Causality and Semantic Separation

ANNA ZHANG, Massachusetts Institute of Technology, USA
QINGLAN LUO, Wellesley College, USA and Massachusetts Institute of Technology, USA
LONDON BIELICKE, University of California at Los Angeles, USA
EUNICE JUN, University of California at Los Angeles, USA
ADAM CHLIPALA, Massachusetts Institute of Technology, USA

The design of scientific experiments deserves its own variation of formal verification to catch cases where scientists made important mistakes, such as forgetting to take confounding variables into account. One of the most fundamental underpinnings of science is *causality*, or what it means for interventions in the world to *cause* other outcomes, as formalized by computer scientists like Judea Pearl. However, these ideas had not previously been made rigorous to the standards of the programming-languages community, where one expects a (syntactic) program analysis to be proved sound with respect to a natural semantics. In the domain of causality, as the relevant "program analysis," we focus on d-*separation*, a classic condition on graphs that can be used to decide when the design of an experiment controls for sufficiently many confounding variables, even though the reason that this condition works is often unintuitive. Our central result (mechanized in Rocq) is that *d*-separation exactly coincides with a novel *semantic* definition inspired by noninterference from the theory of security. This characterization provides a structural semantic foundation for *d*-separation and helps explain why the graph-theoretic condition is correct, independently of probabilistic assumptions. For each given automated test on the quality of an experiment design, our theorem justifies an associated method for falsifying the world-modeling hypothesis behind the experiment.

CCS Concepts: • **Software and its engineering → Formal software verification**; **Semantics**.

Additional Key Words and Phrases: causal models, *d*-separation, logical foundations of science

ACM Reference Format:
Anna Zhang, Qinglan Luo, London Bielicke, Eunice Jun, and Adam Chlipala. 2026. Causality and Semantic Separation. *Proc. ACM Program. Lang.* 10, PLDI, Article 196 (June 2026), 24 pages. https://doi.org/10.1145/3808274

1 Introduction

A scientific experiment can be seen as a program, perhaps a complex one, that performs particular interactions with its environment, with the goal of refining an understanding of how that environment works. That is, the scientists begin with a *world model* that identifies a set (often infinite) of possible programs that could represent the environment. Through the experiment, a scientist manages to shrink the set of world models that remain plausible. The hazards in experimental design are so common and diverse that the process of enumerating them is known as assessing *threats to validity*. How is a scientist to know an experimental design accounts for all of the important hazards? We found that question similar to the question of how a programmer knows that

Authors' Contact Information: Anna Zhang, Massachusetts Institute of Technology, Cambridge, USA, annazhang@alum.mit.edu; Qinglan Luo, Wellesley College, Wellesley, USA and Massachusetts Institute of Technology, Cambridge, USA, ql101@mit.edu; London Bielicke, University of California at Los Angeles, Los Angeles, USA, londonbielicke@g.ucla.edu; Eunice Jun, University of California at Los Angeles, Los Angeles, USA, emjun@ucla.edu; Adam Chlipala, Massachusetts Institute of Technology, Cambridge, USA, adamc@csail.mit.edu.

ACM 2475-1421/2026/6-ART196
https://doi.org/10.1145/3808274

a program behaves properly, which leads to the whole discipline of program verification. Noting the apparent lack of such a discipline for experiment design, we set out to start building up its foundations, which led us to the specific element of *causality* that this paper covers.

First, let us review the elements of program verification often taken for granted, to help in drawing connections to experiment design. Nearly any program verification involves *assumptions* and *proof obligations*. The assumptions could include the semantics of the language that the program is written in or the behavior of external libraries to which the program links. Then we have a *specification* covering how the program is allowed to behave, when the assumptions hold.

Similar elements are apparent in experiment design. An experiment runs in some real-world setting, where it is reasonable to model the world as a program. We do not know exactly which program, but we aim to learn more about the possibilities. If we simply fix the "type signature" of the world in terms of which measurable quantities exist, then we allow too broad a space of programs, where it is hopeless to use observation to narrow the possibilities significantly. Thus, just as a program verification may assume a specification for library functions, an experiment assumes a *world model* that fixes a class of programs that ought to include the *true* world program. Then the experiment, the program to be verified, interacts with the world and uses what it learns to rule out possible world models. So *the soundness of logic for ruling out possible worlds is central to science.*

To make this idea more tangible, consider a simple example from medicine. Suppose we are testing a new treatment for a heart condition, and a scientist's model assumes that gender has no direct causal effect on recovery rates once treatment and age are controlled for. This assumption corresponds to a particular structure in the model's causal graph. If, in a randomized clinical trial, we observe that outcomes still vary systematically by gender after conditioning on treatment and age, then the experiment has falsified that model. In this sense, the experiment serves as a test oracle for the world model: it probes the model's assumptions through carefully designed manipulations.

Our framework aims to formalize the connection between the structure of a causal model and the outcomes of well-controlled experiments in the same way that programming-language semantics formalize what can be proved about a program's behavior. Program verification often relies on tractable automated analyses of programs, which should be proved sound with respect to semantics. For instance, an abstract interpretation [Cousot and Cousot 1977] is used to assign overapproximating syntactic descriptions to the variables of a program, iterating to find a fixed point in a lattice. We define soundness with respect to the assumed semantics of the programming language. In experiment design, there are similar techniques that generations of scientists have relied upon. The field of *causal modeling* defines the analogue of programs in a language: a *structural causal model* or *causal model*, a DAG whose nodes are fundamental quantities in the world (perhaps observable, perhaps not), where directed edges indicate (potential) causal relationships between nodes. There is also the equivalent of a set of program analyses. In this paper, we focus on d-*separation*, a decidable property that characterizes when a set of nodes in a DAG blocks all influences between two others – for example, when conditioning on that set avoids spurious conclusions due to confounding. Although the definition of d-separation is simple, why it works is not obvious. We therefore set out to prove a relationship with a semantic characterization that explains the correctness of the graph-theoretic condition.

We were inspired by two families of software-related techniques.

(1) *Information-flow security* drives reasoning about how programs protect secrets and block undue influence by their environments through controlling how information flows between different inputs, outputs, and state elements. The gold-standard semantic property is *noninterference* [Goguen and Meseguer 1982], for instance characterizing confidentiality by saying that varying secret inputs to a program must never change public outputs. Similar kinds of

information-flow restrictions are central to the value of causal DAGs in restricting the sets of programs needed to model real-world phenomena, but the field of causality had lacked a similarly elegant *semantic* explanation of what d-separation checks guarantee.

(2) *Program testing* allows us to find bugs by subjecting programs to clever sequences of inputs, designed to exercise corner cases. Regardless of the clever test suite, we need a *test oracle* that evaluates program behavior against a specification. As an application of our new semantic characterization of d-separation, we demonstrate how to prove soundness of test oracles, in the sense that, through observation of the world, they are able to *falsify* world models. In a (somewhat subtle) sense, we also prove that certain oracles are *complete*, meaning if we happened to know that no possible test could fail the oracle, then the d-separation syntactic check must succeed on the DAG.

So, summing up, we introduce a new semantic condition characterizing what d-separation establishes, and we start to explore its consequences for justifying the sound design of experiments. Our central result is fully mechanized in the Rocq Prover, and we provide the code as an associated artifact [Zhang et al. 2026b]. A full version of the paper, including appendices, is available on arXiv [Zhang et al. 2026a].

We now review the relevant concepts from the literature on experimental design and causal reasoning. Then, we present our new semantic characterization of d-separation and our main theorem justifying it (soundness and completeness). Afterward, we examine implications of that theorem: a more intuitive way of understanding what d-separation tells us, which allows us to prove soundness and completeness for test oracles to use in experiments. With the payoff established, we conclude by sketching the proof of our central theorem.

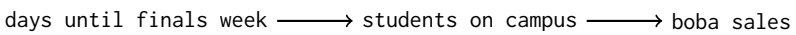
2 Background

2.1 Experimental Design and Validity

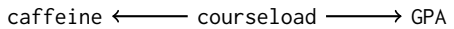
The purpose of scientific experiments is to establish causal relationships. For example, randomized controlled trials are the gold standard for establishing causality in medicine. Experiments are well-designed if they provide unbiased, valid causal estimates. In practice, ideal designs are expensive, requiring many participants and resources. Certain design choices are more efficient but may inadvertently introduce confounding. Confounding, or the presence of variables that influence both causes and effects, can lead to an inability to calculate causal estimands [Cinelli et al. 2020].

In *Experimental and Quasi-experimental Designs for Generalized Causal Inference*, Shadish et al. [2002] put forth a now-prevalent theory of scientific validity for experiments. They outline four types of validity: external, construct, internal, and statistical-conclusion. External validity refers to the generalizability of experimental findings. Construct validity pertains to how well the variables measured (e.g., IQ test results vs. SAT scores) represent the intended construct (e.g., intelligence). Internal validity in an experiment refers to the accurate estimation of causal conclusions. Statistical-conclusion validity relates to the appropriate and correct formulation of statistical models to analyze data collected from an experiment. While Shadish et al. provide these definitions and enumerate threats to validity, or ways in which each type of validity can be compromised, no work has since provided a formal, principled way to detect scientific validity automatically in experiments. The closest set of tools for diagnosing validity come in the form of graphical criteria (e.g., backdoor criterion) using Pearl's causal diagrams [Pearl 2009], as discussed below. Our work is a step towards connecting experimental design with causal reasoning by establishing a formal framework for analyzing the semantics of causal diagrams.

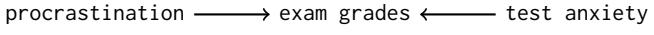
Synthesizing the data collected through a scientific experiment to establish evidence for causal relationships involves the following steps. First, researchers must formulate a causal model that



(a) The number of students studying on campus **mediates** the effect of finals season on boba sales.



(b) From analysis of experimental data alone, it may appear that higher caffeine consumption is a cause of a lower GPA. However, the causal model shows that the **confounding** variable, a heavy courseload, is actually the cause of both.



(c) While lots of procrastination and high test anxiety are otherwise unrelated, conditioning on the **collider** (e.g., looking only at students with low exam grades) can create a false association between the two.

Fig. 1. These causal models illustrate examples of mediators, confounders, and colliders in academic settings.

encodes their hypotheses about the structure of causal relationships as a DAG. This model must be falsifiable. It should make predictions that can be tested and potentially contradicted by observed data. Second, researchers assign probabilities to values of nodes (variables) and use statistical methods (e.g., linear regression, Bayesian inference) to update these probabilities in light of experimental data. Our work is primarily focused on the first step. We formalize how experimental designs support or challenge the falsifiability of a given causal model.

2.2 Frameworks of Causality

Structural causal models (SCMs) [Pearl 2009] represent causal relationships using DAGs, where nodes correspond to variables, and edges represent direct causal effects. SCMs encode assumptions about causal structure explicitly and support reasoning about interventions using the do-operator, which simulates external manipulation of variables (i.e., random assignment in an experiment).

The Potential Outcomes framework (PO) [Imbens and Rubin 2015; Rubin 1974] characterizes causal effects by comparing counterfactual outcomes for the same unit (e.g., patient) under different treatments (e.g., drug vs. placebo). While PO provides an instance-based view, SCMs provide a graphical, compositional view of causal networks from which to derive counterfactuals. We focus on SCMs due to their structural similarity to information-flow security models (see Section 7).

2.3 Fundamental Concepts in SCMs

Understanding causality using SCMs begins with acknowledging that different types of questions about the world require fundamentally different kinds of reasoning. The hierarchy known as the "Ladder of Causation" [Pearl and Mackenzie 2018] is helpful for categorizing these levels of causal understanding: association, intervention, and counterfactuals. While correlations in observational data alone can answer questions at the first level, answering intervention or counterfactual questions typically requires assumptions about the underlying causal structure of the world. SCMs, represented as DAGs, support such reasoning.

It is useful to consider causal *paths* because they represent possible routes through which influence can flow among variables in a causal model. These paths are undirected in the sense that they may contain edges pointing both forward and backward, allowing us to examine dependencies that emerge from various structural configurations. (This need to account for edges flowing in both directions can perhaps be seen as the reason why we need to develop new theoretical machinery, beyond what arises for noninterference in information-flow security.) The following three node structures often appear in such paths:

Definition 2.1. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a causal model, where $a, b, c \in \mathcal{V}$.

- b is a **mediator** of a and c if $(a, b) \in \mathcal{E}$ and $(b, c) \in \mathcal{E}$ or if $(b, a) \in \mathcal{E}$ and $(c, b) \in \mathcal{E}$.
- b is a **confounder** of a and c if $(b, a) \in \mathcal{E}$ and $(b, c) \in \mathcal{E}$.
- b is a **collider** of a and c if $(a, b) \in \mathcal{E}$ and $(c, b) \in \mathcal{E}$.

Definition 2.2. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a causal model and let P be a path in $\mathcal{G}$. If a, b, and c are three consecutive nodes on P, then b is a **mediator, confounder, or collider on P** according to which relationship from Definition 2.1 holds among a, b, and c.

Examples appear in Figure 1. Understanding how to *adjust*, or statistically account, for these structures is central to causal inference. For instance, adjusting for a confounder helps remove estimation bias by "blocking," or mitigating, spurious associations, but adjusting for a collider can actually introduce bias by opening a non-causal path between variables.

More formally, causal relationships are naturally related to the concept of independence. In particular, a relevant property in a causal model is whether two nodes a and b are independent conditioned on some subset of nodes $Z \subseteq \mathcal{V}$. If so, then an experiment conditioning on (e.g., through randomization or sampling) the variables in Z will ensure that a has no effect on b, and vice versa. This concept can be formalized further using probabilities.

Definition 2.3. Let $\mathcal{V}$ be a finite set of variables, and let $\mathbf{P}(\cdot)$ be a probability distribution over $\mathcal{V}$. Let $a, b \in \mathcal{V}$ and $Z \subseteq \mathcal{V}$. Then, variables a and b are **conditionally independent given Z** if

$$\mathbf{P}(a = \alpha \mid b = \beta, Z = (\zeta_1, ..., \zeta_k)) = \mathbf{P}(a = \alpha \mid Z = (\zeta_1, ..., \zeta_k))$$

for all possible assignments of values $\alpha, \beta, (\zeta_1, ..., \zeta_k)$.

While this definition is precise, there is no clear mapping to reasoning about causal graphs. The probabilistic notion of conditional independence is inherently algebraic: it tells us that knowing Z renders a and b independent, but it offers no guidance on how to determine this relationship from the structure of a graph. In practice, repeatedly checking conditional independencies from joint distributions is computationally expensive and provides little insight into *why* the independence holds. What we need is a way (a "program analysis" when we see causal DAGs as our programs) to read off these relationships directly from the graph itself through a structural criterion that corresponds to conditional independence in the underlying distribution. The notion of d-separation provides this graphical criterion.

Definition 2.4. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a causal model. Then, nodes a and b are ***d*-connected given Z**, where $Z \subseteq \mathcal{V}$, if and only if there exists an undirected path P from a to b in $\mathcal{G}$ such that the following two conditions hold:

(1) No mediator or confounder on P is in Z.
(2) Every collider on P has a descendant in Z (we consider a node as its own descendant).

We say a and b are ***d*-separated given Z** if and only if they are not d-connected given Z.

We can think of the variables in Z as "blocking" any effect that a may have on b. For example, in Figure 1b, `caffeine` and `GPA` are d-separated given $Z = \{\texttt{courseload}\}$ but not $Z = \emptyset$.

In *Causality*, Pearl [2009] connects the notions of d-separation and conditional independence, stating that for a causal model $\mathcal{G}$, if a and b are d-separated given Z, then conditional independence given Z holds between a and b for every probability distribution $\mathbf{P}(\cdot)$ compatible with $\mathcal{G}$. Conversely, if a and b are not d-separated given Z, then there is at least one distribution $\mathbf{P}(\cdot)$ compatible with $\mathcal{G}$ in which a and b are not conditionally independent given Z.

While d-separation offers a graph-theoretic criterion for deciding conditional independence, and the probabilistic definition formalizes it algebraically, both leave a semantic gap: they do not explicitly connect the structure of a DAG to the mechanisms that generate the observed distributions. On one hand, d-separation provides a syntactic rule for detecting conditional independence without grounding it in the specifics of a data-generating process. On the other hand, the probabilistic definition treats conditional independence as a property of distributions without reference to underlying causal mechanisms, making it agnostic to the data-generating process. Neither approach fully captures the intuition that conditional independence should arise from how variables are causally related through structural assignments and shared randomness. We will introduce a semantic definition to fill this gap by grounding conditional independence in the functional dependencies encoded by the DAG.

More specifically, we have found the definition of d-separation unintuitive in terms of the conditions it levies on paths. Why are these conditions *exactly* the right ones to rule out bad confounding? Pearl's proved equivalence [Pearl 2009] seems to offer just that kind of answer, but much is hiding in the phrase "compatible with $\mathcal{G}$." This statement restricts the probability distribution $\mathbf{P}$, such that for every DAG node v with parents $\mathrm{Pa}(v)$ (those nodes with edges heading into v), we have $\mathbf{P}(v) = \Pi_i \mathbf{P}(v \mid \mathrm{Pa}(v)_i)$. Setting aside the difficulty of confirming such probability factorizations in practice, we note that conditional independence is a property of a particular distribution rather than of the graph structure itself. In particular, conditional independence can arise accidentally (e.g., through cancellation effects) even when a causal path exists. We are left wondering why the concept of probability is needed to explain causality, which can also be formalized for purely deterministic systems.

To that end, we give a deterministic semantic characterization of what d-separation establishes, isolating its structural meaning before layering on probabilistic assumptions. This characterization is not intended as a new algorithmic test; its purpose is to provide a faithful semantic foundation for d-separation, thereby justifying the use of d-separation as the computationally efficient static analysis of causal diagrams.

3 A New Semantic Characterization of Separation Under Conditions

To capture the ideas of conditional independence in a more intuitive way, we introduce a new concept called *semantic separation*: a characterization of d-separation that specifies how values propagate through a graph, rather than focusing on purely syntactic criteria. The semantic interpretation helps us understand not only the relationships among the nodes but also what these relationships imply about the behavior of variables under interventions or counterfactual scenarios.

3.1 Function-Based Formal Semantics

Appendix A describes in detail how we represent causal models in Rocq, including the development of key graph-theoretic and causal-inference foundations. In this section, we present the underlying semantic model in mathematical terms. The Rocq formalizations of the semantics are described in Appendix B.

Causal models tell us the relationships that may exist between nodes. Assuming that a causal model accurately captures the causal relationships between nodes, each node's value is determined by a function of the values of its parents. In practice, however, the exact functional form of these relationships is unknown; to account for this uncertainty, we include an independent *unobserved error term* for each node. These unobserved terms, common in causal inference, represent latent factors not explicitly included in the DAG. They are often visualized as greyed-out parent nodes. This abstraction reflects the reality that causal graphs rarely model all relevant background factors, so any omitted influences on a node are absorbed into its unobserved term.

We choose to capture these relationships using node functions. A *node function* assigns a value to a node based on its unobserved term and the values of its parents. Let $\mathcal{X}$ denote the domain of possible node values. Then each node function takes an unobserved term in $\mathcal{X}$ and a list of parent values in $\mathcal{X}$ and returns a value in $\mathcal{X}$. Note that the node function is of course free to ignore or allow little influence by the unobserved term.

We represent a causal model with an overarching *graph function*, which maps each node in the graph to its corresponding node function. Together with an assignment of unobserved terms, it determines the value of every node.

To illustrate how a graph function captures meaningful causal relationships, consider a simple example modeling how a student's sleep, study hours, and focus influence their test score. Suppose our causal graph is as shown in Figure 2.

Then, we might have the following graph function g, where node values lie in $\mathcal{X} = \mathbb{R}$:

$$g(v) = \begin{cases} (e) \mapsto e & \text{if } v = \mathit{sleep} \\ (e) \mapsto e & \text{if } v = \mathit{study} \\ (e, \mathit{sleep}) \mapsto g(\mathit{sleep}) + e & \text{if } v = \mathit{focus} \\ (e, \mathit{focus}, \mathit{study}) \mapsto 0.6\, g(\mathit{focus}) + 0.4\, g(\mathit{study}) + e & \text{if } v = \mathit{score} \end{cases}$$

Here, *sleep* and *study* are exogenous variables whose values depend only on independent unobserved error terms e representing factors such as stress, natural ability, distractions, or imprecision of measuring a variable. The variable *focus* depends causally on *sleep*, and *score* is a weighted combination of *focus* and *study*, both also subject to their own unobserved terms.

Since each node's value depends on its parents' values, and the graph is assumed to be acyclic, the node functions can be evaluated in a topological order, yielding a value for every node and guaranteeing termination.

Different assignments of unobserved terms U correspond to different possible worlds consistent with the causal structure. Specifically, let $U : \mathcal{V} \to \mathcal{X}$ be an assignment of unobserved terms to nodes (equivalently, we will sometimes represent U or another assignment as a mapping, e.g., $\{u : \alpha, v : \beta\}$).

Each node function can be computed with knowledge of U and the structure of $\mathcal{G}$, since it can then determine the unobserved term and parents of the desired node. Thus, we write $f_U : \mathcal{V} \to X$ as the graph function that encodes information about each node function and computes a node's value using U as the unobserved-terms assignment for $\mathcal{V}$.

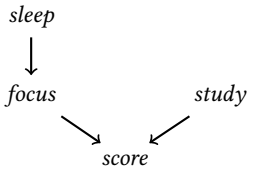


Fig. 2. (Left) This causal model encodes the intuition that amount of sleep affects focus level, and test score is affected by both focus and study time.

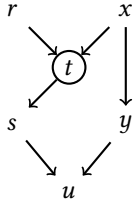


Fig. 3. (Right) $Z = \{t\}$, and $\mathrm{Anc}^*_Z(u) = \{u, s, y, x\}$. Note that although x has a blocked path to u through t, we only require the existence of any unblocked path.

For example, we would rewrite the above function g as

$$f_U(v) = \begin{cases} U(v) & \text{if } v \in \{sleep, study\} \\ f_U(sleep) + U(v) & \text{if } v = focus \\ 0.6\, f_U(focus) + 0.4\, f_U(study) + U(v) & \text{if } v = score. \end{cases}$$

The function f_U thus captures the entire semantics of the causal model under given assignments of unobserved terms.

3.2 Defining Semantic Separation

The semantic framework enables us to define an analogue of conditional independence in a way that aligns with intuition.

Informally, two nodes should be considered independent if changing the value of one has no effect on the value of the other. Under our model, the value of a node is determined by its function, so a natural semantic notion of independence would assert that altering the output of one node's function does not influence the output of the other's. We consider more generally *conditional* independence, of which independence is the case where the conditioning set is empty. In this context, conditioning on a set of nodes Z means we want to hold fixed the values of all nodes in Z while assessing the influence of u on v. To provide the fixed values, we introduce an additional set of assignments, A_Z, which maps each node in Z to its desired conditioned value.

Definition 3.1. We say that a world modeled by unobserved-terms assignments U **properly conditions on Z** if, for the given assignments A_Z, $f_U(z) = A_Z(z)$ for all $z \in Z$.

In other words, Z is the set of nodes being conditioned on, and A_Z specifies the fixed values for those nodes. For example, if $Z = \{\text{gender}\}$ and $A_Z = \{\text{gender} : \text{F}\}$, then a world with function f and unoberved-terms assignments U properly conditions on Z only if $f_U(\text{gender}) = \text{F}$.

Unlike probabilistic independence, at the level of functional semantics, each node's value depends deterministically on its parents and an unobserved error term. Our definition of semantic separation captures when such dependencies prevent causal influence from flowing between two nodes, given that certain variables must be preserved.

In particular, if u and v are semantically separated, then in a world where $f(u) = \alpha$ and all nodes in Z are properly conditioned, if we were to modify the world *minimally* so that $f(u) = \beta$ and all nodes in Z remain properly conditioned, then $f(v)$ would be unaffected. The randomness or error leading to these different settings can be absorbed into the unobserved-terms assignments, U.

We now dig into the meaning of "minimally": the goal is to ensure that the only possible influence on $f(v)$ comes from changes that causally descend from u. Since a node's value is affected only by its parents (and its own unobserved term), which are in turn only affected by their parents, we are led to restrict change to the *ancestors* of u. However, since conditioned nodes have fixed values, changes to ancestors of u can only affect $f(u)$ if they do not have to pass through conditioned nodes. We thus define the following notion.

Definition 3.2. Given a node u and a conditioning set Z, a node w is an **unblocked ancestor** of u if either $w = u$, or there exists a directed path from w to u such that no internal nodes on the path (including w) are in Z. The set of unblocked ancestors of u given Z is denoted $\text{Anc}^*_Z(u)$.

An example is shown in Figure 3.

When we intervene minimally on u, its change may propagate through the graph and inadvertently alter variables in the conditioning set Z. Since variables in Z must remain fixed, we may need to adjust other variables so that the values of Z are restored. Intuitively, this means undoing,

or *repairing*, any changes to conditioned variables that arise from the intervention, in order to model the concept that Z is being held constant structurally.

For example, consider determining how a drug influences health. Some influence may occur through blood-oxygen levels, while experimenters want to determine what other pathways exist. Therefore, they might compare subjects whose blood-oxygen levels are held fixed, counteracting the drug's effect on blood oxygen using some other well-established treatment. This secondary intervention restores the conditioned variable. In general, however, such adjustments may themselves propagate through the system and disturb other factors that are intentionally being held constant, requiring further corrections until the conditioned variables are restored.

We formalize this intuition as a sequence of repair steps that restore the nodes in Z. With this setup, we now define semantic separation. One way of seeing this definition is as defining a kind of *operational semantics* of actions that experimenters might take, such that semantic separation holds when any such experimenter actions that reach certain termination conditions have preserved certain values.

Definition 3.3. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, and let $u, v \in \mathcal{V}$. Let f be a graph function, let α, β be values, and let A_Z be conditioned assignments for Z. Let $U_0, U_1, ..., U_\ell$ be a *sequence* of unobserved-terms assignments, which satisfy the conditions below:

(1) *Initialization.* $f_{U_0}(u) = \alpha$, and f_{U_0} properly conditions on Z.
(2) *Catalyst update.* $f_{U_1}(u) = \beta$, and U_1 differs from U_0 only for $a \in \mathrm{Anc}^*_Z(u)$.
(3) *Reparative propagation.* For all $i = 2, ..., \ell$, U_i differs from U_{i-1} only for

$$a' \in \bigcup_{\substack{z \in Z \\ \exists a \in \mathrm{Anc}^*_Z(z), \\ U_{i-2}(a) \neq U_{i-1}(a)}} \mathrm{Anc}^*_Z(z).$$

(4) *Termination.* $1 \leq \ell \leq |\mathcal{V}| + 1$.
(5) *Restored conditioning.* $f_{U_\ell}(u) = \beta$, and f_{U_ℓ} properly conditions on Z.

We say that u and v are **semantically separated given Z** if, for all such $f, \alpha, \beta, A_Z, U_0, ..., U_\ell$, the value of v remains unchanged: $f_{U_0}(v) = f_{U_\ell}(v)$.

Definition 3.3 is somewhat intricate, quantifying over many auxiliary assignments, but this complexity is intentional; the definition must faithfully capture the semantic idea that a change to u must propagate through the model and, if it disturbs the conditioning set Z, can be repaired only by locally justified updates to unblocked ancestors of the disturbed nodes.

We now unpack the definition and build intuition for each requirement, showing that each condition is necessary to rule out spurious noncausal changes while allowing precisely those modifications that reflect legitimate causal propagation.

For instance, in the *initialization*, we establish a reference world to anchor our semantics based on unobserved-terms assignments U_0, in which the value of u is fixed at α, and the system is well-behaved with respect to Z.

In the *catalyst update*, we introduce the "minimal modification" to change the value of u to β, yielding new unobserved-terms assignments U_1 that produce the new value for u. We need the additional constraint on the nodes for which U_1 differs from U_0 in order to isolate *why* $f(v)$ changed in the case that the value of v does not stay constant. Specifically, if we did not constrain U_1 at all, then a change in $f(v)$ might be attributable not to $f(u)$ but to other aspects of the changing unobserved-terms assignments. For example, in the simplest case, if u and v are disconnected nodes (and thus should be semantically separated for any Z), we could still have $U_0(v) \neq U_1(v)$, which could cause a difference in $f(v)$ that is entirely unrelated to u. Restricting changes to only

unobserved terms of $\text{Anc}^*_Z(u)$ ensures that when we compare the two different worlds represented by U_0 and U_1, those differences are localized to the nodes that could actually affect u. In doing so, we rule out irrelevant sources of variation.

At a high level, these first two stages simply describe how to introduce a perturbation to u by changing only the unobserved terms of influence and observing whether the change affected v. To make the ideas more concrete, consider the following simple example.

Example 3.4. Consider the case of the following graph: $x \to u \to y \to v$, where $Z = \emptyset$. Note that u and v are d-connected given Z, since the path $u \to y \to v$ is unblocked. Intuitively, we would expect that u and v are *not* semantically separated, since u has this direct causal path to v.

Suppose we begin with unobserved terms given by U. According to our definition, a change to the value of u can be induced by modifying either $U(u)$ or $U(x)$ to get U', since $f(u)$ could depend on both $U(u)$ and $f(x)$ (using the notation of Definition 3.3, $U_0 = U$ and $U_1 = U'$). Such a change may propagate to y (such that $f_U(y) \neq f_{U'}(y)$), whose value depends on $f(u)$, and then further to v, which depends on $f(y)$. Hence, a perturbation of u can ultimately alter $f(v)$ (such that $f_U(v) \neq f_{U'}(v)$), showing that u and v are not semantically separated.

Of course, there exist specific functions f and unobserved-terms assignments U for which perturbing $f(u)$ happens not to affect $f(v)$. However, Definition 3.3 quantifies over all such assignments and functions, requiring that no possible change to u under the permitted conditions alter v. Therefore, the existence of even a single case where $f(v)$ changes is sufficient to conclude that u and v are not semantically separated.

Now, consider the same graph but with $Z = \{y\}$. In this case, u and v are d-separated by Z. Returning to our definition, we see that for all f, $f(v)$ depends only on $U(v)$ and $f(y)$. Since v is not an unblocked ancestor of u or any conditioned node, $U(v)$ cannot change. Furthermore, since $y \in Z$, the *restored conditioning* step requires that its value $f(y) = A_Z(y)$ stays fixed at the end of the process. Therefore, $f(v)$ remains invariant across all permissible changes, and u and v are semantically separated.

It may be tempting simply to define semantic separation by requiring that the value of v stays constant through only the first two stages, since they capture the idea that the value of u is perturbed, and v must go unchanged. However, this approach fails to account for how values of Z could affect $f(u)$; their effect will not be from the ancestors of u, since their passed-on values are fixed. This idea highlights a key insight: determining semantic separation requires not only a change to the value of u but also the *propagation* of that change throughout the rest of the graph. We are interested in whether the value of v stays constant once the change has been fully propagated. In particular, after the initial catalyst change to $f(u)$, if any values of Z are no longer properly conditioned, we allow changes to recondition them, leading to the stage of *reparative propagation.*

The (possibly empty, since ℓ could equal 1) sequence of unobserved-terms assignments $U_2, ..., U_\ell$ repairs any disrupted values in the conditioning set Z. In particular, for $i \geq 2$, each U_i is responsible for restoring the conditioned nodes that were affected by the preceding assignment U_{i-1}. During the propagation process, previously repaired nodes may in turn disturb others, requiring further adjustments. Consequently, the repairs proceed gradually through the sequence, with each U_i allowed to differ from U_{i-1} only for ancestors of the nodes in Z that were affected in the previous step. In this way, each set of assignments incrementally restores proper conditioning, using only changes justified by the preceding state.

Example 3.5. Consider the simple structure $u \to w \leftarrow v$ with $Z = \{w\}$, as shown in Figure 4. We would expect u and v not to be semantically separated; for instance, consider $f(w) := f(u) \oplus f(v)$, which is permitted since u and v are both parents of w. Suppose $A_Z(w) = 1$, and we change the

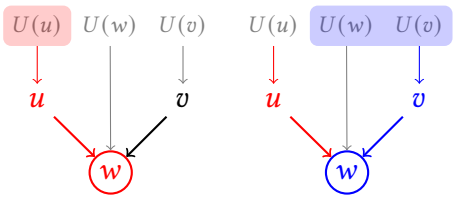


Fig. 4. (Left) *Catalyst update*: modifying $U(u)$ (highlighted) changes u, propagating to w and violating its conditioned value.
(Right): *Reparative propagation*: restoring w may use either $U(w)$ or $U(v)$ (highlighted). Here, repair is performed via $U(v)$, forcing a change in v.

value of u from 0 to 1. To preserve the conditioned value of w, the value of v must change from 1 to 0. Thus, the values of u and v are dependent on each other.

We can observe this dependence via our definition. In this small graph, all catalysts must originate from changing $U(u)$ to obtain U', since u has no unblocked ancestors besides itself (using the concrete f in the above paragraph, perhaps $f_U(u) := U(u) \bmod 2, U(u) = 0, U'(u) = 1$). This change affects $f(w)$, so we must perform a reparative-propagation step to restore it. The repair could come from $U'(v)$ or $U'(w)$, since changing $U'(u)$ again would simply revert u to its original value. (It is of course possible to have oscillating or cyclic sequences, repairing via $U'(u)$, which reperturbs $f(w)$, requiring yet another change, but here we consider the minimal sequence.) Suppose the change comes from $U'(v)$. As a result, $f(v)$ changes, showing that u and v are not semantically separated.

Example 3.5 aligns with the graphical intuition: u and v are d-connected in this graph since the collider w is its own conditioned descendant. However, it is arguably more useful to reason through their lack of semantic separation, as above, than to observe the more-technical and less-intuitive collider-descendant rule given in d-separation.

Example 3.6. Now consider a case with two consecutive conditioned nodes, such as the path $u \to z_1 \to z_2 \leftarrow v$, where $Z = \{z_1, z_2\}$, as shown in Figure 5. Note that u and v are d-separated because z_1 blocks the path between them as a conditioned mediator.

Suppose we change $U(u)$ to obtain U', affecting z_1, which in turn may affect z_2. The reparative-propagation condition prevents an alternate repair of z_2 via $U'(v)$, since no *unblocked* ancestors of z_2 changed from U to U'. Instead, restoring z_1 (which *does* have unblocked ancestors that changed) to its original value will automatically restore z_2.

In this example, our formulation of the reparative-propagation step blocks a potential noncausal path of influence, ensuring that dependencies cannot "skip through" consecutive conditioned nodes. This restriction illustrates how the semantic definition enforces local, causally justified repairs.

The sequence continues until all conditioned nodes z once again satisfy $f_{U_t}(z) = A_Z(z)$, after which we require that v remain unchanged between the original and fully repaired worlds. This requirement is captured in *restored conditioning*.

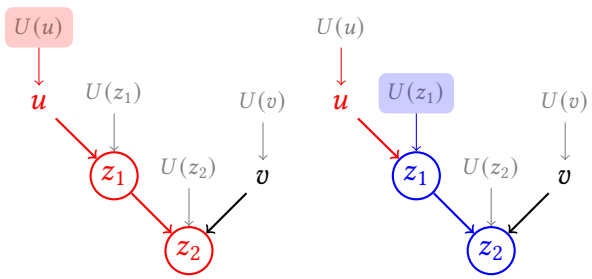


Fig. 5. (Left) *Catalyst update*: modifying $U(u)$ (highlighted) changes u, propagating through z_1 to z_2 and violating both conditioned nodes.
(Right): *Reparative propagation*: restoring z_1 may only use $U(z_1)$ (highlighted), which also restores z_2.

The *termination* condition ensures that the definition is well-founded, and the sequence is finite. Since at least one conditioned node must be repaired for each i, we can bound the length of the repair sequence $U_2, ..., U_\ell$ by the number of conditioned nodes, which is of course upper-bounded by the total number of nodes $|\mathcal{V}|$. As discussed in Example 3.5, this sequence can be *cyclic*, in that a repaired node becomes unconditioned again in a later set of assignments and must be repaired again. While cyclic sequences are not forbidden by Definition 3.3, the definition must hold for all sequences satisfying the criteria, so redundant or oscillating repair paths can be ignored.

Semantic separation thus holds exactly when, across all such perturbation-repair scenarios, the value of v remains invariant. Intuitively, changing u cannot affect v except through violations of Z, aligning directly with what d-separation expresses graphically. The next section will show that this semantic notion coincides precisely with d-separation, providing a formal bridge between graphical and semantic causal models.

3.3 Semantic Separation $\Longleftrightarrow$ d-Separation

Using the notion of semantic separation established in Definition 3.3, we present our central result:

Theorem 3.7. *The notions of semantic separation and* d*-separation coincide exactly. In particular, for a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two distinct nodes $u, v \in \mathcal{V}$, and a conditioning set $Z \subseteq \mathcal{V}$ with $u, v \notin Z$, u and v are semantically separated given Z (for all possible graph functions f and conditioning assignments A_Z) if and only if they are* d*-separated given Z in $\mathcal{G}$.*

Our results are stated for an arbitrary domain $\mathcal{X}$ of node values. We assume:

- $\mathcal{X}$ contains at least two distinct elements.
- Equality on $\mathcal{X}$ is decidable; that is, there exists a Boolean equality function satisfying reflexivity and symmetry.

We prove Theorem 3.7 by splitting it into its two logical directions. We show the forward direction via the contrapositive: assuming that there exists a d-connecting path from u to v, we define a specific graph function in which changing the value of u alters the value of v, thereby violating semantic separation. We prove the backward direction by showing that if u and v are not semantically separated, i.e., there exist unobserved-terms assignments satisfying the conditions of Definition 3.3 that propagate a change to the value of v, then we can show the existence of a d-connected path from u to v. The forward and backward directions are proven in Sections 5 and 6, respectively.

Before turning to the practical advantages of this semantic view, it is helpful to clarify that our framework indeed assumes experimenters may carefully tweak only certain variables (e.g., the ancestors of u) within a world, while others remain fixed. This abstraction captures the idealized design of controlled experiments; while in practice such intervention may be approximate, many experimental setups, whether randomized trials in medicine or controlled simulations in physics, aim to perform precisely this kind of manipulation. In this sense, our semantics provides a natural target for how such experiments can be represented formally.

Assuming Theorem 3.7 is true, we now examine situations in which the semantic definition offers benefits over the purely structural perspective offered by d-separation.

4 Advantages of Semantic Separation

4.1 Special Cases of the Definition

Definition 3.3 is not obviously more elementary than the standard definition of d-separation, which we called unintuitive. However, specializing our new definition to different cardinalities of Z yields simplified forms that one may find not much less intuitive than noninterference.

Consider first the case of $Z = \emptyset$, where it is easy to see that any repair step must be vacuous.

Corollary 4.1. *Given a graph $\mathcal{G}$ and two of its nodes u and v, they are* d*-separated without conditioning ($Z = \emptyset$) exactly when, under any compatible graph function (a graph function where each node's function depends only on its parents in the DAG), modifying any assignment U only by changing* the ancestors of *u can never modify the value of v.*

This definition is extremely close to classic noninterference, with the ancestors of u playing the role of the secret inputs and v the role of the public output. We have merely picked up a kind of thoroughness condition, forcing us to acknowledge every node upstream of u as a potential influence. (Classic noninterference is phrased in terms of black-box functions, with no concept of dataflow within, so it makes sense that the classic definition does not try to account for inputs causally influencing others.)

The true complexity of causal reasoning only becomes apparent when we need to account for confounding by choosing the right Z. Let us consider the special case of $Z = \{z\}$.

Corollary 4.2. *Given a graph $\mathcal{G}$ and two of its nodes u and v, they are* d*-separated conditioning on $\{z\}$ exactly when, under any compatible graph function, modifying any assignment U in either of the following ways can never modify the value of v.*

(1) Changing any ancestors of u not blocked by z, such that the value of z itself is not *affected.*
(2) Changing any ancestors of u not blocked by z, such that the value of z itself is *affected; followed by a single step of repair, changing only ancestors of z (since we assume a DAG, no relevant path to z can go through z internally, anyway) to reestablish the value of z without changing the value of u.*

4.2 Test Oracles for Experiments

The scientist's full repertoire depends on probability and statistics, which we have not yet folded into our formalization, but already we can justify some important ingredients in experiments. Consider the general problem of determining whether a causal DAG is a correct formalization of a real phenomenon. We may run experiments to try to *disprove* the theory that we modeled the phenomenon correctly with our DAG. Each run of an experiment measures some of the nodes in G (partial f_U for unknown U).

We want to know *when it is sound to declare the original DAG falsified, considering particular measurements obtained in some experiment.*

Corollary 4.3. *Suppose we are given a graph $\mathcal{G}$ and two of its nodes u and v that are* d*-separated without conditioning (as can be confirmed via a simple graph algorithm). Consider an experimental run, implicitly running in some graph function f and unobserved-terms assignments U, begun by measuring the values $f_U(u)$ and $f_U(v)$. Now intervene on u, producing U', in some way guaranteed to change only the U values for ancestors of u. If measuring v now confirms $f_U(v) \neq f_{U'}(v)$, then we have falsified the hypothesis that $\mathcal{G}$ models the world accurately.*

Proof. By Corollary 4.1. □

Now, the devil is in the details of how one intervenes during an experiment to ensure that only the target variables are manipulated (formally captured by the do-operator, which replaces the intervened variable's function with a constant so that it no longer depends on its usual parents [Pearl 2009]), while holding all other structural relationships constant. Ensuring that any real-world interventions do not change the unobserved factors of other nodes unintentionally is out of the scope of our framework. Nevertheless, it is edifying to prove a principle for drawing conclusions

from experiments in a purely deterministic setting: d-separation can be useful even in the absence of probability.

Corollary 4.4. *Suppose we are given a graph $\mathcal{G}$ and two of its nodes u and v that are* d*-separated conditioning on $\{z\}$. Consider an experimental run, implicitly running in some graph function f and unobserved-terms assignments U, begun by measuring the values $f_U(u)$, $f_U(v)$, and $f_U(z)$. Now intervene on u, producing U', in some way guaranteed to change only the U values for ancestors of u not blocked by z in $\mathcal{G}$, while also modifying the observed value of z. Then make a further intervention into U'' guaranteed to change only the U' values for ancestors of z, restoring the prior value of z without changing the value of u. If measuring v now confirms $f_U(v) \neq f_{U''}(v)$, then we have falsified the hypothesis that $\mathcal{G}$ models the world accurately.*

Proof. By Corollary 4.2. □

To illustrate these special cases, consider a simple example inspired by Figure 1b, relating caffeine consumption to GPA. Suppose an experimenter hypothesizes a causal diagram in which extra caffeine has no direct effect on GPA once courseload is controlled. Our formalism for $|Z| = 1$ provides a way to test this hypothesis: the experimenter repeatedly recruits new subjects, measures their GPAs, and applies the intervention (e.g., increasing caffeine intake). Whenever a subject's courseload changes because of the intervention, we modify it directly, say through a requirement to drop excess classes. After such repairs, we confirm that GPA never changes. If it does, then the causal diagram – and thus the scientific hypothesis – is falsified. In general, this procedure can be repeated to search systematically for counterexamples to the hypothesized causal model.

We have stated these theorems to cover what we might call *soundness* of a test oracle for experiments: if the experimenter intervenes in a certain way and collects measurements satisfying a property, then we know that $\mathcal{G}$ is inaccurate.

We might also aim for *completeness*: intuitively, if $\mathcal{G}$ is inaccurate, then some experiment of the given kind can be run to uncover that inaccuracy. The reality is not that simple, as Theorem 3.7 describes semantic separation *in all possible compatible graph functions f*, while a given sequence of experiments presumably only runs in the one f that describes reality accurately. That f may have bad properties that prevent our chosen kind of experiment from noticing an inconsistency. We thus instead obtain completeness in terms of all compatible worlds, rather than a single realized world. Prior work effectively axiomatized the independence requirements of these worlds; our result shows that the semantic property can be proved without assuming such axioms.

Our hope is that our refined semantic understanding of d-separation can be a stepping stone to proving explicitly probabilistic theorems about soundness of experimental designs that appeal to checking relevant d-separations. Having established the payoff of our central theorem, we return to its proof. We sketch the main ideas of each direction of implication, leaving most technical details to the appendices.

5 Forward Direction: Semantic Separation Implies d-Separation

We now prove the forward direction of Theorem 3.7 by contrapositive. Suppose that u and v are not d-separated given Z, so there exists a d-connected path from u to v in $\mathcal{G}$. We will show that u and v are not semantically separated given Z. Concretely, we will construct a graph function f and unobserved-terms assignments $U_0, ..., U_\ell$ that satisfy the five conditions of Definition 3.3 but result in different values for $f_{U_0}(v)$ and $f_{U_\ell}(v)$, thus violating semantic separation.

At a high level, the strategy is as follows: we define a specific graph function f that evaluates each node differently based on its structural role in the d-connected path, specifically whether it is a mediator, confounder, or collider. This function will be constructed such that all noncolliders

on the path are assigned the same value. Since both u and v are endpoints and thus noncolliders, this property ensures that $f(u) = f(v)$. We will then construct a valid sequence of unobserved-terms assignments that modifies u's value and causes the change to propagate through the path to v, ultimately changing $f(v)$ as well, which will demonstrate that u and v are not semantically separated and complete the contrapositive proof.

5.1 Constructing a Function to Equate Node Values

We examine simple cases of d-connected paths before generalizing to an arbitrary path.

5.1.1 A Chain of Mediators. We start with the simplest case: the path from u to v consisting of a single edge $u \to v$. Since u is a parent of v, we can define $f(v) := f(u)$ to propagate the value directly. Now suppose the path is a longer chain of mediators:

$$u \to m_1 \to m_2 \to \cdots \to m_k \to v.$$

In this case, we define $f(v) := f(m_k)$, $f(m_i) := f(m_{i-1})$ for $i = 2, ..., k$, and $f(m_1) := f(u)$. Each node's value depends only on its parent in the path, and so this definition satisfies the structure of a causal model while ensuring all values along the path are equal.

We must also ensure that such a function respects conditioning on Z; otherwise it cannot be used in Definition 3.3. In particular, if some descendant of a node on the path lies in Z, its value could be disrupted if the values along the path change. To prevent this improper conditioning, we simply define $f(z) := A_Z(z)$ for all $z \in Z$, ensuring that all conditioned nodes remain fixed and do not rely on any values of nodes in the path. Thus, we have shown that if u and v are connected by a path of all mediators (i.e., a directed path), then there is a graph function that equates their values.

Note that the assumption that the path is d-connected is crucial. If any $m_i \in Z$, then changing $f(u)$ could violate the conditioning of m_i. Furthermore, the change would not propagate past m_i to v, since $f(m_i) = A_Z(m_i)$ would be fixed.

The case of a directed path in the opposite direction (i.e., towards u) is symmetric; we flip the function definitions accordingly to ensure again that all nodes evaluate to the same value.

5.1.2 A Single Confounder. We now consider a simple fork: $u \leftarrow c \to v$, where c is a confounder. Since both u and v have c as a parent, we can define $f(u) := f(c)$ and $f(v) := f(c)$. This graph-function definition will equate all values along the path to equal the value of the confounder.

More generally, suppose the path is two chains of mediators connected by a single confounder:

$$u \leftarrow m_1 \leftarrow \cdots \leftarrow m_k \leftarrow c \to n_j \to \cdots \to n_1 \to v.$$

Then, we combine this idea with the ideas in Section 5.1.1, defining the values of the left chain to be equal to $f(m_k)$ and the right chain to be equal to $f(n_j)$, and finally $f(m_k) := f(c)$ and $f(n_j) := f(c)$. Again, all values are equated along the path.

5.1.3 A Single Collider. Suppose the path is $u \to c \leftarrow v$, where c is a collider. We wish to equate all noncollider nodes on the path (just u and v in this case). This case is the least intuitive because u and v are both parents of c, but any effect of the value of u on the value of v must occur through c, their shared child node.

By the definition of d-connectedness, c must have a descendant in Z. First, consider the case where $c \in Z$. Then, there is some assigned value $x := A_Z(c)$ which c must evaluate to in every properly conditioned setting of unobserved-terms assignments. Choose some $y \neq x$, using the assumption that node values come from a type with at least two elements. Then, we can define

$$f(c) := \begin{cases} x & \text{if } f(u) = f(v) \\ y & \text{otherwise,} \end{cases}$$

and any unobserved-terms assignments that properly condition on Z necessarily force $f(u) = f(v)$.

Now, suppose $c \notin Z$, so it has some descendant $d \in Z$. Let $d_1, ..., d_k$ be the intermediate nodes of the directed path from c to d, as shown in Figure 6. Using the mediator-chain strategy of Section 5.1.1, we can equate the values of nodes on c's descendant path:

$$f(d) := f(d_k), f(d_i) := f(d_{i-1}), f(d_1) := f(c),$$

so that $f(c) = f(d)$. We then define $f(c)$ in the same way as above, where $x := A_Z(d)$. Again, any unobserved-terms assignments that properly condition on Z force $f(u) = f(v)$.

5.1.4 General Construction of f^{path}. We now generalize the above ideas to an arbitrary d-connected path P from u to v. We construct a graph function f^{path} that forces all noncollider nodes on P to take on the same value in a setting of unobserved terms that properly conditions on Z.

Specifically, we partition the nodes in $\mathcal{V}$ into six sets and define a specific node function for each node based on the set into which it falls. Recall that each node function computes the node's value from its unobserved term and the values of its parents.

- S_1 (sources): nodes on P with no neighboring parents on the path. The node function simply copies the value of the unobserved term.
- S_2 (transmitters): every node on P with exactly one neighboring parent on the path. The node function simply copies the value of that parent along P.
- S_3 (colliders): every node on P whose two neighboring nodes on the path are both parents. The node function compares the values of those two parents. If they are equal, then the node function outputs $x := A_Z(d)$, where d is the conditioned descendant associated with the collider (possibly the collider itself). Otherwise, it outputs a distinct value $y \neq x$.
- S_4 (descendants): nodes along paths to conditioned descendants from colliders in P (choosing one path per collider). The node function copies the value of the node's parent along the chosen descendant path.
- S_5 (Z-residual): conditioned nodes not included in $\bigcup_{i=1}^{4} S_i$. The node function copies the node's assigned value in A_Z.
- S_6 (residual): remaining nodes. Assign any arbitrary node function.

Here, all confounders lie in S_1 and mediators in S_2. The endpoints u and v fall into either S_1 or S_2 depending on the directions of the edges adjacent to them. For example, using the graph in Figure 7,

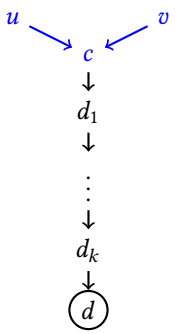


Fig. 6. (Left) Assuming the path highlighted in blue is d-connected, then c must have a descendant in Z. Assuming $c \notin Z$, it must have a descendant $d \in Z$ and a descendant path to d.

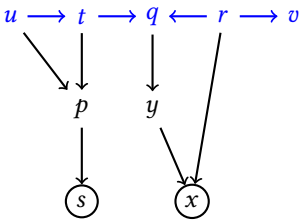


Fig. 7. (Right) Consider the d-connected path from u to v highlighted in blue, where $Z = \{s, x\}$. Then, the partition is as follows: $S_1 = \{u, r\}$, $S_2 = \{t, v\}$, $S_3 = \{q\}$, $S_4 = \{y, x\}$, $S_5 = \{s\}$, $S_6 = \{p\}$.

we would assign

$$f^{\text{path}}(t) := f^{\text{path}}(u), \qquad f^{\text{path}}(v) := f^{\text{path}}(r), \qquad f^{\text{path}}(x) := f^{\text{path}}(y)$$

$$f^{\text{path}}(y) := f^{\text{path}}(q), \qquad f^{\text{path}}(q) := \begin{cases} A_Z(x) & \text{if } f(t) = f(r) \\ \rho & \text{otherwise, for some } \rho \neq A_Z(x). \end{cases}$$

For each collider $c \notin Z$, we select a descendant $d \in Z$ (which exists because P is d-connected) and add all nodes on the directed path from c to d (excluding c, including d) to S_4. However, an issue arises: we know that sets S_1, S_2, S_3 are disjoint since we assume the path from u to v is acyclic, and thus each node on the path is categorized into exactly one set. Then, we can assign each node in each set a different node function. However, nodes in S_4 may also be on P or appear on multiple descendant paths, which would prevent the clean assignment of node functions.

Fortunately, whenever a d-connected path exists, there also exists one with all descendant paths disjoint from each other and from the path itself. The existence proof appears in Appendix C. We can thus assume, without loss of generality, that the sets $S_1, ..., S_6$ form a disjoint partition of $\mathcal{V}$, and we can thus fully construct f^{path}. The Rocq definition is provided in Appendix D.

We can see that under f^{path}, the values of the nodes depend on the sources; each source will propagate its value to any neighboring transmitters. Thus, the values of all sources, determined by their unobserved terms, must agree.

Definition 5.1. For any α, we say U is **source-fixed to α** if $U(w) = \alpha$ for all sources $w \in S_1$.

Given any source-fixed U, f_U^{path} indeed equates noncollider values. At a high level, a node in S_2 copies its parent's value, and chains of S_2 nodes are anchored by S_1 nodes, whose values are set directly by the unobserved terms, which are all equal since U is source-fixed. Furthermore, since any two chains of transmitters that collide at an S_3 node take on the same value, the S_3 node will take on the correct A_Z value to pass on to its conditioned descendant. Thus, for source-fixed U, f^{path} also properly conditions on Z. We formally show these two results in Appendix E.

Thus, with source-fixed unobserved-terms assignments U, we have a function that properly conditions on Z and forces the values of all noncollider nodes—most importantly, the values of u and v—to be equal. We next construct a sequence of assignments that ensures the propagation of a value from u to v, violating semantic separation.

5.2 The Sequence of Unobserved-Terms Assignments

We aim to use f^{path} to prove that u and v are *not* semantically separated given Z, as described in Definition 3.3. Choose some $\alpha \neq \beta$. If U_0 is source-fixed to α, we will have $f_{U_0}^{\text{path}}(u) = f_{U_0}^{\text{path}}(v) = \alpha$. If we can create a sequence $U_1, ..., U_\ell$ satisfying the requirements of Definition 3.3 such that U_ℓ is source-fixed to β, then $f_{U_\ell}^{\text{path}}(u) = f_{U_\ell}^{\text{path}}(v) = \beta$, and notably $f_{U_0}^{\text{path}}(v) \neq f_{U_\ell}^{\text{path}}(v)$, proving that u and v are not semantically separated given Z.

We again examine simple cases of a d-connected path from u to v before providing the general construction of such a sequence.

5.2.1 A Directed Edge. In the case that P is a single directed edge $u \to v$, then $S_1 = \{u\}$. We can simply define $U_0 = \{u : \alpha\}$ and $U_1 = \{u : \beta\}$, where the assignments for other nodes are arbitrary (but identical) for both U_0 and U_1. Note that U_0 and U_1 indeed differ only for members of $\text{Anc}_Z^*(u)$ (only u). Here, $\ell = 1$, and $f_{U_0}^{\text{path}}(v) \neq f_{U_\ell}^{\text{path}}(v)$, so u and v are not semantically separated.

5.2.2 A Single Confounder. Now consider the case that P is a simple fork $u \leftarrow w \to v$. Here, $S_1 = \{w\}$. Define $U_0 = \{w : \alpha\}$, where other nodes are assigned arbitrarily. Note that changing the

unobserved term of u will not cause any change in $f^{\text{path}}(u)$, since $f^{\text{path}}(u)$ depends on the value of w. However, w is an unblocked ancestor of u. Thus, we can define

$$U_1(w') := \begin{cases} \beta & \text{if } w' = w \\ U_0(w') & \text{otherwise.} \end{cases}$$

Note that U_0 and U_1 are both source-fixed (to α and β, respectively), so if we let $\ell = 1$, we once again see that U_0 and $U_1 = U_\ell$ satisfy the requirements from Definition 3.3, and thus u and v are not semantically separated.

5.2.3 A Single Collider. Now, consider the case that P has a single collider: $u \to w \leftarrow v$, where $w \in Z$. Here, $S_1 = \{u, v\}$. Define $U_0 = \{u : \alpha, v : \alpha\}$. Define

$$U_1(w') := \begin{cases} \beta & \text{if } w' = u \\ U_0(w') & \text{otherwise,} \end{cases} \quad \text{and} \quad U_2(w') := \begin{cases} \beta & \text{if } w' = v \\ U_1(w') & \text{otherwise.} \end{cases}$$

Note that under U_1, we are no longer properly conditioning on Z. In particular, $f^{\text{path}}_{U_1}(u) = U_1(u) = \beta$, but $f^{\text{path}}_{U_1}(v) = U_1(v) = \alpha$. Recall that $f^{\text{path}}_{U_1}(w) = A_Z(w)$ only if $f^{\text{path}}_{U_1}(u) = f^{\text{path}}_{U_1}(v)$.

We thus perform a reparative-propagation step, defining U_2. Note that U_2 differs from U_1 for only $v \in \text{Anc}^*_Z(w)$, which satisfies Condition 3 of Definition 3.3. We let $\ell = 2$, and note that U_ℓ is source-fixed to β. Thus, $f^{\text{path}}_{U_\ell}(v) = \beta \neq f^{\text{path}}_{U_0}(v)$, so u and v are not semantically separated.

5.2.4 General Construction of Sequence. These simple cases show us that the propagation of a change in the value of u can occur via the sources in the path (the nodes in S_1). In particular, if we change the unobserved terms of the sources, one-by-one, until the unobserved-terms assignments are source-fixed, and the value of v is changed, then we can show that u and v are not semantically separated.

Concretely, suppose there are ℓ sources, and suppose they are organized in order of their appearance in P, such that $S_1 = [s_1, ..., s_\ell]$. Choose some $\alpha \neq \beta$. Define U_0 to be any unobserved-terms assignment source-fixed to α. Then define the following sequence of unobserved-terms assignments, for $i = 1, ..., \ell$:

$$U_i(w) = \begin{cases} \beta & \text{if } w = s_i \\ U_{i-1}(w) & \text{otherwise} \end{cases} \tag{5.1}$$

Note that for all edge orientations of P, S_1 will have at least one node. Thus, the sequence will contain at least U_1, as needed for Definition 3.3.

The sequence also meets the remaining conditions of Definition 3.3. In particular, the reparative-propagation step (Condition 3) holds because each triple of consecutive U_i, U_{i+1}, U_{i+2} changes the unobserved terms for only two consecutive sources, which must be separated by a collider in the path, of which the two sources are both unblocked ancestors. The full proof is given in Appendix F.

6 Backward Direction: d-Separation Implies Semantic Separation

We now prove the backward direction of Theorem 3.7 by contrapositive: assume that u and v are not semantically separated, so there exists some graph function f and sequence of unobserved-terms assignments $U_0, ..., U_\ell$ satisfying the conditions of Definition 3.3, such that $f_{U_0}(v) \neq f_{U_\ell}(v)$. We then use this sequence to show the existence of a d-connected path given Z from u to v, which will show that the two nodes are not d-separated.

6.1 Change Originates From Unblocked Ancestors

While in Section 5, we leveraged a specific d-connected path to propagate a change to v, we now need to consider what a change in a function's value must imply about the structure of the graph.

In particular, recall that a node's value depends on its unobserved term and the values of its parents. Thus, if $f_U(v) \neq f_{U'}(v)$ for some U, U', then it must be true that either $U(v) \neq U'(v)$, or $f_U(a) \neq f_{U'}(a)$ for some $a \in \mathrm{Pa}(v)$. In the latter case, it again must be true that $U(a) \neq U'(a)$, or $f_U(a') \neq f_{U'}(a')$ for some $a' \in \mathrm{Pa}(a)$. Since $\mathcal{G}$ is acyclic, this chain of parents must eventually terminate at a node whose unobserved term in U differs from its unobserved term in U'. Furthermore, note that each node in the chain is not in Z, since $f(z)$ is fixed for all $z \in Z$.

We formally describe this conclusion in the following lemma:

Lemma 6.1. *For any graph function f and two unobserved-terms assignments U, U', such that f_U and $f_{U'}$ both properly condition on Z, if $f_U(w) \neq f_{U'}(w)$ for some $w \in \mathcal{V}$, then there exists a node $a \in \mathrm{Anc}^*_Z(w)$ such that $U(a) \neq U'(a)$.*

Proof. See Appendix G. □

Lemma 6.1 already leads us towards a d-connected path to v because it points us to a specific unblocked ancestor of v. In particular, if two nodes share an unblocked ancestor, then they are automatically d-connected, since we can enumerate the possible configurations that can occur: either the shared ancestor coincides with one of the two nodes, or a common unblocked ancestor acts as a confounder connecting the two nodes via two directed paths. In either case, all mediators are not conditioned on. In the second case, the confounder (the unblocked ancestor itself) is also not conditioned on. This result is shown formally in Appendix G.

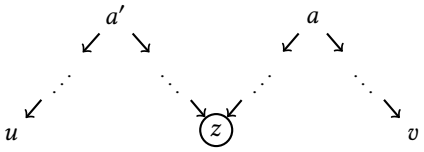
6.2 d-Connected Paths For Short Sequences

We sketch the process of finding a d-connected path between u and v for short sequences of assignments that satisfy the conditions of Definition 3.3 but change the value of v.

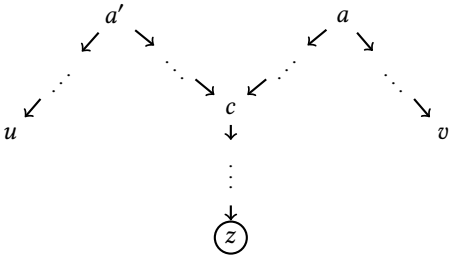
Suppose $\ell = 1$, so we change the value of v between U_0 and U_1. Then, since U_0 and U_1 both properly condition on Z, Lemma 6.1 tells us that there exists a node $a \in \mathrm{Anc}^*_Z(v)$ such that $U_0(a) \neq U_1(a)$. However, U_0 and U_1 are constrained such that they only differ for values in $\mathrm{Anc}^*_Z(u)$. Thus, u and v share an unblocked ancestor, so they must be d-connected.

Now, consider the case that $\ell = 2$, so $f_{U_0}(v) \neq f_{U_2}(v)$. Again, by Lemma 6.1, there exists a node $a \in \mathrm{Anc}^*_Z(v)$ such that $U_0(a) \neq U_2(a)$. If $U_1(a) = U_2(a)$, then by identical logic to the case $\ell = 1$, u and v are d-connected. Otherwise, if $U_1(a) \neq U_2(a)$, then there must exist $z \in Z$ such that $a \in \mathrm{Anc}^*_Z(z)$, and there is an $a' \in \mathrm{Anc}^*_Z(z)$ such that $U_0(a') \neq U_1(a')$. Then, $a' \in \mathrm{Anc}^*_Z(u)$. We can then construct the path shown in Figure 8a. We must consider the possibility that the directed paths making up the path overlap each other, which we handle formally in Appendix I. One particularly interesting possible overlap is between the directed paths $a' \cdots z$ and $a \cdots z$. Note that if they do not overlap, then z is a collider in the path, and $z \in Z$, so the path is indeed d-connected. It is however possible that the paths overlap at some node c and take the same path to z. Luckily, this resulting path exactly mimics the setup of d-connectedness for a collider that has a descendant in Z, as shown in Figure 8b.

Thus far we have only constructed examples involving paths with at most one collider and two confounders. To establish full equivalence to d-separation, we expect to rely on d-connected paths of arbitrary structure with any number of mediators, confounders, and colliders. This observation provides intuition behind why our definition of semantic separation requires a *sequence* of assignments rather than a single step: the change must propagate through potentially many intermediate nodes, depending on the structure of $\mathcal{G}$. In the following section, we generalize this reasoning to



(a) The two-confounder path constructed from unblocked ancestors a' and a. Note that by construction, the path is d-connected.



(b) If the paths overlap at collider c, then the path is still d-connected since z is a conditioned descendant.

Fig. 8. For any u, v such that v's value is affected by a change in u with a sequence of unobserved-terms assignments U_0, U_1, U_2, we can construct a d-connected path from u to v using $z \in Z$ and shared ancestors a', a.

arbitrary-length sequences that satisfy the constraints in Definition 3.3, thereby leveraging the full expressive power of d-connectedness.

6.3 Generalizing to Arbitrary-Length Sequences

Intuitively, for $i \geq 1$, each transition from U_i to U_{i+1} changes an unblocked ancestor whose influence passes through a shared descendant $z \in Z$. Each transition introduces a collider, and concatenating the directed paths from these ancestors to their descendants yields a d-connected path, showing that u and v must be d-connected.

A natural strategy for generalizing this logic to arbitrary-length sequences $U_0, ..., U_\ell$ is to attempt induction on ℓ. However, this approach does not directly succeed. In particular, let the induction hypothesis be that any sequence with $\ell = n$ showing semantic dependence between two nodes implies a d-connected path between them. For a sequence with $\ell = n+1$, one might try applying the hypothesis to the subsequence $(U_1, U_2, \ldots, U_{n+1})$ to get a path from some $z \in Z$ to v, then prepend a path from u to z. However, this approach fails because U_1 may differ from U_0 only for ancestors of u, while later steps may involve changes across ancestors of multiple $z_i \in Z$. Since there is no guarantee of a d-connected path from each of these z_i to v, we cannot inductively extend the argument. Applying the hypothesis to the prefix $(U_0, U_1, \ldots, U_n)$ encounters a different problem: U_{n+1} must fully recondition on Z, whereas intermediate steps need not, which once again prevents a clean inductive step.

Importantly, the sequence need not be minimal: v may change earlier than the final step. If v changes after U_1, some $z \in Z$ must have changed first, enabling v's downstream change. This observation motivates the correct inductive strategy. In the base case, v changes directly due to U_1, implying a shared unblocked ancestor with u. Otherwise, v changes through a node $z \in Z$, and by locating the step where z was first affected, we can apply induction: either z itself shares an unblocked ancestor with u, or its change was triggered by the repair of another node z', recursively extending the path toward u. Full details are provided in Appendix H.

By concatenating this inductively obtained path from u to z with the path from z to v, we construct a d-connected path from u to v. Full details of this construction and its role in completing the backward direction appear in Appendix I, thus establishing the central equivalence.

7 Related Work

We build on the formalism of causal diagrams that Pearl [1988, 1995] introduced. Geiger, Verma, and Pearl [Geiger et al. 1990] first established the equivalence between d-separation and a semantic notion based on three types of independence: probabilistic, relational, and correlational. We offer a deterministic perspective. A book [Pearl 2009] and a survey article [Pearl 2010] are helpful to overview more-recent developments. The semantics style that we use in this paper, where graph nodes are given meanings via deterministic functions, comes from viewing causal diagrams as encoding information about conditional independences [Pearl 1988].

7.1 Program Analysis for Scientific Validity

PlanOut [Bakshy et al. 2014] is a DSL for specifying and executing large-scale online experiments. Planalyzer [Tosch et al. 2019] checks internal-validity violations in PlanOut scripts and generates statistical-analysis plans for estimand calculations. These projects are not grounded in world models, while we aim to develop foundations for such grounding through semantics of causal networks. Similarly, Dagitty [Textor et al. 2011] enables graphical causal analysis but requires manual alignment by users of experiments and statistical analyses with graphs, whereas we aim to build towards formally proving such correspondences.

Research on synthesizing statistical analyses spans HCI and statistics but is often domain-specific (e.g., Experiscope [Guimbretière et al. 2007] for sensor logs, Statsplorer [Wacharamanotham et al. 2015] for teaching statistics) or supports limited data-collection strategies (e.g., Touchstone [Mackay et al. 2007], Touchstone2 [Eiselmayer et al. 2019]). The Automatic Statistician [Lloyd et al. 2014] is a general-purpose approach but relies solely on data, compromising external and internal validity.

The Tea [Jun et al. 2019], Tisane [Jun et al. 2022], and rTisane [Jun et al. 2024] projects are most closely related to our work, as they (i) assess statistical-analysis synthesis through the lens of scientific validity and (ii) introduce a formal-methods perspective to reasoning about validity. However, their focus is on establishing that statistical models can be synthesized from input causal graphs in a way that is usable by end-users who are not causality experts. As such, these systems feature higher-level DSLs for eliciting domain-specific causal assumptions and apply recommendations from the statistics literature to construct regression models. They neither reason about the semantic properties of causal diagrams nor reconcile functional and graph-theoretic views of causality. Instead, their implementations operate on graph patterns (e.g., identifying mediators, confounders, and colliders) without specifying what these structures mean semantically about the assumed world.

Our work focuses on establishing a clear semantic foundation of causality. In our envisioned scientific reasoning stack, our work provides lower-level foundations on which systems such as Tea, Tisane, and rTisane could build. In the future, such systems could leverage our framework to obtain provably valid guarantees rather than relying solely on informal statistical "best practices."

7.2 Probability in Semantics and Verification

One major use of probability in programming languages is in probabilistic programming, introduced with the Church programming language [Goodman et al. 2008] and remaining an active area of study in the programming-languages research community (e.g. with Anglican [Tolpin et al. 2016]) and statistical practice (e.g. with Stan [Carpenter et al. 2017]). Kozen [1981] gave an early formal semantics of probabilistic programs, McIver and Morgan [2005] developed program-verification methods in this domain, and a variety of verification work has followed, for instance in relational reasoning for privacy properties [Barthe et al. 2012]. These two bodies of work use probability

explicitly to state their results, while our work shows how a celebrated probabilistic theorem can also be given a deterministic characterization.

7.3 Program Analysis and Information-Flow Security

Abstract interpretation [Cousot and Cousot 1977] is an influential early formalism, assigning meanings to summaries of program state computed by dataflow analysis [Kildall 1973] with uniform proof that those summaries are accurate. A starting point for our work was trying to attach similar *semantic* correctness conditions to forms of analysis from causal inference, starting with d-separation.

The closest such semantic condition from the literature is noninteference as proposed by Goguen and Meseguer [1982], creating the basis for almost all later work on information-flow security. Dual versions of noninterference capture both confidentiality (an adversary does not learn secret values of variables) and integrity (an adversary cannot influence the values of important variables). An especially well-developed proof approach is *security types* [Myers 1999], which e.g., associate program variables with levels of secrecy and use static analysis to check that the types are respected. Similar guarantees can also be provided through run-time analysis, as developed in both operating systems [Zeldovich et al. 2006] and high-level programming languages [Stefan et al. 2011].

Our work enables an analogy between, on the one hand, security types and noninterference; and, on the other hand, d-separation and our new semantic characterization. Quantitative information-flow security [Gray 1991] has also been studied for some time in the context of probabilistic state machines and offers inspiration for our planned extensions toward probability and statistics.

8 Limitations and Future Work

Our goal is to develop formal foundations for reasoning about causal diagrams as part of a broader theory of experiment verification. In particular, semantic separation captures a basic task in scientific reasoning: determining when one variable cannot causally influence another under assumed relationships. In this sense, our work isolates a *structural* layer of reasoning about causality.

Future work should build up to the explicitly probabilistic and statistical setting needed to characterize real experiments. The "full stack" of scientific reasoning incorporates not only probability and statistics (e.g., assuming a normal distribution for each unobserved quantity) but also further assumptions about possible f values (e.g., perhaps every node function must be linear). We see several key ingredients of scientific analysis yet to be justified through future work.

- Falsifying graphs as world models is a good start, but in general we want to run experiments to narrow the sets of possible graph functions. For instance, if we assume graph functions are linear, we can learn tighter intervals containing their true coefficients.
- Real measurements often have continuous domains, making any principle questionable when it appeals to precise equality or inequality, motivating use of assumptions about probability distributions behind common statistical methods.
- As mentioned above, it may be difficult to convince ourselves that experimental procedures only modify unobserved terms relevant to particular graph nodes, or we may need to run more-observational studies that depend on sampling subjects with particular inherent properties. Randomized selection of subjects and assignment to conditions (e.g., treatment vs. control) then become important.

We are optimistic that the foundations we have begun to develop will help characterize the building blocks of future probabilistic frameworks in that vein. Today, PLDI papers often include mechanized proofs of their key claims, and perhaps some day papers throughout the natural and social sciences can have associated mechanized proofs of soundness for their experiment designs.

Artifact-Availability Statement

The Rocq development [Zhang et al. 2026b] was evaluated and is publicly available.

References

Eytan Bakshy, Dean Eckles, and Michael S Bernstein. 2014. Designing and deploying online field experiments. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 283–292.

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Philadelphia, PA, USA) *(POPL '12)*. Association for Computing Machinery, New York, NY, USA, 97–110. doi:10.1145/2103656.2103670

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software* 76, 1 (2017). doi:10.18637/jss.v076.i01

Carlos Cinelli, Andrew Forney, and Judea Pearl. 2020. A crash course in good and bad controls. *Sociological Methods & Research* (2020), 00491241221099552.

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California) *(POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. doi:10.1145/512950.512973

Alexander Eiselmayer, Chatchavan Wacharamanotham, Michel Beaudouin-Lafon, and Wendy Mackay. 2019. Touchstone2: An Interactive Environment for Exploring Trade-offs in HCI Experiment Design. (2019).

Dan Geiger, Thomas Verma, and Judea Pearl. 1990. d-separation: From theorems to algorithms. In *Machine intelligence and pattern recognition*. Vol. 10. Elsevier, 139–148.

J. A. Goguen and J. Meseguer. 1982. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy*. 11–11. doi:10.1109/SP.1982.10014

Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI*.

J.W. Gray. 1991. Toward a mathematical foundation for information flow security. In *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. 21–34. doi:10.1109/RISP.1991.130769

François Guimbretière, Morgan Dixon, and Ken Hinckley. 2007. ExperiScope: an analysis tool for interaction data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1333–1342.

Guido W. Imbens and Donald B. Rubin. 2015. *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. Cambridge University Press, New York.

Eunice Jun, Maureen Daum, Jared Roesch, Sarah E Chasins, Emery D Berger, Rene Just, and Katharina Reinecke. 2019. Tea: A High-level Language and Runtime System for Automating Statistical Analysis. In *Proceedings of the 32nd Annual Symposium on User Interface Software and Technology*. ACM.

Eunice Jun, Edward Misback, Jeffrey Heer, and René Just. 2024. rTisane: Externalizing conceptual models for data analysis prompts reconsideration of domain assumptions and facilitates statistical modeling. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–16.

Eunice Jun, Audrey Seo, Jeffrey Heer, and René Just. 2022. Tisane: Authoring Statistical Models via Formal Reasoning from Conceptual and Data Relationships. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–16.

Gary A. Kildall. 1973. A Unified Approach to Global Program Optimization. In *Proceedings of the 1st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. 194–206. doi:10.1145/512927.512945

Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350. doi:10.1016/0022-0000(81)90036-2

James Lloyd, David Duvenaud, Roger Grosse, Joshua Tenenbaum, and Zoubin Ghahramani. 2014. Automatic construction and natural-language description of nonparametric regression models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

Wendy E Mackay, Caroline Appert, Michel Beaudouin-Lafon, Olivier Chapuis, Yangzhou Du, Jean-Daniel Fekete, and Yves Guiard. 2007. Touchstone: exploratory design of experiments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1425–1434.

Annabelle McIver and Carroll Morgan. 2005. Abstraction, Refinement and Proof for Probabilistic Systems. In *Springer Monographs in Computer Science*.

Andrew C. Myers. 1999. JFlow: practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, USA) *(POPL '99)*. Association for

Computing Machinery, New York, NY, USA, 228–241. doi:10.1145/292540.292561

Judea Pearl. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann. First edition.

Judea Pearl. 1995. Causal Diagrams for Empirical Research. *Biometrika* 82, 4 (1995), 669–688. doi:10.1093/biomet/82.4.669

Judea Pearl. 2009. *Causality: Models, Reasoning, and Inference* (2nd ed.). Cambridge University Press.

Judea Pearl. 2010. An Introduction to Causal Inference. *The International Journal of Biostatistics* 6, 2 (2010). doi:10.2202/1557-4679.1203

Judea Pearl and Dana Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect* (1st ed.). Basic Books, Inc., USA.

Donald B. Rubin. 1974. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology* 66, 5 (1974), 688–701. doi:10.1037/h0037350

William Shadish, Thomas D Cook, and Donald Thomas Campbell. 2002. *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin Boston, MA.

Deian Stefan, Alejandro Russo, John C. Mitchell, and David Mazières. 2011. Flexible dynamic information flow control in Haskell. In *Proceedings of the 4th ACM Symposium on Haskell* (Tokyo, Japan) *(Haskell '11)*. Association for Computing Machinery, New York, NY, USA, 95–106. doi:10.1145/2034675.2034688

Johannes Textor, Juliane Hardt, and Sven Knüppel. 2011. DAGitty: a graphical tool for analyzing causal diagrams. *Epidemiology* 22, 5 (2011), 745.

David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. 2016. Design and Implementation of Probabilistic Programming Language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages* (Leuven, Belgium) *(IFL 2016)*. Association for Computing Machinery, New York, NY, USA, Article 6, 12 pages. doi:10.1145/3064899.3064910

Emma Tosch, Eytan Bakshy, Emery D Berger, David D Jensen, and J Eliot B Moss. 2019. Planalyzer: Assessing threats to the validity of online experiments. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–30.

Chat Wacharamanotham, Krishna Subramanian, Sarah Theres Volkel, and Jan Borchers. 2015. Statsplorer: Guiding novices in statistical analysis. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2693–2702.

Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. 2006. Making Information Flow Explicit in HiStar. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*. USENIX Association, Seattle, WA. https://www.usenix.org/conference/osdi-06/making-information-flow-explicit-histar

Anna Zhang, Qinglan Luo, London Bielicke, Eunice Jun, and Adam Chlipala. 2026a. Causality and Semantic Separation. arXiv:2604.22041 [cs.PL] https://arxiv.org/abs/2604.22041

Anna Zhang, Qinglan Luo, London Bielicke, Eunice Jun, and Adam Chlipala. 2026b. *Causality and Semantic Separation: Artifact*. doi:10.5281/zenodo.19985589

Received 2025-11-13; accepted 2026-04-03