From Programs to Causal Models*

Thomas F. Icard

Stanford University, Stanford, CA, USA
icard@stanford.edu

Abstract

The purpose of the present contribution is to explore the consequences of building causal models out of programs, and to argue that doing so has advantages for the semantics of subjunctive conditionals and of causal language. We establish basic results about expressivity and give examples to show both the power of the framework and the ways in which it differs from more familiar causal frameworks such as structural equation models.

1 Motivation

The idea that we represent causal relationships with internal "simulation" models has a long and distinguished history, arguably going back to Hume. Perhaps the most prominent contemporary formalization of this idea involves causal Bayesian networks, which define *generative models* over some fixed set of random variables. While Bayes nets are useful for many purposes, some authors have advocated for a more general formalism, known as structural equation models (SEMs), which explicitly encode functional dependencies among variables and relegate all randomness to so called exogenous variables [12]. Unencumbered by the demand for a non-circular account of causal claims, a number of recent researchers in philosophy, linguistics, and psychology have proposed analyzing the semantics of subjunctive conditionals and other ostensibly causal language by appeal to such causal models [16, 9, 14, 17], in place of the once dominant but more abstract "system-of-spheres" models founded on world-similarity-orderings [10].

SEMs come with a number of advantages. By making causal information explicit, they support a precise notion of *intervention*, which grounds hypothetical and counterfactual claims. They can also be applied to a wider array of phenomena than standard Bayes nets, e.g., by allowing certain kinds of cyclic dependencies among variables, which is purportedly important for semantics [5, 14]. Despite these and other attractions, there is a sense in which SEMs depart from the original idea of a simulation model. A prediction in this framework, counterfactual or otherwise, is determined by a solution to a (generally unordered) system of equations, in line with the kinds of models found in physics, economics, and engineering disciplines. But in general structural equations do not simulate; they describe. While this declarative emphasis may be quite appropriate for many purposes, it is desirable to have a similarly expressive framework that retains the procedural character of a simulation model.

A number of authors in artificial intelligence, and more recently in cognitive science, have proposed an idea very much in this vein, to define simulation models using *arbitrary programs* in some rich programming language [13, 11, 4, 3, 1]. Much of the emphasis in this literature is on defining complex probability models with efficient inference procedures. But some authors have also highlighted the fact that these simulation models, just like Bayes nets, may embody causal structure. Despite this important work, a precise analysis of *programs as causal models* has not been given. The purpose of the present contribution is to establish some of the basic definitions and results, and to motivate the idea for semantics of natural language. Rather than

*Thanks to Noah Goodman, Duligur Ibeling, Dan Lassiter, and Krzysztof Mierzewski for helpful discussions.

offer a specific compositional analysis of counterfactuals or causal claims, the aim is to establish the framework with sufficient precision so that any semantic analysis that invokes a notion of "intervention on a simulation model" can be seamlessly accommodated.

As programs themselves have a causal structure, we can use this very structure as a "semi-iconic" causal representation and, as we shall see, as a representation of other non-causal dependence relations as well. The resulting framework provides an attractive setting for a quite general theory of subjunctive conditionals. Highlighted are two especially notable features.

The first is that the framework affords a simple and intelligible way of capturing quantificational and more generally "open-world" reasoning [13, 11], whereby counterfactual suppositions alter which (and even how many) individuals (or other variables) are being considered. The following example is inspired by one from Kaufmann [9, 1164]:

Example 1. Imagine a number of students have shown up to take an exam, and that students typically forget to bring their own pencils. Suppose we say that a student is prepared for the exam just in case either they brought their own pencil, or there are enough pencils for everyone who needs one. (If there are too few, out of fairness no one will be given one.) Upon learning that (1) is true of the situation, does it follows that the counterfactual in (2) is also true?

(1) All of the students are prepared for the exam.

(2) If there had been another five students, they would all be prepared.

This depends on further causal facts: either (1) is true because there is some mechanism in place guaranteeing as many pencils as students, in which case (2) is definitely true; or (1) just happens to be true, in which case (2) could well be false. We would like to model both of these cases, and even the inference about which is more likely—and thus how likely (2) is overall—without having to make specific upfront assumptions about how many students there could be.

A second notable feature is that the move from declarative to procedural emphasis has important logical ramifications, already for propositional logic of counterfactuals.

Example 2. If Alf were ever in trouble, the neighbors Bea and Cam would both like to help. But neither wants to help if the other is already helping. Imagine the following scenario: upon finding out that Alf is in trouble, each looks to see if the other is already there to help. If not, then each begins to prepare to help, eventually making their way to Alf but never stopping again to see if the other is doing the same. If instead, e.g., Cam initially sees Bea already going to help, Cam will not go. One might then argue that (3) and (4) are both intuitively true:

(3) If Alf were in trouble, Bea and Cam would both go to help.

(4) If Alf were in trouble and Bea were going to help, Cam would not go to help.

No existing semantic account of counterfactuals—including both world-ordering models and SEMs—can accommodate this pair of judgments, as $A \boxrightarrow (B \wedge C)$ implies $(A \wedge B) \boxrightarrow C$. The only way to make (3) and (4) both true is to insist that the temporal information be made explicit (evidently unlike typical examples modeled with SEMs [5, 12]). In contrast, by suppressing temporal information in a way that mirrors the surface forms of (3) and (4), it will be easy to find an intuitive simulation making $A \boxrightarrow (B \wedge C)$ true, but $(A \wedge B) \boxrightarrow C$ false.

In what follows we first present the definition of intervention for deterministic programs using Turing machines for concrete illustration, and establish some basic facts about expressivity. We then expand the framework to probabilistic programs so as to handle probabilistic counterfactuals. We consider a number of examples, including Examples 1 and 2, throughout. We also discuss logical and other foundational issues along the way.

2 Intervening on Programs

In thinking of a program as defining a simulation model, we are imagining that there are some variables that initially have some values (the "input") and the program proceeds along, changing these values until it halts, at which point the combination of variable values is construed as the "output" of the simulation. Let us assume programs are Turing machines and that we have a dedicated tape and a fixed interpretation of the tape as a representation of the joint state of infinitely many natural-number-valued variables $\{X_n\}_{n\in\mathbb{N}}$.[1] A *state description* is a set of values $\mathbf{x} = \{x_n\}_{n\in\mathbb{N}}$ for all the variables, only finitely many of which may be non-zero; and a *partial state description* will be any set $\{x_i\}_{i\in I}$, for $I \subseteq \mathbb{N}$. A program can thus be conceived as a (partial) transformation of state descriptions. Let us write $\varphi_{\mathsf{T}}^{\mathbf{x}}(X_i)$ for the value X_i takes on when running machine T on input $\mathbf{x}$, provided T halts (it is undefined otherwise). If we want to consider programs with no (equivalently constantly-0) input, we simply write $\varphi_{\mathsf{T}}(X_i)$.

Of course, the interest in programs as simulation models is not just that they transform inputs to outputs, but that there can be rich dynamics in the course of this transformation. Indeed, a program embodies *counterfactual* information about what *would* happen were we to hold fixed the values of some of the variables throughout the computation.

Definition 1 (Intervention)**.** An *intervention* $\mathcal{I}$ is a computable function that takes (the code of) a program T and produces (code for) a new program $\mathcal{I}(\mathsf{T})$ by selecting a partial state description, $\{x_i\}_{i\in I}$ with $I \subseteq \mathbb{N}$ computable, and holding fixed the values of $\{X_i\}_{i\in I}$ to $\{x_i\}_{i\in I}$ in the computation that T performs. Specifically, $\mathcal{I}$ does the following:

1. Add instructions to the beginning to set the finitely many non-0 variables to their values.
2. Before every instruction α add a routine that checks whether the current cell belongs to a variable X_i with $i \in I$. If $i \notin I$, keep α just as before. If $i \in I$, enter a new state for which there is an instruction just like α, except that the value of the cell is not changed.

Intervening on SEMs involves setting a variable to a given value and then asking what solutions to the equations exist. The intuition here is rather different: intervention on a program involves setting a variable to a given value and letting that manipulation have an effect on the dynamics of the program (i.e., the "simulation"). For a very simple illustration let us return to Example 2. In this example we will help ourselves to "pseudo-code" using `if...then` statements and setting variables to values (writing X `:=` `n` for a number `n`, or X `:=` Y for the current value of variable Y), knowing that we can easily transform all of this into Turing machine code.

Example 3. Let us formalize relevant parts of Example 2 with five binary variables:

B: Bea goes to help	D: Bea intends to help	A: Alf is in trouble
C: Cam goes to help	E: Cam intends to help	

Then consider the following simple program:

```
if A = 1 and C = 0 then D := 1
if A = 1 and B = 0 then E := 1
B := D
C := E
```

[1] Where π is a computable pairing function and $\mathcal{V} = \langle V_n\rangle_{n\in\mathbb{N}}$ is the infinite vector of values on the value tape, let us assume X_i is represented in unary by the infinite sublist $V^{(i)} = \langle V_{\pi(i,1)}V_{\pi(i,2)}V_{\pi(i,3)}\dots\rangle$. We furthermore assume that programs are written in a normal form so that the value of each X_i is always encoded as a contiguous sequence of 1's followed by the infinite constantly-0 string.

Suppose that in our default initial state all variables are set to 0. It is easy to check that intervening to set $A = 1$ would result in $B = C = 1$. However, if we intervene to set $A = B = 1$, then the program would halt with $C = 0$.

2.1 The Logic of Counterfactual Simulation

One of the main principles in axiomatizations of SEMs is what Pearl calls *composition* [12, 5] (also known as *Cautious Monotonicity* in the literature on non-monotonic logic):

$$(A \boxrightarrow B \wedge A \boxrightarrow C) \Rightarrow (A \wedge B) \boxrightarrow C$$

The declarative character of SEMs, whereby counterfactuals concern finding solutions of equations, establishes this principle as clearly valid: if any solution setting A to 1 would have both B and C set to 1, then any solution that sets A and B to 1 would have C set to 1.

By contrast, on a straightforward construal of what '$\boxrightarrow$' means for programs—intervene to make the antecedent true and see whether the program halts with the consequent true—the composition axiom, while satisfiable, is not valid, as shown by Example 3. At the risk of belaboring the point, the procedural interpretation invokes a very different intuition from the declarative: even though setting $A = 1$ eventually leads to $B = 1$ and $C = 1$, holding $B = 1$ fixed throughout the computation may disrupt the sequence of steps that leads to $C = 1$.

It is possible to give a complete axiomatization of counterfactuals in this setting [6], showing that the logic fails to include several of the validities shared by logics of SEMs and logics interpreted over systems-of-spheres. In a sense, at least concerning the question of which combinations of counterfactual statements can be given a consistent interpretation, the procedural simulation-based perspective can thus be thought of as more general than the declarative SEM approach. For reasons of space, we leave a fuller treatment of these logical issues, and the interpretive questions they raise, for another occasion [6].

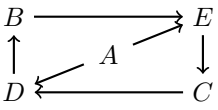
2.2 Defining Causal Graphs

The program in Example 3 clearly reveals an underlying causal structure, which is what supports specific patterns of counterfactuals. Which causal structures can arise from programs? We make this question precise by borrowing a concept from the philosophical literature on causation [18].

Definition 2. Let program T be given. We say that X_i is a *direct cause* of X_j, written $X_i \to X_j$, just in case there are two interventions $\mathcal{I}_1$ and $\mathcal{I}_2$ that hold every variable fixed except for X_j, which differ only in the values assigned to X_i, and for which $\varphi_{\mathcal{I}_1(\mathsf{T})}(X_j) \neq \varphi_{\mathcal{I}_2(\mathsf{T})}(X_j)$.

In other words, $X_i \to X_j$ if X_i directly influences X_j in at least some possible context.

Example 4. Returning to Example 3 it is easy to see that the causal graph defined by this program is as follows:



Note that A does not directly influence B and C, but only via D and E, respectively. Note also that the graph is cyclic, viz. the path $B \to E \to C \to D \to B$.

While in many cases it will be easy to determine the causal graph of a program, the problem in general is unsurprisingly undecidable.

Proposition 1. The problem of determining whether $X_i \to X_j$ is (merely) semi-decidable.

Proof Sketch. That it is semi-decidable is clear: simply dovetail search through all possible pairs of interventions. If there is a pair that results in different values for X_j we will find it.

To see that the problem is not decidable, we reduce it to the problem of determining whether a machine computes a constant function. For any number n consider the Turing machine $\mathsf{T}[n]$ that runs the nth machine T_n on input X_1 and then writes the result of the computation (if it halts) to X_2. We clearly have $X_1 \to X_2$ if and only if T_n does not compute a constant function. As the mapping $n \mapsto \mathsf{T}[n]$ is computable, determining $X_1 \to X_2$ cannot be in general. □

Say that a graph $(\{X_n\}_{n\in\mathbb{N}}, \to)$ is *computably enumerable* (*c.e.*) if the set $\{\langle i, j\rangle : X_i \to X_j\}$ is computably enumerable. It turns out that programs give us all possible c.e. graphs.

Proposition 2. Every c.e. graph is the causal graph for some program.

Proof Sketch. Suppose A is a c.e. set. We describe a program for which $X_i \to X_j$ exactly when $\langle i, j\rangle \in A$. Our program begins by searching to find the first variables with non-zero values (never halting if none is found). Suppose these variables are X_i and X_j with $i < j$.

If $X_i = 1$, then we begin enumerating A until we find the pair $\langle j, i\rangle$ (again, never halting if we never find it). Once found, we write the contents of X_j to X_i and halt. If $X_i > 1$, then we search for the pair $\langle i, j\rangle$ in A. Once found, we write the contents of X_i to X_j and halt.

Clearly, if $\langle n, m\rangle \in A$, then we can find a configuration that witnesses the fact that $X_n \to X_m$. We simply set X_n to either 2 or 3, and X_m to 1 (keeping all others at 0); clearly X_m will depend on X_n no matter whether $n < m$ or $m < n$. If $\langle n, m\rangle \notin A$, then no configuration holding everything but X_n fixed will allow the value of X_m to vary. □

3 Probabilistic Computation and Counterfactuals

To handle causality and counterfactuals in a probabilistic setting we move to stochastic simulations, which we formalize using *probabilistic Turing machines*. In addition to the variable tape encoding $\{X_n\}_{n\in\mathbb{N}}$ we add a *random bit tape* with values $R = \langle R_i\rangle_{i\in\mathbb{N}}$, each bit R_i intuitively representing the result of a fair coin flip.[2] This random source plays a similar role to exogenous variables in SEMs, but in the present context it induces random behavior in our machine: different sequences appearing on the random bit tape may lead to different computations performed by the Turing machine. With T a probabilistic machine, $r \in \{0,1\}^*$ a finite binary sequence, and $\mathbf{Y}$ a sequence of variables from $\{X_n\}_{n\in\mathbb{N}}$, let us write $\varphi^r_{\mathsf{T}}(\mathbf{Y})$ for the sequence $\mathbf{y}$ of values that variables $\mathbf{Y}$ take on provided T has halted after accessing exactly the random bits of r.[3] Then, as each random bit has probability 2^{-1} and any sequence r has probability $2^{-|r|}$, we can express the probability that machine T halts with values $\mathbf{Y} = \mathbf{y}$ as follows:

$$P_{\mathsf{T}}(\mathbf{Y} = \mathbf{y}) \quad = \quad \sum_{r:\varphi^r_{\mathsf{T}}(\mathbf{Y})=\mathbf{y}} 2^{-|r|}$$

Because machines may have positive probability of not halting at all, the sum over all outputs $\mathbf{y}$ may be less than 1. In this sense P_{T} will be a *semi-measure*. Some authors have suggested limiting attention to machines that almost-surely (with probability 1) halt. It is argued in [7] that this is unnecessarily restrictive, in part because of natural examples like the following.

[2] More formally, the distribution on infinite binary strings is given by the Borel probability space $(\{0,1\}^\omega, \mathbb{P})$, where $\mathbb{P}$ is the infinite product of Bernoulli(1/2) measures. See, e.g., [3].

[3] Thus, $\varphi^r_{\mathsf{T}}(\mathbf{Y})$ is undefined if T either reads only an initial segment of r or moves beyond r on the random bit tape. Note that given a particular random bit sequence R, the operation of the machine is fully deterministic.

Example 5. Imagine a race between a tortoise and a hare. We have variables $T_0, T_1, T_2, \ldots$ for the position of the tortoise at each time step, and variables $H_0, H_1, H_2, \ldots$ similarly for the hare. Where `Flip(1/4)` is a procedure that returns 1 with probability 1/4 and `Unif(1,7)` returns a number between 1 and 7 uniformly, we might imagine a simulation like this:

T_0 `:= 1;` H_0 `:= 0`
`while (`H_t `<` T_t`)`
    T_{t+1} `:=` T_t `+ 1;` H_{t+1} `:=` H_t
    `if Flip(1/4) then` H_{t+1} `:=` H_t `+ Unif(1,7)`

Whereas this program would almost-surely halt, any small change to the program (e.g., incrementing the tortoise's pace by ϵ) would lead to positive probability of the hare never catching up, even though the two programs may be practically indistinguishable [7].

From a theoretical point of view, we can characterize exactly which semi-measures $P(\mathbf{Y})$ can be defined by a probabilistic Turing machine. We say $P(\mathbf{Y})$ is *enumerable* if for each $\mathbf{y}$ the probability $P(\mathbf{Y} = \mathbf{y})$ can be computably approximated by an increasing sequence of rationals.

Proposition 3 ([7])**.** For every probabilistic Turing machine T, $P_{\mathsf{T}}(\mathbf{Y})$ is an enumerable semi-measure; moreover, every enumerable semi-measure is $P_{\mathsf{T}}(\mathbf{Y})$ for some T.

To capture the causal structure of a probabilistic program we can use the very same definition of intervention (Def. 1). In Example 5, for instance, while $P_{\mathsf{T}}(H_2 \geq T_2) \approx .36$, under an intervention $\mathcal{I}$ that sets H_1 to 1 we would have $P_{\mathcal{I}(\mathsf{T})}(H_2 \geq T_2) \approx .21$. We can also carry over our definition of direct cause (Def. 2) with only slight modification.

Definition 3. Given probabilistic program T, we say $X_i \to X_j$ just in case there are two interventions $\mathcal{I}_1$ and $\mathcal{I}_2$ that hold every variable fixed except for X_j, which differ only in the values assigned to X_i, and for which $P_{\mathcal{I}_1(\mathsf{T})}(X_j) \neq P_{\mathcal{I}_2(\mathsf{T})}(X_j)$.

That is, holding everything but X_j fixed, changing X_i effects a change in probability of X_j.

Example 6. The causal structure of the program in Example 5 consists of two infinite chains:

$$T_0 \longrightarrow T_1 \longrightarrow T_2 \longrightarrow T_3 \longrightarrow \cdots$$

$$H_0 \longrightarrow H_1 \longrightarrow H_2 \longrightarrow H_3 \longrightarrow \cdots$$

As one would expect, the computability and universality results, Props. 1 and 2, apply without change for these probabilistic analogues: we still obtain exactly the c.e. causal graphs.

3.1 Conditioning

Central to the probabilistic setting is the operation of *conditioning* a distribution, which in this context amounts to restricting attention to those runs of the simulation model that eventuate in a particular outcome. Specifically, we can define a (universal) machine COND that takes (codes of) two machines T and F as arguments and (provided F almost-surely halts and returns 1 with positive probability) defines a new simulation model $\mathsf{COND}(\mathsf{T}, \mathsf{F})$ that correctly represents the conditioned semi-measure. For example, if F is a program that checks whether variables $\mathbf{Z}$ would have values $\mathbf{z}$, then $P_{\mathsf{COND}(\mathsf{T},\mathsf{F})}(\mathbf{Y}) = P_{\mathsf{T}}(\mathbf{Y} \mid \mathbf{Z} = \mathbf{z})$, where the latter is defined by the usual ratio formula. This shows that the enumerable semi-measures, or equivalently (by Prop.

3) the machine-definable distributions, are closed under computable conditioning. (See [3] for details on COND and [7] for the general setting of enumerable semi-measures.)

As with other graphical models, conditioning on a "causally upstream" variable is the same as intervening on that variable. For instance, we have $P_{\mathsf{T}}(H_2 \geq T_2 \mid H_1 = 1) = P_{\mathcal{I}(\mathsf{T})}(H_2 \geq T_2)$ in Example 5. The interest comes in combining observations with interventions. Indeed, for Pearl the essence of a counterfactual $A \boxdot\!\!\rightarrow B$ is captured by a three-step procedure [12, 206]:

1. **Abduction**: Update the model with any relevant observations.
2. **Action**: Modify the model by intervening to make A true.
3. **Prediction**: Use the modified model to compute the probability of B.

Enabling this combination of operations is in fact a major consideration favoring SEMs over Bayes nets, according to Pearl [12, §1.4]. If we like, we can perform the same combination of operations over probabilistic programs.

Example 7. Continuing with Example 5, suppose we observed a run like this:

$$T_0 = 1 \quad H_0 = 0 \qquad T_1 = 2 \quad H_1 = 1 \qquad T_2 = 3 \quad H_2 = 1 \qquad T_3 = 4 \quad H_3 = 4$$

Given the actual trajectory, the hare caught up by time 3 and the simulation terminated. But we could ask, given what happened, if (counter to the facts) the hare had not jumped forward at time 1, would the hare still have caught up by time 3? Where F is a program that verifies the observations above, we first condition T on F to obtain a new program $\mathsf{COND}(\mathsf{T}, \mathsf{F})$. This effectively fixes the first six random choices to ensure that the program (without any interventions) would produce these very observations. However, when we then intervene to set H_1 to 0, running the manipulated program forward results in $H_2 = 0$ and $H_3 = 3$, which means the hare has not caught up and the program would not have halted by time 3.

Given the same observations, and under the same counterfactual supposition, we can also ask what would be the probability of the hare catching up by time 4. If $\mathcal{I}$ is the intervention setting $H_1 = 0$, this is given by $P_{\mathcal{I}(\mathsf{COND}(\mathsf{T},\mathsf{F}))}(H_4 \geq T_4)$, which happens to be 3/14.

3.2 A Note on D-separation and Conditional Independence

Much of the interest in graphical structures in the literature on probability stems from the possibility of reading off (conditional) independence facts from simple graphical properties. For Bayes nets and SEMs, the critical concept is that of *d-separation*. Roughly speaking, variables $\mathbf{Z}$ d-separate $\mathbf{X}$ from $\mathbf{Y}$ if every possible path of information flow from $\mathbf{Y}$ to $\mathbf{X}$ is blocked by some variable in $\mathbf{Z}$.[4] This guarantees that, conditional on $\mathbf{Z}$, $\mathbf{X}$ is independent of $\mathbf{Y}$; that is, $P(\mathbf{X} \mid \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Y}, \mathbf{Z})$ (see, e.g., [12]).

How does this look for causal graphs defined by programs? Fixing program T, let us say that X_i depends on the nth random bit, written $R_n \to X_i$, just in case there is an intervention $\mathcal{I}$ and sequences r_1 and r_2 that differ only at the nth place, such that $\varphi^{r_1}_{\mathcal{I}(\mathsf{T})}(X_i) \neq \varphi^{r_2}_{\mathcal{I}(\mathsf{T})}(X_i)$. Evidently, if $R_n \to X_i$ and $R_n \to X_j$ this may induce a dependence between them even when X_i and X_j are d-separated in the context of graph $(\{X_n\}_{n\in\mathbb{N}}, \to)$.

If we want d-separation to guarantee (conditional) independence, we have two obvious choices. One is to include $\{R_n\}_{n\in\mathbb{N}}$ as variables in the graph alongside $\{X_n\}_{n\in\mathbb{N}}$ and expand

[4] Specifically, for every path from $\mathbf{Y}$ to $\mathbf{X}$ there must be three variables U, V, W along the path such that either (1) $U \to V \to W$ or $U \leftarrow V \to W$, and $V \in \mathbf{Z}$, or (2) $U \to V \leftarrow W$ and no descendent of V is in $\mathbf{Z}$.

the edge relation $\rightarrow$ accordingly. The other is to insist that we only write programs in such a way that no random bit is a direct cause of two different variables. A similar stipulation is often made in the context of SEMs [12, §1.4]. It is clear that Prop. 2 would not be affected by such a requirement (since that did not require use of the random source at all), but it is perhaps an interesting question whether the universality result in Prop. 3 would still hold. At any rate, either of these stipulations allows for essentially the same argument as for Bayes nets or for SEMs to show conditional independence.

4 Open-World Reasoning

A hallmark of ordinary reasoning in natural language is our ability to deal with situations at a level of abstraction that does not depend on knowing which, or how many, individuals pertain to a given situation. There is no claim that this kind of reasoning is impossible to formalize in other frameworks; the point to emphasize is rather that this kind of reasoning is very natural for simulations built using familiar programming tools such as recursion [13, 11, 4]. In this section we return to consider how one might model the situation described in Example 1.

Suppose we have the following variables, with their intended meanings:

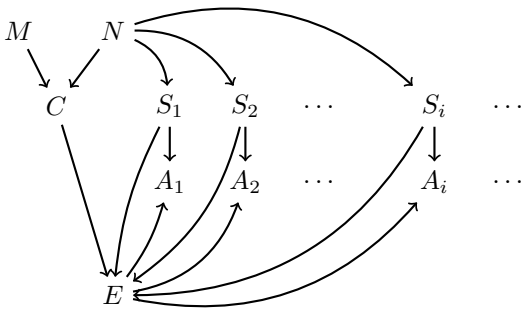
N: number of students	$S_1, S_2, \ldots S_i \ldots$: student i brought their own pencil
M: a mechanism is in place to guarantee the same number of pencils as students	$A_1, A_2, \ldots A_i \ldots$: student i is prepared
C: number of extra pencils	E: There are enough pencils

Let us assume M and E take on values 0 (false) and 1 (true), while variables S_i and A_i take on three values: 0 ("undefined"), 1 (false), and 2 (true). Intuitively A_i should be defined exactly when S_i is, and that should happen only when there actually is an ith student.

In the following program T we assume that four routines for generating numbers randomly are given: $\mathcal{D}_M$, $\mathcal{D}_N$, $\mathcal{D}_C$, and $\mathcal{D}_S$. These can be thought of as defining the "prior" generating procedures for the relevant variables; the precise details will not matter for this example.

$$
\begin{array}{l}
M \texttt{ := } \mathcal{D}_M\texttt{; } N \texttt{ := } \mathcal{D}_N \\
C \texttt{ := if } M \texttt{ then } N \texttt{ else } \mathcal{D}_C \\
\texttt{for } i \texttt{ from 1 to } N\texttt{: } \quad S_i \texttt{ := } \mathcal{D}_S \\
E \texttt{ := } C \geq |\{i : S_i \texttt{ = } 1\}| \\
\texttt{for } i \texttt{ from 1 to } N\texttt{: } \quad A_i \texttt{ := max(}S_i\texttt{,}E\texttt{+1)}
\end{array}
$$

The causal graph for T would then look like this:



What does it mean to say that all of the students are prepared (as in (1) from Example 1)? It simply means that none of the variables A_i have value 1, a statement that can be easily checked. Suppose that we know $M = 1$—that there are definitely enough pencils for everyone—and we learn that everyone is prepared. (In fact, the latter is already guaranteed by $M = 1$.) Conditioning our model with a program F_1 that represents these two observations, it is easily seen that the counterfactual in (2), "If there had been another five students, they would all be prepared," has probability 1 according to the program $\mathsf{COND}(\mathsf{T}, \mathsf{F}_1)$.

Suppose on the other hand that we knew $M = 0$. Suppose also that $\mathcal{D}_C$ typically produces small numbers relative to $\mathcal{D}_N$—meaning that there are normally many more students than extra pencils—and that $\mathcal{D}_S$ is such that students almost never bring their own pencils. In such a case learning (1) would be quite surprising. We would moreover expect that there just happened to be enough pencils, but had there been any more students there would not have been enough. Thus, given the conditioned program $\mathsf{COND}(\mathsf{T}, \mathsf{F}_2)$ and where $\mathcal{I}$ is the intervention setting N to $N + 5$, the statement, "All of the students are prepared," has low probability according to $\mathcal{I}\big(\mathsf{COND}(\mathsf{T}, \mathsf{F}_2)\big)$. That is, according to $\mathsf{COND}(\mathsf{T}, \mathsf{F}_2)$, the counterfactual (2) has low probability.

What if we did not know anything about M at all, but merely learned (1). Let F_3 represent observation of (1). As (1) is fully expected when M is true, but quite surprising when M is false, ordinary Bayesian reasoning shows that $\mathsf{COND}(\mathsf{T}, \mathsf{F}_3)$ will now assign M higher probability: in effect, M will now be drawn from a distribution $\mathcal{D}'_M$ that puts more weight on 1 than $\mathcal{D}_M$. The probability of (2) will be intermediate between 1 (the prediction if $M = 1$) and the prediction when $M = 0$, with the precise weighting depending on $\mathcal{D}'_M$, as it intuitively should be.

Two features of this example should be highlighted. The first is, once again, reasoning about this situation does not require making any fixed assumption about which individuals are present. The second is that, though we depict all dependence relations with the same arrow $\rightarrow$, the program T embodies some rather different relationships among variables. For instance, M does not "cause" C in any ordinary sense, but rather modulates whether C depends on N. Similarly, N determines S_i just in the sense that it determines whether there is a student i at all; intuitively, N modulates whether the variable S_i is a relevant part of the current simulation. (Note that we could have made N depend on all of the S_i's.) Philosophers have recognized deep similarities between causal and other kinds of dependence [15]. Because $\rightarrow$ is defined simply by reference to the causal structure of the program, we have blurred all such distinctions.

5 Conclusion

The objective of this paper has been to clarify some of what it might mean to use programs to define causal models, and to ground causal and counterfactual language. In many cases—evidently including most examples considered in recent work in semantics that invoke causal models—the difference between the present framework and more familiar frameworks, such as suitably general classes of SEMs, will not matter. Nonetheless, we have highlighted some important differences, some of which surface already at the level of basic logical validities.

There is certainly no claim that the present framework captures *causality* better than other frameworks. For understanding causal explanation in science, for example, SEMs may often be more useful (see, e.g., [18]). At the same time, for understanding ordinary causal judgments—which have their own distinctive character, cf. [2]—one might argue that the role of simulation is fundamental [4, 3, 1]. Insofar as this is true, we would expect it to be reflected in how people speak about causation as well. At least two points are worth mentioning on this theme.

First, as just mentioned, the framework blurs the distinction between causal and non-causal counterfactuals. While the empirical literature clearly shows that people discriminate causation

from statistical association, it is less obvious that there is any fundamental *cognitive* distinction between causal and, say, logical dependence (or, relatedly, explanation). Within the framework explored here, causal counterfactuals ("If the vase had dropped, it would have broken") can be treated in the very same way as non-causal counterfactuals ("If the vase had been turquoise, then it would have been blue"), which is especially convenient when we need to treat counterfactuals that combine causal and other kinds of dependence (such as sentence (2) from Example 1).

Second, the general framework fits in nicely with a view according to which people select and evaluate counterfactuals stochastically, over richly structured representations, in such a way that the relevant simulation probabilities reflect psychological biases (availability, anchoring, etc.) that can have little to do with "objective" statistics of a situation. This allows incorporating well known psychological effects right into the analysis of conditionals and causal language. As an example, it is often observed that moral considerations affect the way people construct counterfactual scenarios, and, presumably associated with this, their judgments of actual cause ("what caused what"). For instance, the very same act can be judged as more or less causal depending on how people judge its moral status. Recently proposed explanations of these and related phenomena fit very harmoniously with the framework explored here [8].

References

[1] Nick Chater and Mike Oaksford. Programs as causal models: Speculations on mental programs and mental representation. *Cognitive Science*, 37(6):1171–1191, 2013.

[2] David Danks. *Unifying the Mind: Cognitive Representations as Graphical Models*. MIT, 2014.

[3] Cameron E. Freer, Daniel M. Roy, and Joshua B. Tenenbaum. Towards common-sense reasoning via conditional simulation: Legacies of Turing in artificial intelligence. In R. Downey, editor, *Turing's Legacy*. ASL Lecture Notes in Logic, 2012.

[4] Noah D. Goodman, Joshua B. Tenenbaum, and Tobias Gerstenberg. Concepts in a probabilistic language of thought. In Eric Margolis and Stephan Laurence, editors, *The Conceptual Mind: New Directions in the Study of Concepts*. MIT Press, 2015.

[5] Joseph Y. Halpern. Axiomatizing causal reasoning. *Journal of AI Research*, 12:317–337, 2000.

[6] Duligur Ibeling and Thomas Icard. On the conditional logic of simulation models. In *Proc. 27th IJCAI*, 2018.

[7] Thomas Icard. Beyond almost-sure termination. In *Proc. 39th CogSci*, 2017.

[8] Thomas Icard, Jonathan Kominsky, and Joshua Knobe. Normality and actual causal strength. *Cognition*, 161:80–93, 2017.

[9] Stefan Kaufmann. Causal premise semantics. *Cognitive Science*, 37(6):1136–1170, 2013.

[10] David Lewis. *Counterfactuals*. Harvard University Press, 1973.

[11] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In *Proc. 19th IJCAI*, pages 1352–1359, 2005.

[12] Judea Pearl. *Causality*. Cambridge University Press, 2009.

[13] Avi Pfeffer and Daphne Koller. Semantics and inference for recursive probability models. In *Proc. 7th National Conference on Artificial Intelligence (AAAI-00)*, pages 538–544, 2000.

[14] Paolo Santorio. Interventions in premise semantics. *Philosophers' Imprint*, 2018.

[15] Jonathan Schaffer. Grounding in the image of causation. *Phil. Studies*, 173(1):49–100, 2016.

[16] Katrin Schulz. "If you'd wiggled A, then B would've changed": Causality and counterfactual conditionals. *Synthese*, 179(2):239–251, 2011.

[17] Steven Sloman, Aron K. Barbey, and Jared M. Hotaling. A causal model theory of the meaning of *cause*, *enable*, and *prevent*. *Cognitive Science*, 33(1):21–50, 2009.

[18] James Woodward. *Making Things Happen: A Theory of Causal Explanation*. OUP, 2003.