\mathcal{V}$ can be partitioned into $\mathcal{V}_{rch}$ and a set of data variables $\mathcal{V}_{dat}$,*
2. *$\mathcal{R}(V^{rch}) = \{ \top, \bot \}$ for any $V^{rch} \in \mathcal{V}_{rch}$, and,*
3. *for G_{ctl} the subgraph of M's causal network containing all nodes in $\mathcal{V}_{rch}$ and all edges between them, and for all contexts $\vec{u}$ and data variable assignments $\vec{v}$, if all parents $\vec{V}_p^{rch}$ of a node $V^{rch} \in G_{ctl}$ are set to $\bot$, this node is also set to $\bot$, i.e., if $(M, \vec{u}) \vDash [\vec{\mathcal{V}_{dat}} \leftarrow \vec{v}](V_p^{rch} = \bot)$ for each parent $\vec{V}_p^{rch}$, then $(M, \vec{u}) \vDash [\vec{\mathcal{V}_{dat}} \leftarrow \vec{v}](V^{rch} = \bot)$.*

The active variables in G_{ctl}, i.e., those equal to $\top$, represent the current control flow. Data flow variables are assigned depending on which nodes in G_{ctl} are active.

For instance: if we consider structured control flow, G_{ctl} is connected and thus (because M is acyclic) a tree. Furthermore, the set $\{ V^{rch} \mid (M, u) \vDash V^{rch} = \top \}$ always comprises a path. The relationship to non-concurrent imperative programming languages becomes clearer, if we express each function F_V for a data variable $V \in \mathcal{V}_{dat}$ as

$$F_V = \begin{cases} F_{V,V_1^{rch}} & \text{if } V_{pos}^{rch} = V_1^{rch} \\ \quad\vdots & \\ F_{V,V_n^{rch}} & \text{if } V_{pos}^{rch} = V_n^{rch} \\ F_{V,V_p^{rch}} & \text{otherwise} \end{cases}$$

where V_{pos}^{rch} is the current position, i.e., the deepest element of the path $\{ V^{rch} \mid (M, u) \vDash V^{rch} = \top \}$, V_p^{rch} the parent of V_{pos}^{rch}, and $F_{V,V_1^{rch}}, \ldots, F_{V,V_n^{rch}}$ depend only on variables in $\mathcal{V}_{dat}$. Essentially, at each control-flow position, $F_{V,V_i^{rch}}$ may assign a value to V, or otherwise the parent assignment remains in effect.

Intervention on control flow variables. For most (but not all) purposes, we are not interested in control flow variables as parts of sufficient causes. We could filter them out, i.e., if $\vec{X}$ is a sufficient cause, report $\vec{X} \setminus \mathcal{V}_{rch}$. Instead, we chose to achieve this by restricting intervention to a subset of variables $\mathcal{V}^{res} = \mathcal{V}_{dat} = \mathcal{V} \setminus \mathcal{V}_{rch}$. Both approaches model different expectations on the system. We want to assume the control flow to be (locally) determined and not consider failure in, e.g., conditional branching. The same assumption is made by Datta, Garg, Kaynar, Sharma and Sinha (DGKSS) [8] and Beckers [2].

Upon close inspection, this is similar to what DGKSS's definition of causal traces achieves. The process calculus they propose implicitly restricts intervention to the data

flow. On the other hand, it does not support branching, so it is difficult to assess the implications within this calculus. This models different expectations on the system: do we want to assume the control flow to be (locally) determined, or are we considering failure in, e.g., conditional branching. If we never assume failure, we can gather fewer causes, but all of them correspond to a sufficient cause without this assumption.

Altering condition *SF2* in Definition 4, we can model restricted intervention as

SF2' For all $\vec{z}$, $(M, \vec{u}) \vDash [\mathcal{V}^{res} \setminus \vec{X} \leftarrow z]\varphi$, with $\mathcal{V}^{res} = \mathcal{V} \setminus \mathcal{V}_{rch}$.

Theorem 1. *All (minimal) sufficient causes $\vec{X}$ with restriction subsume a (minimal) sufficient cause without restriction.*

Proof. SF2' is equivalent to

$$\begin{aligned} &(M, \vec{u}) \vDash [\mathcal{V} \setminus \{ \vec{X} \cup \mathcal{V}_{rch} \} \leftarrow z]\varphi \forall z \\ \iff &(M, \vec{u}) \vDash [\mathcal{V} \setminus \{ \vec{X} \cup \mathcal{V}_{rch}^{min_X} \} \leftarrow z]\varphi \forall z \end{aligned}$$

for some minimal set of control flow nodes $\mathcal{V}_{rch}^{min_X}$. Given that $\vec{X}$ is minimal w.r.t. *SF2'*, and hence $\vec{X} \cap \mathcal{V}_{rch} = \emptyset$, this sufficient cause $\vec{X} \cup \mathcal{V}_{rch}^{min_X}$ is also minimal w.r.t. *SF2*, since for any subset $\vec{X'} \subsetneq \vec{X} \cup \mathcal{V}_{rch}^{min_X}$, s.t. $(M, \vec{u}) \vDash [\mathcal{V} \setminus X' \leftarrow z]\varphi \forall z$,

$$\begin{aligned} &(M, \vec{u}) \vDash [\mathcal{V} \setminus \{ \vec{X'}|_{\mathcal{V}^{res}} \cup V_{rch} \} \leftarrow z]\varphi \ \forall z \\ \iff &(M, \vec{u}) \vDash [\mathcal{V}^{res} \setminus \vec{X'}|_{\mathcal{V}^{res}} \leftarrow z]\varphi \ \forall z. \end{aligned}$$

Since, $\vec{X'}|_{\mathcal{V}^{res}} \subsetneq \vec{X}$ by minimality of $\mathcal{V}_{rch}^{min_X}$, this contradicts the minimality of $\vec{X}$ w.r.t. to *SF2'*. □

The converse does not hold, consider, e.g., Halpern's extension of the conjunctive forest fire example [13, Example 3.6], where three different mechanism can produce the fire, depending on whether $A \wedge B$, $\neg A \wedge B$ or $A \wedge \neg B$ is the case. In this case, the possibility that all of these mechanisms fail, even if $A \vee B$, leads to a third cause containing both A and B (see Table 1, Forest fire, disjunctive, ext.).

In summary, forbidding intervention on control flow variables may lose some granularity in the distinction of causes, but is often appropriate, when the control flow is assumed to be infallible. Notice however, that none of these notions captures the fact that what ever mechanism produces the fire in case $\neg A \wedge B$ or $A \wedge \neg B$ was not related to its actual coming about. This will be the subject of the next two sections.

What is control flow outside computer programs? For a computer program written in an imperative language, control flow is widely understood to be the order in which statements are executed, e.g., a sequence of line numbers. This definition can be easily extended to distributed systems, however, many examples in the causality literature discuss human agents in physical interaction. Our main objective is a reliable notion of causality for distributed systems, but we also want it to be grounded in the existing body of work on causality. To this end, we make the modelling principles we adhered to explicit. We neither claim that these modelling principles are universal, nor that

control-flow is a notion that can be defined in all scenarios where causality applies. They applied, however, to the 34 examples we found in the literature, as we will see.

We assume the modeller has an intuition of how a variable assignment translates to an (intuitive) 'course of events', and when a 'course of events' should be considered equivalent to another. As discussed in the previous paragraph, we restrict intervention to data flow variables. Hence, we consider only variable assignments $\vec{\mathcal{V}} = \vec{v}$, that result from an intervention on the data flow variables, but not control flow variables, i.e., there is a context $\vec{u}$ and data variable assignment $\vec{v}_{dat}$ s.t. $(M, \vec{u}) \models [\vec{\mathcal{V}_{dat}} \leftarrow \vec{v}_{dat}](\vec{\mathcal{V}} = \vec{v})$. For brevity, we call these assignments *valid*.

1. Each control flow variable V^{rch} should correspond to a relevant event, and vice versa.
2. For every valid assignment, V^{rch} should be $\top$ if the corresponding event occurred.
3. For every valid assignment, V^{rch} should be $\bot$ if the corresponding event did not, or *did not yet*, occur.
4. A control flow variable should only be a parent of another control flow variable (in G_{ctl}) iff the occurrence of the event corresponding to the child depends on the occurrence of the event corresponding to the parent.

Example 6 (Early preemption)**.** *Victoria's coffee is poisoned by her bodyguard ($B = 1$), but before the poison takes effect, she is shot by an assassin ($A = 1$). She dies ($D = 1$).*

Following modelling principles 1 and 2, we introduce at least the control flow variables P_r, S_r and PE_r, which, if set to $\top$, represent the events 'Victoria is poisoned', 'Victoria is shot' and 'the poison takes effect', respectively. The poison can only take effect if Victoria was not shot, but, modelling principle 3 dictates that $S_r = \bot$ only means that she was not *yet* shot, i.e., it might just be that not enough time has passed, but she will eventually get shot before the poison takes effect. We thus need to introduce a fourth control-event, 'Victoria was not shot during the time the poison needs to take effect', represented by NS_r. While $F_{S_r} = (A = 1)$ and $F_{NS_r} = \neg(A = 1)$, and thus all valid assignments result in one being the negation of the other, we will later consider the coming about of a course of events, which includes counterfactuals scenarios where a course of events can be incomplete. Thus, it is possible that, in a counterfactual scenario, $S_r = NS_r = \bot$, which intuitively means that not enough time has passed for the poison to take effect.

Following modelling principle 4, PE_r is a child of both P_r and NS_r. The poison takes only effect ($PE_r = \top$) if it was administered ($P_r = \top$) and some time has passed without Victoria getting shot ($NS_r = \top$). Following the same principle, PE_r is *not* a child of S_r, as $S_r = \bot$ could mean that Victoria is not getting shot at all, but also that she is *not yet* getting shot. Hence, $F_{PE_r} = \neg S_r \wedge P_r$ would be incorrect, as the poison may or may not take effect due to the shot occurring later. Note that the control-flow graph is not linear in this case, representing the independence of poisoning and the shooting.

Preserving control flow

We present further examples that are problematic for existing definitions of cause in literature, followed by a new definition of cause (using control flow) that handles all these examples and many others. Consider the following example known as 'bogus prevention' [18, 20].

Example 7 (Bogus prevention, branching)**.** *An assassin has a change of heart and refrains from putting poison into the victim's coffee ($P = 0$). Later, the bodyguard puts antidote into the coffee ($A = 1$). Is putting the antidote into the coffee the reason the victim survives ($S = 1$)? Let $\mathcal{U} = \{ U_P, U_A \}$, $\mathcal{V} = \{ P, A, S \} \cup \{ P_r, NP_r, N_r \}$ and the following equations describe a causal model M_{bogus} with control flow variables $\{ P_r, NP_r, N_r \}$.*

$$P = U_P \qquad A = U_A$$
$$NP_r = \neg(P = 1) \qquad P_r = (P = 1)$$
$$N_r = P_r \wedge (A = 1) \qquad S = \begin{cases} 1 & \textit{if } NP_r = \top \\ 1 & \textit{if } N_r = \top \\ 0 & \textit{otherwise} \end{cases}$$

Here, P_r and NP_r model the control flow after a conditional checking if the poison was or was not administered. In the positive branch, P_r is set, and only there N_r can be reached, namely if the antidote was given and thus the poison neutralized. S is set to 1 once the poison was neutralized or if it was not administered.

This example is known to be problematic for the counterfactual approach to causation. For instance, in Halpern's modelling [13, Example 3.4], the bodyguard putting in antidote is part of a cause. Similarly, it is part of a sufficient cause, when intervention is not restricted, or, if intervention is restricted, the absence of the poison is not part of the sufficient cause anymore. Together with Hitchcock, Halpern argues that this cause could be removed by considering normality conditions [16]. Blanchard and Schaffer criticise 'under-constrained unclarities' of this approach, calling theorists to 'pay more attention to what counts as an apt causal model [..] before adding more widgets into causal models' [5].[1]. Putting it simply: it is often unclear what is normal. Halpern and Hitchcock [16, p. 26] provide an ad-hoc solution by adding a variable representing the chemical reaction neutralizing the poison, basically introducing the control flow variable N_r which is true if the control flow reached a point where the poison was previously administered ($P_r = \top$) and the bodyguard pours antidote into the coffee.

Intuitively, the antidote should not be considered a cause of the victim's surviving because it was irrelevant within the *actual* course of events. We can capture this through the actual value of the control flow variables. DGKSS [8] and Beckers [2] require interventions to be consistent with the actual temporal order in which events occurred. Translated to our setting, this is akin to considering $\vec{X}$ s.t. for all $\vec{z}$ $(M, \vec{u}) \vDash [\mathcal{V}^{res} \setminus \vec{X} \leftarrow \vec{z}](C = \vec{\top} \implies \varphi)$, where C is the actual control flow $C := \{ V \in \mathcal{V}_{rch} \mid$

[1] They also provide a more intuitive account of bogus prevention, but unfortunately, it only applies to Hitchcocks's definition [19], which has other shortcomings [17, Example 4.4].

$(M, \vec{u}) \models V = \top\}$, and $\vec{\top}$ is a sufficiently long sequence consisting only of $\top$. But often, the coming about of the actual course of action is as important as φ itself.

Example 8 (Agreement)**.** *A and B vote. If $A = B$, an agreement is reached (signified by $R \in \mathcal{V}_{rch}$ with $F_R = (A = B)$), and the outcome is announced ($O = A$ if $R = 1$ and otherwise $\bot$). A and B agree on v in actuality.*

If the control flow is fixed, then $A = v$ by itself is a sufficient cause of $O = v$, despite the fact that A and B need to agree to produce any outcome, as $\vec{z}$ needs to set B to v in order to preserve $R = 1$. This illustrates that the actual control flow should not be presumed a priori to the cause. On the other hand, if φ is established early on (and monotone in time, e.g., violations of safety properties like weak secrecy and authentication [1]), there is no need to find causes for the control flow after φ occurred.

Thus we propose the following definition of sufficient cause, which forbids deviation from the actual control flow and is specific to causal models with control flow variables.

Definition 6 (Control flow preserving sufficient cause)**.** *For a causal model M with control flow variables $\mathcal{V}_{rch}$, let $C \subseteq \mathcal{V}_{rch}$ be the actual control flow in context $\vec{u}$, i.e., the set of control nodes set to $\top$. Then, a* control flow preserving sufficient cause *(CFPSC) is defined like a sufficient cause (see Definition 4), but with SF2 modified as follows:*

CFS2. $\forall \vec{z}.(M, \vec{u}) \models \left[(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \mathcal{V}^{res} \setminus \vec{X} \leftarrow \vec{z}\right] \varphi.$

This definition handles Example 8 correctly: $(A, B, O) = (v, v, v)$ is the only CFPSC, capturing the fact that the agreement needs to be reached in order for v to be announced. (A, O) by itself is not a CFPSC, witnessed by $\vec{z}$ setting B to $v' \neq v$ which results in $R = \bot$.

For *Bogus prevention* (Example 7), which was the motivation for Halpern and Hitchcock's introduction of normality conditions and could previously – in its original formulation [18] – only be treated by means of normality conditions, our approach provides a direct treatment. As all control flow nodes except $C = (NP_r)$ are fixed to zero, A is not causally relevant. Intuitively, the point at which A matters because the poison was administered (P_r) is not available for any counter factual. The actual control flow C, where the poison is not administered, guarantees the victim's surviving ($S = 1$), but needs to be established by *not* administering the poison ($NP_r = \top$). Hence (P, S) is (the only) CFPSC, given that $(M, \vec{u}) \models [(P_r, N_r) \leftarrow (\bot, \bot), A \leftarrow z]S = 1$ for $z = 1, 2$.

Late preemption (Example 5 with control flow variables BS_i, $i \in \mathbb{N}_n$) is also handled correctly; here (T_1, BS) is the only CFPSC, i.e., Suzy's throw caused the bottle to shatter, but not Billie's. Because $C = (BS_1)$, BS_2 is set to $\bot$ and Billie's throw has no bearing on the outcome.

Early preemption, where the victim is poisoned, but shot before the poison takes effect (Example 6, can be captured similarly, e.g., considering the model described by

Hitchcock [20, p. 526], or our own adaptation (cf. Table 1). Intuitively, C captures the control flow where the victim is shot but the poison has not yet taken effect. Setting its complement to $\bot$ correctly disregards the control flow representing the poison taking effect due to the victim not being shot.

Related work

We first discuss related work on sufficient causation, which is the basis for CFPSC, then related work concerning control flow and finally link our insights on control flow to the notion of structural contingencies in the literature.

Sufficient causation Going back to Lewis [23], most definitions of actual causation investigate claims of the form 'had A not occurred, B would not have occurred'. The notions put forward by Pearl and Halpern [13, 17] follow this idea, which arguably captures a form of *necessary* causation. We elaborate this point at the end of this section.

By contrast, Datta, Garg, Kaynar, Sharma, and Sinha aim at capturing minimal sequence of protocol actions *sufficient* to provoke a violation, in order to provide a tool for forensics as well as a building block for accountability [8]. The appeal of sufficient causation is that there is a clear interpretation of what it means for A to be part of a sufficient cause (A, B): A is (jointly with B) causing the event. Notions of necessary causation typically lack this kind of interpretation and require secondary notions like blame to determine joint responsibility. Furthermore, in particular in the context of distributed systems and program analysis, it comes in handy for debugging and forensics that each sufficient cause basically captures a chain of events which, on its own, leads to an outcome [7].

However, necessary causes are more succinct and seem to capture what is meant by A causes B in natural language better. We suspect the latter is because natural language often uses "cause" to mean "part of a cause", but weighing these two notions against each other is not the scope of this work, and more ever, depends on the application one has in mind. We are interested in the coming about of events, so we focus on sufficient causation.

DGKSS's notion of causation [8] is based on a source code transformation of non-branching programs within a simple process calculus. We therefore cast their idea of intervening on events that do *not* appear in the candidate cause within Pearl's framework (see Definition 4). This methodology allows us to understand and generalize effects implicit in the definition of the calculus and translate them back. Sufficient causes are different from DGKSS's cause traces in that the latter yield entire traces and that intervention is performed on code instead of variables. DGKSS's calculus fixes the control flow to the actual control flow, since their cause traces contain the line number of the statement effectuating each event. Every intervention on these cause traces (which roughly corresponds to the z in SF2) needs to contain these line numbers in the same order and is disregarded otherwise. Example 8 was the motivation to depart from this and capture the coming about of the actual control flow.

Besides DGKSS's work, there is only little work on sufficient causation formalizing which events are jointly sufficient to cause an outcome. Going back to early attempts of formulating actual causation in a purely logical framework [24], Halpern [12] formulates the NESS test [29] within Pearl's causality framework. As with the NESS test, this notion only allows for singular causal judgements ([12, Theorem 5.3]) and is thus not suited for capturing joint causation. Halpern [14] also proposed a notion of sufficient causes which requires a sufficient cause to be a) part of all actual causes (hence inducing three variants of the definition, for each notion of actual causation put forward) and b) to ensure φ for all contexts. If contingencies are restricted to control flow variables and causes to data flow variables, the first condition is very similar to CFPSC. By way of the second condition, one avoids that φ appears in all sufficient causes (as it is a trivial actual cause of itself), however, this condition prohibits capturing joint causation in cases where rare external circumstances (e.g., strong wind making lighting the cigarette in Example 2 impossible) could have prevented the outcome altogether, although they actually did not. For distributed systems, this is almost always the case due to possible loss of messages in transition, hence we consider this criterion too strong for these purposes.

Causation and control flow Instead of control flow, Beckers extends structured equations with time [2, Part II]. His proposal for actual causation involves a notion very similar to the NESS criterion. The notion derived from it does not capture joint causes, but captures many preemption examples.

Sharma's thesis extends DGKSS's model with some control flow: the $\oplus$ operator can capture choices the parties make, e.g., $V := 0 \oplus 1$ lets the party set V to either 0 or 1 [26]. As agents can still not branch, the previous discussion on DGKSS's paper applies.

Besides Beckers and DGKSS, there are other formalisms for causal models that include control flow, but do not aim at capturing actual causality [9, 27]. We purposefully formulated control flow within Pearl's causality framework, to compare with existing definitions and provide insight into how control flow can guide the modelling task and improve definitions.

Relation to structural contingencies in Halpern 2015 We review Halpern's modification [13] of Halpern and Pearl's definition of actual causes [17]. First, because it is well-known, second, because it employs a secondary notion called *structural contingencies*, which appears to be related to control flow. We a) give evidence that contingencies relate to control flow wherever they are successful, b) show that they are problematic if data flow is involved, and c) provide an interpretation of structural contingencies in terms of control flow, supporting the argument that preemption is first a modelling problem, which should be covered by distinguishing control flow from data flow, and then a matter of the definition of a cause.

Definition 7 (Review: actual cause)**.** $\vec{X} = \vec{x}$ *is an* actual cause *of* φ *in* $(M, \vec{u})$ *if the following three conditions hold.*

AC1 and AC3. Just like SF1 and SF3.

AC2. There are $\vec{W}$, $\vec{w}$ and $\vec{x}'$ such that $(M, \vec{u}) \models (\vec{W} = \vec{w})$, and $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w},]\neg\varphi$.

AC2 is a generalisation of the intuition behind Lewis' counterfactual. The special case $\vec{W} = ()$ corresponds to Lewis' reasoning that $\vec{X} = \vec{x}$ is *necessary* for φ to hold, because there exists a counterfactual setting $\vec{x}'$ that negates φ. AC2 weakens this condition in order to capture causes that are 'masked' by other events. The set of variables $\vec{W}$ is called *structural contingency*, as it captures the aspects of the actual situation under which $\vec{X} = \vec{x}$ is a cause. Variables in $\vec{W}$ can only be fixed to their actual values.

We observe that in all examples in Halpern's paper where non-empty contingencies appear (Suzy-Billy, Ex. 3.6 and Ex. 3.7 after adding variables), they exclusively contain variables added to the model to "describ[e] the mechanism that brings about the result". These are the precisely variables we would consider control flow variables. The only exceptions are Example 3.9b and 3.10, where Halpern points out that Definition 7 gives unintuitive results. Definition 6, captures Halpern's intuition correctly (see Table 1, Train [10] and Careful Poisoning).

While structural contingencies work well if they contain control flow variables, they can give unintuitive results if they contain data flow variables. Consider the following causal model M_{BoS} inspired by the 'Battle of Sexes' two-player game.

Example 9 (Bach or Stravinsky)**.** *A couple ($P_1, P_2 \in \mathcal{V}$) agreed on meeting this evening, but they cannot recall whether they wanted to attend a Bach or a Stravinsky concert ($\mathcal{R}(P_1) = \mathcal{R}(P_2) = \{ B, S \}$). They are taking the train ($T = 1$), but only if they both go to the same concert ($F_C = P_1 = P_2$ for the control flow variable C). Contrary to the awkward situation in the famous two-player game, they have left a note $N \in \mathcal{U}, \mathcal{R}(N) = \{ B, S \}$ in their calendar, which helps them remember ($F_{P_1} = N$ and $F_{P_2} = N$). Is the fact that the note reminds them to attend the Bach concert ($N = B$) a cause for taking the train together ($T = 1$)?*

Definition 7, as well as the definition preceding it [17], lead to an unintuitive result because variables that concern data flow are considered as contingencies. No matter which value is chosen for N, T is always 1. However, $\vec{W}$ can be set to P_1 or P_2, fixing it to the actual value of N. Hence, for, e.g., $\vec{u} = (B)$, it holds that $(M_{BoS}, \vec{u}) \models [N \leftarrow S, \vec{W} = (P_2) \leftarrow B](T \neq 1)$, and thus the choice of the input is considered a cause for $T = 1$, although 1 is output no matter what the input is. It appears that P_1 and P_2 should not be permissible contingencies. Definition 6 handles this example correctly: (P_1, P_2, T) is the only CFPSC, as the control flow (C) is preserved no matter what value N has, as long as P_1 and P_2 agree on doing what it says (see Table 1, BoS). Our conclusion is that contingencies should be restricted to control flow variables to avoid such spurious causes.

Thus, if the use case allows for making control flow variables explicit, we can restrict structural contingencies $\vec{W}$ to control flow variables and actual causes $\vec{X}$ to data flow variables, and consider AC2' as follows:

AC2'. For $\vec{X} \subseteq \mathcal{V}^{res}$, $C \subseteq \mathcal{V}_{rch}$ and $(M, \vec{u}) \models \vec{C} = \vec{c}$ a subset of the actual control flow, there is $\vec{x}'$ such that $(M, \vec{u}) \models [\vec{C} \leftarrow \vec{c}, \vec{X} \leftarrow \vec{x}']\neg\varphi$.

Under these circumstances, we can relate actual causes and CFPSC by comparing AC2' to CFS2. A priori, both are very different: AC2 (and AC2') formulate a necessity criterion on $\vec{X}$,[2] while CFS2 formulates a sufficiency criterion. Interestingly, there is a duality between necessary and sufficient causes [22], which we can use to compare the two w.r.t. their treatment of control flow: If the set of variables is finite, the set of all sufficient causes can be obtained from the set of all necessary causes $\mathcal{X} = \{ \vec{X}_1, \ldots, \vec{X}_m \}$ by, first, considering them as a boolean formula in $\vec{X}$ in disjunctive normal form (DNF) ($\vec{X}$ is in $\mathcal{X}$ iff either $\vec{X}$ equals $\vec{X}_1$, or if it equals $\vec{X}_2$, etc.), second, computing the conjunctive normal form of this formula and, finally, switching $\wedge$ and $\vee$. For example, (A, B) is the only sufficient causes in the conjunctive forest fire example, and thus A and B are both necessary causes. We adapted this result to CFPSCs (see the Appendix for theorem and proof) and obtain a dual definition of control flow preserving *necessary* causes, where CFS2 translates to

CFN2. For $\vec{X} \subseteq \mathcal{V}^{res}$ and for $C \subseteq \mathcal{V}_{rch}$ the actual control flow, there is $\vec{x}'$ such that

$$(M, \vec{u}) \vDash [(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \vec{X} \leftarrow \vec{x}'] \neg \varphi.$$

We can now compare AC2' to CFN2 to get a clear picture of how actual causation handles control flow, if we incorporate the distinction of control flow variables and data flow variables as discussed above. AC2' is much more liberal in how the counterfactual control flow can be related to the actual control flow. By choosing an arbitrary subset of all control flow variables, not only those set to $\top$ or $\bot$, each counterfactual setting of a control flow variable may enforce the actual control flow (if it is fixed to $\top$), prevent counterfactual flow that contradict the actual course of events (if it is fixed to $\bot$), but may also just be computed based on the equations (if it is not part of the subset). CFN2 is more rigid in this respect, strictly prohibiting the counter-factual control flow to deviate from the actual control flow, but leaving it otherwise free (motivated by Example 8). This explains, e.g., the difference in Weslake's Careful Poisoning example (see Table 1), where the assassin only adds poison to the coffee if he is sure the antidote was added previously. The antidote is (wrongly to most) considered an actual cause for the victim surviving, as the counterfactual where it is not administered can still consider the control flow where the assassin added the poison.

We summarize: a restriction of *structural contingencies* to control flow seems to avoid unintuitive results in some cases, without losing accuracy on any examples we considered. Given this restriction, contingencies obtain an interpretation in terms of the relation between the actual control flow and the control flow considered in counterfactuals. In comparison to CFPS2, this relation is much more loose. The Careful Poisoning example suggests that it is too loose.

Validation

Our goal was to provide an account of the relation between control flow and causation; Definition 6 served this goal by demonstrating that an explicit treatment of control flow

[2] If $\vec{X} = \vec{x}$ is an actual cause under contingency $\vec{W} = \vec{w}$, then $\vec{X}' = \vec{X} \cdot \vec{W} = \vec{x} \cdot \vec{w}$ is a necessary cause, i.e., $(M, u) \models [\vec{X}' \leftarrow \vec{x} \cdot \vec{w}] \neg \varphi$.

	Definition 4	Definition 7	Definition 6
Forest fire, Ex. 2	(A, B, FF)	$(A), (B), (FF)$	(A, B, FF)
— disjunctive (overdet.) [11, p. 278]	$(B, O), (S, O)$	$(B, S), (O)$	$(B, O), (S, O)$
— disjunctive, ext. [13, Ex. 3.7]	(MD, L, C, FF), (MD, B, C, FF), (L, A, C, FF)	$(MD), (L), (C), (FF)$	(MD, L, FF)
Late preemption, Ex. 5	$(T1, BS1, BS)$, $(T2, BS1, BS2, BS)$	$(T1), (BS1), (BS)$	$(T1, BS)$
Early preemption [20, p. 526]	$(A, D_1, D_2), (B, P, D_2)$	$(A), (D_1), (D_2)$	(A, D_2)
— (ctl), Ex. 6[a]	(A, S_r, D), $(B, P_r, S_r, PE_r, NS_r, D)$	$(A), (S_r), (D)$	(A, D)
Bogus prevention [18]	$(P, S), (A, S)$	$(P, A), (S)$	$(P, S), (A, S)$
— ad-hoc [16, p. 29]	$(P, S), (A, S, PN)$	$(P), (S)$	(P, S)
— (ctl), Ex. 7	(P, S, NP_r), (A, S, P_r, NP_r, N_r)	$(P), (S), (NP_r)$	(P, S)
— (ctl, reversed)[b]	(P, D, NP_r), (A, D, NP_r, P_r, PN_r)	$(P), (D), (NP_r)$	(P, D)
Careful Poisoning [28, Ex. 11]	$(A, D), (P, D)$	$(A), (D)$	$(A, D), (P, D)$
— (ctl)[c]	(A, NA_r, P_r, D), (NA_r, A_r, P_r, P, D)	$(A), (NA_r), (P_r), (D)$	(D)
Train [15, Ex. 4]	$(F, RB, A), (LB, RB, A)$	$(F, LB), (RB), (A)$	(F, RB, A), (LB, RB, A)
— [10]	(F, RT, A)	$(F), (RT), (A)$	(F, A)
— (ctl)[d]	$(F, R_r, A), (L_r, R_r, A)$	$(F), (R_r), (A)$	(F, A)
Prisoner [21]	(C, D)	$(C), (D)$	(C, D)
Backup [28, Ex. 1]	$(T, V), (S, V)$	$(T), (V)$	$(T, V), (S, V)$
— (ctl) [28, Ex. 1]	(T, V, T_r), (S, V, T_r, NT_r, S_r)	$(T), (V), (T_r)$	(T, V)
Command [28, Ex. 8]	(M, C)	$(M), (C)$	(M, C)
Agreement, Ex. 8	(A, B, R, O)	$(A), (B), (R), (O)$	(A, B, O)
BoS, Ex. 9	(P_1, P_2, C, T)	$(N), (P_1), (P_2), (C), (T)$	(P_1, P_2, T)
Switch [28, p. 16]	$(S, L2, I), (L1, L2, I)$	$(S), (L2), (I)$	$(S, L2, I)$, $(L1, L2, I)$
Combination Lamp [28, p. 19]	(B, C, L)	$(B), (C), (L)$	(B, C, L)
Shock [28, p. 17]	$(B, C, C1)$	$(A), (B), (C), (C1)$	(B, C)
Push A [28, p. 26]	(P, B, H, D), (P, T, H, D)	$(P), (B, T), (H), (D)$	(P, B, H, D), (P, T, H, D)
— (ctl)[e]	(P, T, P_r, H, D)	$(P), (T), (P_r), (H), (D)$	(P, T, H, D)
Push B [28, p. 26]	(P, T, H, D)	$(P), (T), (H), (D)$	(P, T, H, D)
Fancy Lamp [28, p. 31]	$(A, N3, L), (B, N1, L)$	$(A, B), (A, N1), (B, N3), (N1, N3), (L)$	$(A, N3, L)$, $(B, N1, L)$
Vote [13, Ex. 4.1]	$(V_1, M, P), (V_2, M, P)$	$(V_1, V_2), (M), (P)$	(V_1, M, P), (V_2, M, P)
Ranch [13, Ex. 3.7]	(A_1, A_2, M_1, O)	$(A_1), (A_2), (M_1), (O)$	(A_1, A_2)
Vote 5:2[f]	$(V1, V2, V3, V4, O)$, $(V1, V2, V3, V5, O)$, $(V1, V2, V4, V5, O)$, $(V1, V3, V4, V5, O)$, $(V2, V3, V4, V5, O)$	$(V1, V2)$, $(V1, V3)$, $(V1, V4)$, $(V1, V5)$, $(V2, V3)$, $(V2, V4)$, $(V2, V5)$, $(V3, V4)$, $(V3, V5), (V4, V5), (O)$	$(V1, V2, V3, V4)$, $(V1, V2, V3, V5)$, $(V1, V2, V4, V5)$, $(V1, V3, V4, V5)$, $(V2, V3, V4, V5)$
Pollution, $k = 80$ [13, Ex. 3.11]	(A, D)	$(A), (D)$	(A, D)
Pollution, $k = 50$ [13, Ex. 3.11]	$(A, D), (B, D)$	$(A, B), (D)$	$(A, D), (B, D)$
Pollution, $k = 120$ [13, Ex. 3.11]	(A, B, D)	$(A), (B), (D)$	(A, B, D)

[a] Model with $\mathcal{V}_{rch} = \{ P_r, S_r, PE_r, NS_r \}$ and $P_r = B$, $S_r = A$, $NS_r = \neg A$, $PE_r = P_r \wedge \neg A$ and $D = 1$ iff $S_r = \top$ or $PE_r = \top$.

[b] Like Ex. 7, but reversed control flow: $D = 0$ iff $NP_r = \top$ (poison not administered) or $PN_r = \top$ (poison neutralised).

[c] Model with $\mathcal{V}_{rch} = \{ NA_r, A_r, P_r \}$ and $NA_r = \top$ iff $A = 0$, $A_r = \top$ iff $A = 1$, $P_r = \top$ iff $NA_r = \top \wedge P = 1$ and $D = 1$ iff $P_r = \top$.

[d] Model with $\mathcal{V}_{rch} = \{ L_r, R_r \}$, where $L_r = \top$ iff $\neg F$, $R_r = \top$ iff F and $A = 1$ iff $L_r = \top$ or $R_r = \top$, allowing train to get stuck.

[e] Model with $\mathcal{V}_{rch} = \{ P_r, NP_r \}$, where $P_r = \top$ iff $P = 1$, $NP_r = \top$ iff $P = 0$ and $H = 1$ iff either $P_r = \top \wedge T = 1$ or $NP_r = \top \wedge B = 1$.

[f] Majority vote with 7 participants, 5 of which vote Yea, highlighting the difference between notions based on sufficiency and necessity: 4 Yeas suffice for $O = 1$, but if two voters would switch to Nay, the vote would be overturned.

Table 1: Set of causes for various examples from the literature. The suffix (ctl) marks models adapted to Definition 5.

can help (and is sometimes even necessary) to treat cases where counterfactuals can change the course of events, e.g., in cases of preemption. We further validate our definition against a benchmark suite of 34 examples from the literature. To avoid cherry-picking, we include all examples from [28] and [13]. The source code is available at `https://github.com/rkunnema/causation-benchmark`. We encourage other researchers to use this suite to test their own definitions of causality and to extend it with new examples.

We compare Definition 6 with Definition 4 for illustration, and with Halpern's notion of actual causation (see Definition 7) as a point of reference. We omit notions like defaults and normality, as we want to stress that these are not necessary to deal with examples of preemption, and favour Halpern's notion over prior variants [17], as it distinguishes between joint causation and over-determination.[3]

We include the *complete* set of causes as a comma-separated list of sequences, as, e.g., for Careful Poisoning (ctl) it is important to see that A (the antidote being administered) is *not* a CFPSC. Many examples in the literature do not capture control flow in their modelling, in which case we present results for the original modelling and a modelling according to Definition 5. In all examples we get results that are satisfying according to the discussion in the literature we cite.[4] For lack of space we refer to the cited literature for deeper explanation. The only examples we added (not counting adaptations to Definition 5) are Examples 8 and 9, which we explained already, and Vote 5:2, which contrasts the differing views sufficient causes and actual/necessary causes have to offer.

Conclusion and future work

In cases where control flow can be made explicit, it is worth doing so, as this helps to deal with problematic cases like preemption. We discussed in what way it should be taken into account, and proposed a blue-print for doing so, as well as a definition of causality that preserves the actual course of events. This definition is simple and intuitive, does not require secondary notions, captures joint causation and handles all 34 example in our benchmark correctly (with respect to the respective author's views). Such a definition is useful for computer programs and distributed systems, as the temporal order of events between communicating agents can be captured by a non-deterministic scheduler simulating them. A translation from Petri nets, Kripke structures or process calculi to extended causal models that adheres to the modelling principles discussed in this work can be used to argue the soundness of causality notions formulated within these formalisms. In fact, we advocate this approach, as we believe that a thorough discussion of causality requires a common language.

Vice versa, the causality literature stands to benefit from such translations. Due to the generality of Pearl's framework, modelling is more art than science, which raises concerns about the falsifiability of theories of causation. For bogus prevention and other difficult examples, e.g., late preemption, the 'correct' modelling has been debated

[3] The Litmus test are the two variants of the Forest fire example.

[4] For Switch and Fancy Lamp, our Definition 6 captures that S, respectively, A are not necessary for the outcome by giving a sufficient cause which does not include them.

again and again for each use case specifically. By transferring and analysing existing domain-specific definitions, e.g., DGKSS's definition, to Pearl's framework, we can find a common ground for the discussion of the 'right' modelling, and encourage a similar treatment for other domains where causality is of interest. This way, experiences and observation in well-understood domains can be fed back to the general case and modelling principles be exposed that are otherwise easily overlooked when modelling abstract scenarios ad hoc.

References

[1] Bowen Alpern & Fred B Schneider (1985): *Defining liveness*. *Information processing letters* 21(4), pp. 181–185.

[2] Sander Beckers (2016): *Actual Causation: Definitions and Principles*. Ph.D. thesis, Faculty of Engineering Science, Katholieke Universiteit Leuven. Available at `https://lirias.kuleuven.be/handle/123456789/545701`.

[3] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard Trefler (2009): *Explaining Counterexamples Using Causality*, pp. 94–108. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-02658-4_11. Available at `https://doi.org/10.1007/978-3-642-02658-4_11`.

[4] Giampaolo Bella & Lawrence C. Paulson (2006): *Accountability Protocols: Formalized and Verified*. *ACM Trans. Inf. Syst. Secur.* 9(2), pp. 138–161, doi:10.1145/1151414.1151416. Available at `http://doi.acm.org/10.1145/1151414.1151416`.

[5] Thomas Blanchard & Jonathan Schaffer (2014): *Cause without Default*. In: *Making a Difference*, Oxford University Press, pp. 175–214.

[6] Hana Chockler, Joseph Y. Halpern & Orna Kupferman (2008): *What Causes a System to Satisfy a Specification?* *ACM Trans. Comput. Logic* 9(3), pp. 20:1–20:26, doi:10.1145/1352582.1352588. Available at `http://doi.acm.org/10.1145/1352582.1352588`.

[7] Anupam Datta, Deepak Garg, Dilsun Kaynar & Divya Sharma (2016): *Tracing Actual Causes (CMU-CyLab-16-004)*. Technical Report, Carnegie Mellon University.

[8] Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma & Arunesh Sinha (2015): *Program actions as actual causes: A building block for accountability*. In: *2015 IEEE 28th Computer Security Foundations Symposium*, IEEE, pp. 261–275.

[9] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain & Hudson Turner (2004): *Nonmonotonic causal theories*. *Artificial Intelligence* 153(1-2), pp. 49–104.

[10] Ned Hall (2000): *Causation and the Price of Transitivity*. Journal of Philosophy 97(4), p. 198.

[11] Ned Hall (2004): *Two Concepts of Causation*. In John Collins, Ned Hall & Laurie Paul, editors: Causation and Counterfactuals, MIT Press, pp. 225–276.

[12] Joseph Y. Halpern (2008): *Defaults and Normality in Causal Structures*. In Gerhard Brewka & Jérôme Lang, editors: Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008, AAAI Press, pp. 198–208. Available at http://www.aaai.org/Library/KR/2008/kr08-020.php.

[13] Joseph Y. Halpern (2015): *A Modification of the Halpern-Pearl Definition of Causality*. In Qiang Yang & Michael Wooldridge, editors: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, pp. 3022–3033. Available at http://ijcai.org/Abstract/15/427.

[14] Joseph Y. Halpern (2016): *Actual Causality*. The MIT Press.

[15] Joseph Y. Halpern & Christopher Hitchcock (2011): *Actual causation and the art of modeling*. CoRR abs/1106.2652. Available at http://arxiv.org/abs/1106.2652.

[16] Joseph Y. Halpern & Christopher Hitchcock (2013): *Graded Causation and Defaults*. CoRR abs/1309.1226. Available at http://arxiv.org/abs/1309.1226.

[17] Joseph Y. Halpern & Judea Pearl (2013): *Causes and Explanations: A Structural-Model Approach — Part 1: Causes*. CoRR abs/1301.2275. Available at http://arxiv.org/abs/1301.2275.

[18] Eric Hiddleston (2005): *Causal powers*. British Journal for the Philosophy of Science 56(1), pp. 27–59, doi:10.1093/phisci/axi102.

[19] Christopher Hitchcock (2001): *The Intransitivity of Causation Revealed in Equations and Graphs*. Journal of Philosophy 98(6), pp. 273–299.

[20] Christopher Hitchcock (2007): *Prevention, Preemption, and the Principle of Sufficient Reason*. Philosophical Review 116(4), pp. 495–532.

[21] Mark Hopkins & Judea Pearl (2003): *Clarifying the usage of structural models for commonsense causal reasoning*. In: Proceedings of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, AAAI Press Menlo Park, CA, pp. 83–89.

[22] Robert Künnemann (2017): *Sufficient and necessary causation are dual*. CoRR abs/1710.09102. Available at http://arxiv.org/abs/1710.09102.

[23] David Lewis (1973): *Causation*. Journal of Philosophy 70(17), pp. 556–567.

[24] J. L. Mackie (1965): *Causes and Conditions*. American Philosophical Quarterly 2(4), pp. 245–264.

[25] Judea Pearl (2000): *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA.

[26] Divya Sharma (2015): *Interaction-aware Actual Causation: A Building Block for Accountability in Security Protocols*. Ph.D. thesis, CyLab, Carnegie Mellon University.

[27] Hudson Turner (1999): *A logic of universal causation*. Artificial Intelligence 113(1-2), pp. 87–123.

[28] Brad Weslake (2015): *A partial theory of actual causation*. British Journal for the Philosophy of Science.

[29] Richard W Wright (1987): *Causation, responsibility, risk, probability, naked statistics, and proof: Pruning the bramble bush by clarifying the concepts*. Iowa L. Rev. 73, p. 1001.

Appendix: Duality between CFS2 and CFN2

Fix some finite set $\mathcal{V}^{res} \subset \mathcal{V}$ and some ordering $\{ V_1, \ldots, V_n \} = \mathcal{V}^{res}$ and let $\overline{X}$ denote the following representation of $X \subseteq \mathcal{V}^{res}$ relative to $\mathcal{V}^{res}$: $\overline{X} := (1_X(V_1), \ldots, 1_X(V_n))$. Any set of sets of variables $\mathcal{X} = \vec{X}_1, \ldots, \vec{X}_m$ can be represented as a boolean formula in disjunctive normal form (DNF) that is true whenever $\overline{X}$ is the bitstring representation of $X \subseteq \mathcal{V}^{res}$ such that $X \in \mathcal{X}$: $(\overline{X} = \overline{X}_1 \vee \overline{X} = \overline{X}_2 \vee \cdots \vee \overline{X} = \overline{X}_m)$. Here, $\overline{X} = \overline{X}_i$ merely compares two bitstrings for equality, i.e., it is a conjunction $\bigwedge_{j \in \mathbb{N}_n} \overline{X}|_j = \overline{X}_i|_j$.

Theorem 2 (control flow preserving necessary causes)**.** *For $\mathcal{X}$ the set of (not necessarily minimal) CFPSCs, i.e., adhering to SF1 and CFS2, let $\overline{\mathcal{X}}$ be the DNF representation of $\mathcal{X}$. Then the set of (not necessarily minimal) control flow preserving actual causes, i.e., adhering to AC1 and CFN2, is represented by $\overline{\mathcal{Y}}$, which is obtained from $\overline{\mathcal{X}}$ by transforming $\overline{\mathcal{X}}$ into CNF and switching $\vee$ and $\wedge$. The same holds for the other direction.*

Proof. Fix an arbitrary model M, context $\vec{u}$ and let $C \subseteq \mathcal{V}_{rch}$ be the actual control flow in this context. By Definition 6, CFS2, we can rephrase the assumption $\mathcal{X}$ as follows: For all sequences of variables $\vec{X}$,

$$\vec{X} \in \mathcal{X} \iff \forall \vec{z}.(M, \vec{u}) \models \left[(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \mathcal{V}^{res} \setminus \vec{X} \leftarrow \vec{z} \right] \varphi. \quad (1)$$

As $\mathcal{X}$ is finite, i.e., $\mathcal{X} = \{ \vec{X}_1, \ldots, \vec{X}_n \}$, we can write $\vec{X} \in \mathcal{X}$ as a boolean function over $\{ 0, 1 \}^n$:

$$\vec{X} \in \mathcal{X} \iff (\overline{X} = \overline{X}_1) \vee \cdots \vee (\overline{X} = \overline{X}_m).$$

Like any boolean function, this function can be transformed into canonical CNF and thus the right-hand side can be expressed as $c_1 \wedge \cdots \wedge c_k$ with some conjuncts c_i of form $\bigvee_{j \in \mathbb{N}_n} (\neg)\overline{X}|_j$. Whatever these conjuncts are, we insert this CNF on the left-hand side of (1) and negate both sides. Hence, for all $\vec{X}$,

$$\neg c_1 \vee \cdots \vee \neg c_k \quad \Longleftrightarrow \quad \exists \vec{z}.(M, \vec{u}) \;\vDash\; \left[(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \mathcal{V}^{res} \setminus \vec{X} \leftarrow \vec{z}\right] \neg\varphi.$$

(Note that $c_1, \ldots, c_k$ depend on $\vec{X}$.) We rename $\vec{X}$ to $\vec{Z}$ and $\vec{z}$ to $\vec{x}'$. Let $\{^b/_a\}$ denote b literally replacing a. Thus, for all $\vec{Z}$,

$$\neg c_1 \left\{^{\overline{Z}}/_{\overline{X}}\right\} \vee \cdots \vee \neg c_k \left\{^{\overline{Z}}/_{\overline{X}}\right\} \iff \exists \vec{x}'.(M, \vec{u}) \vDash \left[(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \mathcal{V}^{res} \setminus \vec{Z} \leftarrow \vec{x}'\right] \neg\varphi.$$

We can replace $\mathcal{V}^{res} \setminus \vec{Z}$ by a new variable $\vec{X}$ and quantify over $\vec{X}$ again. This is valid, as $\vec{X} \mapsto \mathcal{V}^{res} \setminus \vec{X}$ is a bijection between the domain of $\vec{X}$ and the domain of $\vec{Z}$. Note that $\mathcal{V}^{res} \setminus (\mathcal{V}^{res} \setminus \vec{X}) = \vec{X}$ and $\overline{Z} = \overline{(\mathcal{V}^{res} \setminus \vec{X})} = \neg\overline{X}$. Thus for all $\vec{X}$

$$\neg c_1 \left\{^{\neg\overline{X}}/_{\overline{X}}\right\} \vee \cdots \vee \neg c_k \left\{^{\neg\overline{X}}/_{\overline{X}}\right\} \iff \exists \vec{x}'.(M, \vec{u}) \vDash \left[(V_v^{rch})_{v \notin C} \leftarrow \vec{\bot}, \vec{X} \leftarrow \vec{x}'\right] \neg\varphi.$$

As each conjunct c_i is a disjunction, the negation of c_i with $\overline{X}$ substituted by $\neg\overline{X}$ can be obtained by switching $\vee$ and $\wedge$. The resulting term is, again, a boolean formula in DNF, so $\overline{\mathcal{X}}$ transforms into $\mathcal{X}$ easily. The reverse direction proceeds with the exact same steps, modulo variables naming. □