Formalizing Causal Models Through the Semantics of Conditional Independence

by

Anna Zhang

S.B. Computer Science and Engineering and Mathematics, MIT, 2025

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by: Anna Zhang
Department of Electrical Engineering and Computer Science
May 16, 2025

Certified by: Adam Chlipala
Arthur J. Conner Professor of Computer Science, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Formalizing Causal Models Through the Semantics of Conditional Independence

by

Anna Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Many foundational tools in causal inference are based on graphical structure and can involve complex conditions that obscure the underlying causal logic. Given the inherent complexity and subtlety of cause-and-effect phenomena, establishing formal guarantees about these tools is both challenging and important. This thesis presents a semantics-driven formalization of causal models within the Coq proof assistant, enabling precise, mechanized reasoning about causal relationships. Central to this work is a new function-based definition of conditional independence, which captures how changes propagate through a causal graph. We prove that this semantic notion is equivalent to the standard graphical criterion of d-separation, thereby establishing a rigorous bridge between structural and semantic interpretations of independence. The formalization includes a library of graph-theoretic and causal-reasoning tools, encompassing key concepts such as mediators, confounders, and colliders. By linking the syntactic and semantic perspectives on causality, this work lays a robust foundation for formally verifying causal assumptions and guiding experimental design.

Thesis supervisor: Adam Chlipala
Title: Arthur J. Conner Professor of Computer Science

Acknowledgments

I am grateful to my advisor, Adam Chlipala, for introducing me to formal verification and for trusting me with a new project. His support and guidance throughout this research, especially during moments of uncertainty, made it not only possible but also enjoyable and rewarding.

I would also like to thank Eunice Jun at UCLA for sharing so much expertise. As someone completely new to causal inference, I learned a lot from her ideas, feedback, and encouragement.

To the rest of our research group, London Bielicke at UCLA and Emery Berger at UMass Amherst, thank you for your feedback and discussions in our weekly meetings. I appreciate both the academic insights and the fun conversations that we've shared.

I am also grateful to Mike Sipser for the opportunity to be a TA for 18.404 in the fall. It provided funding that enabled me to pursue an MEng and, more importantly, an incredibly meaningful teaching experience.

Finally, thank you to my family for their steady support and to my friends for being by my side through the highs and lows of the past four years.

Contents

List of Figures

Chapter 1

Introduction

Formal verification is a methodology in computer science for mathematically proving that a system satisfies its specifications under all valid conditions. One tool that enables formal verification is Coq, an interactive theorem prover that provides a rigorous logical framework for formalizing concepts and constructing proofs [1]. Formal verification provides a level of precision and certainty often unachievable through empirical methods alone, making it widely applicable in designing software and hardware systems where reliability, safety, and correctness are critical.

We apply formal verification to the field of causal inference, the process of reasoning about cause-and-effect relationships between variables. *The Book of Why* [2] highlights that many aspects of causal analysis, such as counterfactual reasoning, require intuitive leaps that are innate to humans but go beyond the capabilities of conventional computational approaches. However, these human-driven methods are often susceptible to error due to unaccounted biases, confounding variables, and incorrect assumptions. Randomized experiments, when feasible, help alleviate some of these concerns. For instance, randomly assigning participants to treatment groups in a drug trial reduces the influence of hidden confounding variables. However, such interventions are not always possible or ethical. Causal inference thus traditionally relies on assumptions and intuitive reasoning, which lack formal guarantees.

We develop a Coq framework to integrate formal verification with causal inference, leading toward the goal of improving the accuracy and robustness of causal models and the experimental conclusions derived from them. In addition to a Coq representation of causal models and a library of graph- and causal inference-related functions with correctness proofs, this thesis presents a function-based formal semantics of causal models and a formally verified equivalence between the semantic and syntactic notions of the widely used term "conditional independence." Together, these results demonstrate how formal methods can uncover hidden assumptions and increase the rigor of causal reasoning. The full Coq implementation including all definitions and mechanized proofs discussed in this thesis is available at https://github.com/annazhang03/causal_models_formalization.

1.1 Fundamental Concepts in Causal Inference

Understanding causality 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" [2] is helpful for categorizing these levels of causal understanding:

- **Level 1: Association.** This level concerns seeing patterns in data with questions like "What behaviors are associated with high grades?" These can often be answered with statistical correlations alone.
- **Level 2: Intervention.** This level goes beyond observation with questions concerning the performance of some action, such as "What would happen to average test scores if students were assigned a weekly project?" Answering such questions requires understanding the effect of an intervention.
- **Level 3: Counterfactuals.** The most advanced level of causal reasoning deals with imagining alternate worlds with questions such as "Would this student have performed better if they had eaten lunch before the exam?" Counterfactual reasoning allows us to evaluate hypothetical scenarios.

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. To support such reasoning, we turn to causal models.

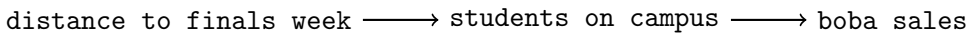
A causal model is a framework to represent and reason about causal relationships between variables. These models are commonly represented by directed graphs, where each variable is represented by a node, and a causal relationship between two variables is represented by a directed edge between the two corresponding nodes in the direction of causal influence. Throughout this thesis, we assume all causal models are acyclic, and we thus can model them with directed acyclic graphs (DAGs).

In addition to being simple and intuitive, causal models provide insights into how changes in one variable propagate to others, allowing rigorous reasoning about relationships between variables, even in the presence of confounding factors. A simple example is shown in Figure 1.1.1b.

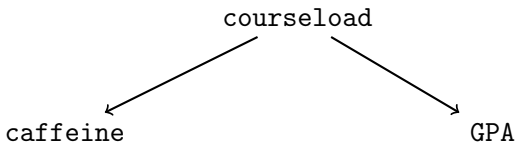
It is useful to consider causal *paths* because they represent possible routes through which influence can flow between variables in a causal model. These paths are undirected in the sense that they may contain edges pointing both forward and backward, allowing us to examine dependencies that emerge from various structural configurations. We define three central structures that appear in such paths:

Definition 1.1.1. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a causal model, where $a, b, c \in \mathcal{V}$.

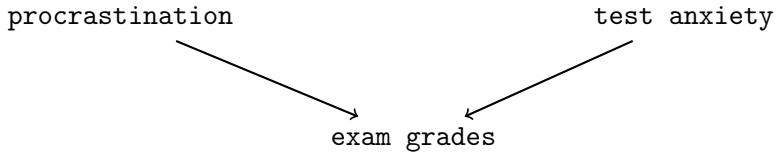
- b is a **mediator** of a and c if $(a, b) \in \mathcal{E}$ and $(b, c) \in \mathcal{E}$ or if $(b, a) \in \mathcal{E}$ and $(c, b) \in \mathcal{E}$.
- b is a **confounder** of a and c if $(b, a) \in \mathcal{E}$ and $(b, c) \in \mathcal{E}$.



(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.

Figure 1.1.1: The subfigures illustrate examples of mediators, confounders, and colliders in academic settings. These simple causal models help demonstrate how different structures can influence observed relationships among variables.

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

Examples of all three of these structures are shown in Figures 1.1.1a, 1.1.1b, and 1.1.1c. Understanding how to adjust for these structures is central to causal inference. For instance, adjusting for a confounder is necessary, but adjusting for a collider introduces bias.

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 the variables of Z will ensure that a has no effect on b, and vice versa.

This concept can be formalized using probabilities.

Definition 1.1.2. 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

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

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 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 notion.

Definition 1.1.3. 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 at least one descendant in Z (here, we consider a node to be 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 1.1.1b, `caffeine` and `GPA` are d-separated given $Z = \{\texttt{courseload}\}$ but not $Z = \emptyset$.

In *Causality* [3], Pearl connects the notions of conditional independence and d-separation, 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. The probabilistic definition treats conditional independence as a black-box property of distributions, and d-separation provides a syntactic rule without grounding it in the actual 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.

1.2 Relevant Work

This thesis draws inspiration from several strands of research in causal inference, spanning both theoretical frameworks and practical tools. We discuss four major areas of influence, which differ from our approach in purpose and scope but provide both motivation and context for the formalization.

We conclude with a fifth area of influence outside the causal-inference literature: the formal verification of the security of hardware systems, which adopts a graph-based perspective on world semantics and information flow that closely parallels our approach to modeling causal systems.

1.2.1 Modeling Counterfactuals

In the Ladder of Causation [2], counterfactuals are on the highest rung, representing the most advanced reasoning about hypothetical scenarios that we cannot directly observe. Modeling these counterfactual scenarios is crucial in experimentation to assess potential interventions and examine hypothetical alternatives. These ideas are particularly prominent when applied to domains such as medicine, where choices regarding interventions could have profound consequences.

Causal models are effective at illustrating the relationships between variables, but their generic form lacks the expressiveness needed to capture counterfactuals. We describe two models, which aim to provide not only the visual intuition of causal models but also explicit mechanisms for representing hypothetical scenarios.

The *twin-network model*, introduced by Balke and Pearl [4], extends the graph-based framework by constructing two parallel versions of the same causal model: one represents the observed world, and the other represents the counterfactual world. These "twin" graphs share the same structure; each variable in the observed world has a corresponding variable in the counterfactual world, and these two variables share an unobserved confounder accounting for experimental error. Depending on the exact query of interest, specific variables can

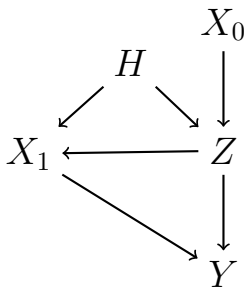


Figure 1.2.1: This causal model [5] represents the relationships among anti-retroviral treatment at time 0 (X_0), HIV viral load after time 0 (Z), anti-retroviral treatment at time 1 (X_1), and CD4 count after time 1 (Y). Here, H is an unmeasured common cause between Z and X_1.

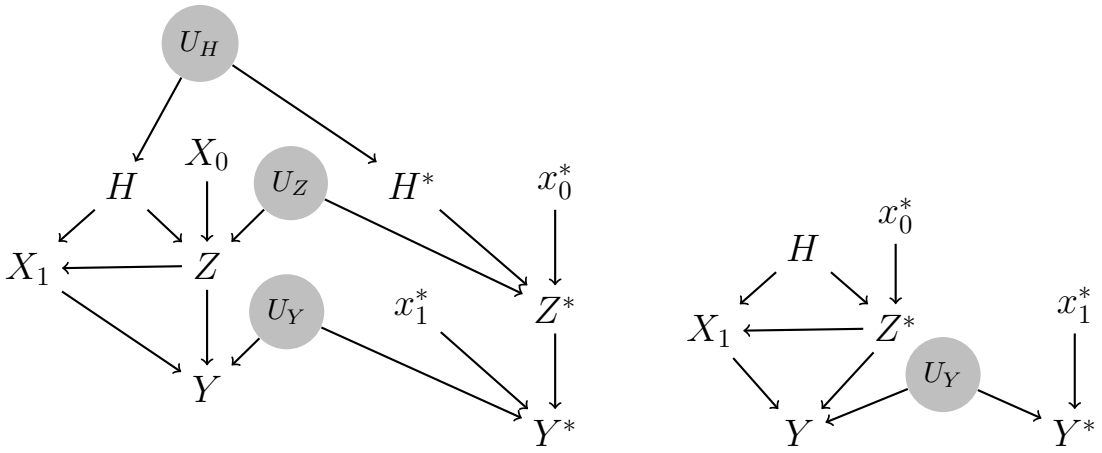
undergo interventions in either world; for example, asking the question "Would providing only one treatment have an effect on the recovery results of the patient, who actually received two treatments?" would be modeled by intervening on the variables representing the two treatments in the counterfactual world. More concretely, consider the causal model of Figure 1.2.1, which models *sequential randomness*. The posed question considers the counterfactual assignments of X_0 and X_1. The corresponding twin network is shown in Figure 1.2.2a.

However, the original twin-network model faced limitations because it did not fully address deterministic relationships between some pairs of observed and counterfactual nodes. In its original form, the model can overlook situations where the counterfactual outcome might be entirely predictable based on the observed data. This oversight resulted in an error in Example 11.3.3 of *Causality* [3], where two nodes, X_1 and Y^* in Figure 1.2.2a, were incorrectly determined to be independent conditioned on Z and $X_0 = x_0^*$.

To address this error, a "preprocessing step" was introduced to merge certain pairs of nodes where such deterministic relationships exist [6]. However, the algorithm for the preprocessing step is difficult to apply and thus not commonly used. A recent development in this space is the introduction of shadow graphs [7], which offer an alternative visual and conceptual organization of the twin network. While the underlying structure remains the same with identical nodes and edges, shadow graphs arrange the counterfactual nodes directly beneath their corresponding factual counterparts. This layout makes dependencies and relationships easier to trace and interpret, preserving the twin network's expressive power while enhancing readability, making it a useful stepping stone toward more formal semantic definitions of counterfactuals.

An additional model for counterfactual reasoning is the *single world intervention graph* (SWIG) [8]. Instead of creating two separate networks, a SWIG "splits" nodes that are subject to interventions, which creates a modified graph where counterfactual variables and observed variables exist within a single model.

The SWIG was introduced with the benefit of overcoming errors such as the one described above. Due to the method of splitting a node into an observed portion and a counterfactual portion, SWIGs do not face the same dependency issues as the twin network. However, the



(a) Prior to the preprocessing step

(b) After the preprocessing step

Figure 1.2.2: Using the causal model of Figure 1.2.1, we generate its twin-network model [4] on the left, with counterfactual nodes represented with asterisks and counterfactual assignments $\{X_0 = x_0^*, X_1 = x_1^*\}$ applied in the graph. The twin network after undergoing the preprocessing step [6] is on the right.

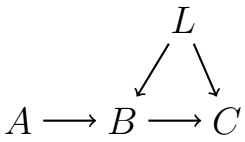


Figure 1.2.3: This causal model has a common structure, where A is termed an *instrumental variable*. It was used by John Snow to determine the source of cholera in 1853 [2]. In that diagram, the water company (A) was the instrumental variable, with water purity (B) and cholera (C) confounded by poverty, location, etc. (L).

twin network is often preferable to the SWIG because it adheres more closely to the traditional representation of causal models. Nevertheless, the complications with the preprocessing step suggest that while the twin-network approach offers a necessary framework for visualizing and analyzing counterfactuals, its practical utility would benefit from a formalization that carefully accounts for the nuances of dependencies between observed and counterfactual worlds.

1.2.2 Inducing Paths

In the work by Spirtes [9], inducing paths are introduced as a strategy for making inferences about causal relationships when there may be latent variables that are not included in the causal model. The motivating example is the common causal-model structure shown in Figure 1.2.3.

Definition 1.2.1. In a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let P be an undirected path between a and b, where $a, b \in \mathcal{V}$. Then, P is an **inducing path over O**, where $O \subseteq \mathcal{V}$, if and only if every member of O on P is a collider on P, and every collider on P is an ancestor of either a or b.

Inducing paths are closely related to d-separation in a manner captured in the following theorem:

Theorem 1.2.2. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a causal model, where $O \subseteq \mathcal{V}$ and $a, b \in \mathcal{V}$. Then, a and b are not d-separated by any subset of $O \setminus \{a, b\}$ if and only if there is an inducing path over O between a and b.*

In Figure 1.2.3, let $O = \{A, B, C\}$. Then, $A \to B \leftarrow L \to C$ is an inducing path over O. By Theorem 1.2.2, A and C are not d-separated by any subset of $O \setminus \{A, C\} = \{B\}$. This statement is true, since conditioning on nothing leaves the path $A \to B \to C$ open, while conditioning on B opens the path $A \to B \leftarrow L \to C$. While we can simply confirm these two cases, the result was not immediately obvious, suggesting that for more complicated graphs, the same conclusion would be difficult to draw manually. Theorem 1.2.2 gives us a more straightforward way of reaching the same result, allowing us simply to find an inducing path.

Inducing paths have been important in the analysis of maximal ancestral graphs (MAGs) and partial ancestral graphs (PAGs), which are used in causal inference to model causal structures in the presence of latent variables [10]. Additionally, several open questions about inducing paths remain, particularly concerning their role in notions like d-separation and the distributions associated with causal graphs [9].

Formally proving the relationship stated in Theorem 1.2.2, which establishes the equivalence between d-separation and the existence of inducing paths, would provide a rigorous foundation for this concept. While Spirtes outlines a proof [9], it includes nontrivial logical leaps, such as the assertion that all nodes on an inducing path are either ancestors of one of the endpoints or ancestors of a collider. A formal proof in Coq would eliminate any ambiguity, as well as allow inducing paths to be integrated seamlessly into the broader causal-model framework, enabling further reasoning and development related to these structures.

1.2.3 Tooling for Causal Inference: Causal Fusion

Causal Fusion [11] is a recent tool developed to support interactive reasoning about causal graphs following the methodology in Bareinboim and Pearl [12]. Through a graphical user interface, it enables users to input causal models as DAGs and test for path-based properties such as d-separation, backdoor and frontdoor criteria, and do-calculus derivability. It also performs probabilistic computations tied to the structure of the model.

While Causal Fusion and our work overlap in the types of queries they support, particularly path analysis and do-operator transformations, our formalization diverges in intent. Causal Fusion is designed as a practical, exploratory tool for researchers and practitioners; in contrast, our work seeks to develop a mathematically rigorous foundation for causal semantics, with a larger goal of applying this foundation to formal reasoning tasks such as experimental design,

as discussed in Section 6.5. Ultimately, we hope that the functions and theorems we define can complement tools such as Causal Fusion by providing correctness guarantees.

1.2.4 Probabilistic Results in Causal Inference

While tools like Causal Fusion support both structural and probabilistic inference in practice, foundational questions remain about the semantics underlying such probabilistic reasoning. Another important thread of related work concerns the relationship between causal structure and probabilistic conclusions, particularly the identification and bounding of probabilities of causation.

Tian and Pearl [13] explore how quantities such as the probability of necessity and sufficiency can be computed or bounded using the functional form of a structural causal model and available observational or interventional data. These results are especially relevant to our work because they leverage semantics that align closely with our own, treating causal models as deterministic functions of parent variables and unobserved background factors.

In the future, we aim to extend our formalization to incorporate probabilities in a similarly principled way. Our goal is not only to replicate identification results within Coq but also explore how semantics-grounded reasoning can clarify the assumptions required to justify probabilistic inferences. In doing so, we hope to bridge the gap between graphical and probabilistic reasoning.

1.2.5 Formal Verification and Hardware-Design Security

While the previous sections discussed work within the field of causal inference, our approach is also informed by formal methods, specifically prior work on mechanized reasoning about systems with complex information flow. In particular, our semantic modeling of worlds draws inspiration from work on verified side-channel security for hardware designs [14]. In this system, graphs are used to represent the flow of information between various components of a hardware architecture, especially when some inputs and outputs are designated as public or private. A goal in such settings is to verify *noninterference*, the property that private inputs do not affect public outputs, ensuring that there is no information leakage.

This paradigm, while conceptually distinct from causal inference, shares an important foundation: both frameworks seek to track how information flows through a system and what guarantees can be made about such flows. However, the structure of the world modeled by causal graphs is more complex; unlike the clean dichotomy of inputs and outputs in confidentiality analysis, causal models often contain latent variables and ambiguous structural roles. For example, confounding variables, which may not be directly observable and do not fall neatly into categories like "inputs" or "outputs," can still undermine experimental designs.

This added complexity means we cannot simply assert properties like noninterference by labeling nodes; instead, we must carefully define what it means for one variable to influence another within the semantics of causal models. The need to justify the impact (or lack thereof) of variables like confounders, mediators, and colliders from first principles calls for

a rigorous semantics in order to bring causal inference closer to the same level of formal assurance that confidentiality analysis has achieved in systems verification.

Chapter 2

Formalizing Causal Models in Coq

This chapter presents the foundational work of implementing causal diagrams and reasoning principles within the Coq proof assistant. We begin by implementing DAGs and then build upon this framework to formalize key causal concepts and theorems, combining classic graph theory with domain-specific ideas from causal inference.

2.1 Causal Diagrams as Directed Acyclic Graphs

In order to formalize causal models, we first build the representation of DAGs within Coq. This step involves creating data structures to represent DAGs and implementing various graph-related functions, such as those for pathfinding, cycle detection, and topological sort. Importantly, each of these functions should have its correctness formally proven using Coq, ensuring that it performs as expected under all circumstances.

We choose to represent causal models as directed graphs in terms of `nodes` and `edges`:

```
Definition node: Type := nat.
Definition nodes := list node.

Definition edge: Type := node * node.
Definition edges := list edge.

Definition graph: Type := nodes * edges.
```

Nodes are simply natural numbers. These choices allow us to reuse several `nat`-specific functions and theorems that are available in the Coq standard library, while still representing nodes in an intuitive way.

Paths are also a crucial part of causal models, since d-connectedness requires the existence of a d-connecting path. We represent paths as follows:

```
Definition path: Type := node * node * nodes.
Definition paths := list path.
```

Here, a path is represented as a tuple of its start node, end node, and intermediate nodes. For example, the path $(1, 4, [2; 3])$ would represent the path $1 \to 2 \to 3 \to 4$. This choice of representation allows us to access the endpoints of a path easily as well as require that a path has a start and an end node. Note that by this definition, a path has at least one edge; the path $(1, 1, [])$ would denote the self-loop at node 1, rather than the 1-path consisting only of node 1.

Formalizing basic graph operations in Coq is far from straightforward. Unlike imperative programming environments, Coq's purely functional setting and totality constraints mean that even relatively simple procedures require careful construction. Moreover, Coq demands not only executable functions but also formal proofs of their correctness, each one built from axiomatic foundations. While not all graph-related functions have been fully verified in our formalization, the unproven parts pertain to well-established classical results (e.g., that there exists a function that finds all acyclic, undirected paths between two nodes in a DAG, or that parents precede children in a valid topological sort). Thus, correctness failures, if any, would reflect limitations in the implementation of these individual functions, not in the logical consistency of the overall system. We revisit this distinction in Section 6.1.

We highlight two major categories of graph functionality: pathfinding and topological sorting. Together, they illustrate both the expressive potential and technical difficulty of working with graph structures in a proof assistant like Coq.

2.1.1 Pathfinding

Identifying paths between nodes is a core operation in graph-based causal reasoning. However, implementing even a basic pathfinding function in Coq is difficult due to its functional nature and the need for total, terminating programs and well-founded recursion.

Our pathfinding implementation relies on recursive exploration. Given a graph $\mathcal{G}$ and two nodes u, v, we aim for the function `find_all_paths_from_start_to_end`$(u, v, \mathcal{G})$ to return a list of paths that contains all undirected paths between u and v in $\mathcal{G}$. The function relies on numerous intermediate results.

- `Fixpoint edges_as_paths_from_start (u: node) (E: edges): paths`

  Returns list of paths of length 2 that start from u by transforming edges that are incident to u into paths, i.e., $(u, v) \mapsto (u, v, [])$.

- `Fixpoint extend_paths_from_start_by_edge (e: edge) (l: paths): paths`

  Returns list of paths that contains all paths in `l`, as well as all paths that can be constructed from extending a path in `l` by `e`. For example, $(1, 4, [2; 3])$ can be extended by $(5, 4)$ into $(1, 5, [2; 3; 4])$. Note that these paths are undirected, meaning that edges can be either forward or backward. Furthermore, a path is not added if it would be made cyclic; for example, $(1, 4, [2; 3])$ would not be extended by $(4, 2)$, since node 2 would appear twice in the resulting path.

- `Fixpoint extend_paths_from_start_by_edges (E: edges) (l: paths): paths`

  Returns result of repeatedly calling `extend_paths_from_start_by_edge` for all edges in `E` on the result of the previous call, beginning with `l`.

- `Fixpoint extend_paths_from_start_iter (E: edges) (l: paths) (k: nat): paths`

  Returns result of repeatedly, `k` times, calling `extend_paths_from_start_by_edges` on the result of the previous call, beginning with `l`.

Then, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a start node u, and an end node v, we can find all undirected paths from u to v in $\mathcal{G}$:

```
Definition find_all_paths_from_start_to_end
                              (u v: node) (G: graph): paths :=
  match G with
  | (V, E) => filter (fun p => v =? path_end p)
                (extend_paths_from_start_iter E
                    (edges_as_paths_from_start u E) (length V))
  end.
```

Since we only want acyclic paths that do not have any repeating nodes, it is enough to iterate $|\mathcal{V}|$ times, which ensures that the function terminates. While the resulting function is inefficient compared to typical imperative algorithms, these definitions are intended primarily to be used as part of a formal semantics rather than for execution on large graphs. For the relatively small causal models used in practice, however, this function is sufficient.

Building on the pathfinding routine, we can define a function to detect directed paths, which are essential for identifying descendants and ancestors of nodes, both of which are concepts used extensively in our semantic treatment of causal models. We also define a function to detect cycles by adapting the directed pathfinding function. Since causal graphs are assumed to be acyclic, cycle detection is a necessary safeguard. Many downstream functions, such as topological sort and tests for d-separation, rely on this acyclicity. As such, cycle detection not only confirms the integrity of the graph but also ensures the well-formedness of the causal reasoning built upon it.

2.1.2 Topological Sort

Having established basic graph traversal and cycle detection, we now turn to topological sort. Since causal graphs are directed and acyclic by assumption, they admit a linear ordering of nodes such that for every directed edge (u, v), u appears before v in the ordering. Topological sorting plays a central role in causal inference, providing an ordering of variables consistent with the direction of causality. This ordering is also crucial for defining the semantics of causal models, which rely on evaluating each node's value based on the values of its parents.

In Coq, our function for topological sort must account for the possibility that the inputted graph is *not* acyclic. To this end, the return type is wrapped in an `option`; the function returns `Some ts` if a valid topological sort `ts` exists and `None` otherwise:

```
Definition topological_sort (G: graph): option nodes
```

Our implementation is based on indegree counting. We iterate $|\mathcal{V}|$ times, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the inputted graph. In each step, we find a node with indegree zero (a node with no parents), append it to the resulting list, and remove it and all its incident edges from the graph. The algorithm recurses on the remaining subgraph until either all nodes are processed or no indegree-zero node is found (in which case the graph contains a cycle and the function returns `None`).

Once this function is defined, we prove several important properties of topological sort, most notably the existence guarantee:

Theorem 2.1.1. *For acyclic graphs, the function always returns a valid topological order. More concretely, for an acyclic graph $\mathcal{G}$, there exists some* (`ts: nodes`) *such that* `topological_sort`$(\mathcal{G})$ = `Some ts`.

Proof. We must show that at each iteration of the function, there is always a node with indegree 0. We proceed via contradiction: assume that in acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, no nodes have indegree 0. Then, we can construct a path of length $|\mathcal{V}| + 1$ by repeatedly prepending parents of the first node. Using the pigeonhole principle, we show that this path must contain a repeated node, implying the existence of a cycle and contradicting our assumption that $\mathcal{G}$ is acyclic. Given this result, we conclude that any acyclic graph must have at least one node with indegree 0, enabling the topological sort to proceed.

In the mechanized proof, we perform induction on $|\mathcal{V}|$ and carefully address all auxiliary results, such as showing that an acyclic graph remains acyclic when a node is removed. □

We also formalize and prove several other key properties of the topological sort, such as the length of the resulting sort being exactly $|\mathcal{V}|$, as well as a node u being in the sort if and only if $u \in \mathcal{V}$. Both of these are proven by induction on the size of the graph.

2.2 Formalizing Key Causal Concepts

Now that the basic DAG framework is established, we shift focus to more specific elements of causal models. The next step involves identifying mediators, confounders, and colliders, which are key components in understanding the causal relationships within a graph. We also define d-separation and d-connectedness and formalize concepts such as interventions via the `do` operator and the backdoor criterion, which connects structure to computable probabilistic quantities.

2.2.1 Identifying Mediators, Confounders, and Colliders

We begin by defining functions to extract mediators, confounders, and colliders from a given path. For instance, to identify colliders, we define a function that, given a path (u, v, l) and a graph $\mathcal{G}$, iterates through the node list $[u]{+}{+}l{+}{+}[v]$ and inspects each triplet of consecutive nodes. If the triplet satisfies the criteria for a collider, namely, the two edges are both incoming into the center node, we append the center node to the result. The output is a list of all colliders in the path.

However, the fact that a node is a collider in a path does not directly reveal which neighboring nodes make it a collider. To address this gap, we prove the following theorem:

```
Theorem colliders_vs_edges_in_path: forall (L: nodes) (G: graph) (x: node),
    In x (find_colliders_in_nodes L G)
    <-> exists y z: node, sublist [y; x; z] L = true
            /\ is_edge (y, x) G = true /\ is_edge (z, x) G = true.
```

Here, the input `L` would be the full list of nodes in a path, i.e., $[u]{+}{+}l{+}{+}[v]$. This result allows us to extract the specific neighbors in the path that cause a node to be a collider, bridging the gap between the abstract identification and concrete structure.

We also prove useful structural lemmas, including that the set of mediators, confounders, and colliders is preserved under path reversal and that every internal node of a directed path is a mediator. These are proven by induction on the length of the path.

Importantly, we show that in acyclic graphs and acyclic paths, no node can simultaneously play more than one of these structural roles. The specific case for mediators is shown below:

```
Theorem if_mediator_then_not_confounder_collider_path:
    forall (G: graph) (u: node) (p: path),
        contains_cycle G = false /\ acyclic_path p
            /\ In u (find_mediators_in_path p G)
        -> ~In u (find_confounders_in_path p G)
           /\ ~In u (find_colliders_in_path p G).
```

This theorem is proven by case analysis, showing that the structural criteria for these roles are mutually exclusive in an acyclic setting. These foundational results are critical for building the semantics of causal inference, which rely on these roles to characterize paths.

2.2.2 Determining *d*-Separation and *d*-Connectedness

With the structural elements defined, we implement a function to determine whether a path is blocked by a conditioning set Z. We define three separate functions that test whether a given path is blocked by a mediator, confounder, or collider, and we combine them into a single predicate:

```
Definition path_is_blocked_bool (G: graph) (Z: nodes) (p: path): bool :=
    is_blocked_by_mediator p G Z ||
```

```
    is_blocked_by_confounder p G Z ||
    is_blocked_by_collider p G Z.
```

We can now define the central notion of d-separation:

```
Definition d_separated_bool (u v: node) (G: graph) (Z: nodes): bool :=
    forallb (path_is_blocked_bool G Z)
            (find_all_paths_from_start_to_end u v G).
```

This definition checks that all paths between u and v are blocked by Z, which corresponds to the classic notion of d-separation in causal DAGs.

We also define d-connectedness in terms of path structure as stated in Definition 1.1.3:

```
Definition d_connected (p: path) (G: graph) (Z: nodes): Prop :=
    overlap Z (find_mediators_in_path p G) = false /\
    overlap Z (find_confounders_in_path p G) = false /\
    all_colliders_have_descendants_conditioned_on
        (find_colliders_in_path p G) G Z = true.
```

We prove the equivalence between these two definitions via a theorem that formalizes the relationship between d-separation and d-connectedness:

```
Theorem d_separated_vs_connected: forall (u v: node) (G: graph) (Z: nodes),
    d_separated_bool u v G Z = false <->
    exists (l: nodes), acyclic_path (u, v, l) /\
                       is_path_in_graph (u, v, l) G = true /\
                       d_connected (u, v, l) G Z.
```

This result, which relies on applications of De Morgan's Law and properties of path enumeration, allows us to move fluidly between the two perspectives.

In our formalization, we typically require paths considered in settings related to d-connectedness and d-separation to be *acyclic*. While the original definitions in causal-inference literature do not explicitly require paths to be acyclic, the acyclicity of the underlying graph implicitly suggests that the paths of interest are simple. In practice, cyclic paths introduce ambiguity: a node may appear multiple times and play conflicting roles, such as being both a confounder and a collider along the same path. Such overlap creates logical inconsistencies and undermines the clean structural interpretation that d-separation provides. Requiring acyclic paths also significantly simplifies reasoning in Coq. For example, many key lemmas and inductive proofs assume that a node's role in a path (e.g., as a mediator, confounder, or collider) is unique and unambiguous. While ensuring paths are acyclic adds some technical burden (especially when handling overlapping paths), it ultimately makes the formal system more robust and intuitive.

2.2.3 Interventions and Backdoor Criterion

We define other functions critical to causal inference, such as the `do` operator, which models the effect of externally intervening on a variable. This operation is implemented by modifying the graph to remove incoming edges to the intervened node, simulating a causal override.

Additionally, we implement a predicate for checking whether two nodes (typically the treatment and the outcome) satisfy the backdoor criterion with respect to a set of nodes Z.

Definition 2.2.1. For $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $a, b \in \mathcal{V}$, a set of variables Z satisfies the **backdoor criterion** relative to (a, b) if no node in Z is a descendant of a, and Z blocks every path between a and b that contains an arrow into a.

The backdoor criterion is an important property because if it holds, then the causal effect of b on a can be computed with a simple formula of probabilities.

A noteworthy outcome from the formalization process was the identification of an informal assumption related to the backdoor criterion. It is commonly accepted that $\mathsf{Pa}(a)$, the set of parents of a, always satisfies the backdoor criterion relative to (a, b) [15]. However, attempting to rigorously prove this theorem in Coq uncovered a subtle edge case: if b is actually a parent of a, then the path $b \to a$ is not blocked by $Z = \mathsf{Pa}(a)$, since the path has no mediators, confounders, or colliders. Thus, this claim is not universally true. Upon review, Judea Pearl also acknowledged the ambiguity, which is difficult to conceptualize since the meaning of a blocked 2-path is unintuitive, suggesting the importance of this rigorous formalization.

2.3 Results From Graph Theory and Causal Theory

In this section, we present a collection of formally verified theorems that bridge structural graph theory and causal reasoning. These results arise repeatedly in downstream proofs involving conditional independence and causal semantics. While some of these results may appear intuitive or straightforward, their formal verification in Coq requires careful definitions and extensive auxiliary lemmas.

Often in formalizing causal reasoning, we must construct or merge paths. To ensure correctness, we need criteria for when d-connectedness is preserved under concatenation.

Theorem 2.3.1. *Let two paths (u, m, l_1) and (m, v, l_2) be d-connected, and assume that the two paths do not share any nodes besides m. Then, the concatenated path $(u, v, l_1{+}{+}[m]{+}{+}l_2)$ is d-connected if m satisfies one of the following:*

1. *It is a mediator in the concatenated path and not in Z.*
2. *It is a confounder in the concatenated path and not in Z.*
3. *It is a collider in the concatenated path and has a descendant in Z.*

Proof. Assume, for contradiction, that the concatenated path is not d-connected. Then there exists some node that blocks the path: either a mediator or confounder in Z, or a collider with no descendant in Z. This node must lie in the first path, the second path, or at the midpoint m. In the first two cases, we contradict the d-connectedness of the original paths. In the third case, we contradict the assumption that m satisfies one of the three listed conditions, since by assumption the resulting path is acyclic, so m can only be one of a mediator, confounder, or collider. Hence, the concatenated path must be d-connected. □

When composing paths, we also often want to avoid cycles or redundancies. Thus, identifying the first point of overlap between two node lists is useful.

Lemma 2.3.2. *Given two lists of nodes l_1 and l_2 that share at least one node, there exists a node x such that:*

$$l_1 = l_1' + +[x] + +l_1'' \text{ and } l_2 = l_2' + +[x] + +l_2'',$$

and there is no overlap between l_1' and l_2'.

Proof. We proceed via induction on l_1, identifying the first point where a node in l_1 appears in l_2. Once found, we split each list at that node and verify the disjointness of prefixes. □

We can in fact strengthen this lemma to avoid later collisions:

Theorem 2.3.3. *Given two lists of nodes l_1 and l_2 that share at least one node, there exists a node x such that:*

$$l_1 = l_1' + +[x] + +l_1'' \text{ and } l_2 = l_2' + +[x] + +l_2'',$$

and there is no overlap between l_1 and l_2'.

Proof. We perform induction on l_2, finding the first point of overlap in l_2 with l_1 and splitting both lists at that node. □

These results help us construct acyclic concatenated paths from intersecting segments, crucial for later disjointness reasoning.

In many proofs, we begin with a directed path between two nodes (e.g., from a collider to its descendant) but require that the path be acyclic. Fortunately, we can always find such a path without losing essential structure.

Theorem 2.3.4. *Let $u \neq v$, and suppose there is a directed path from u to v. Then there exists an acyclic directed path from u to v consisting of a sublist of the original nodes.*

Proof. We perform strong induction on the length of the path. Whenever a cycle is detected (i.e., a repeated node), we remove it, maintaining the original start and end nodes. We prove that subpaths of directed paths remain directed and, by construction, acyclic. □

This result ensures that path-based properties involving descendants can always be reasoned about within the class of acyclic paths.

Continuing with the ideas of intersection points and directed paths, it is often useful to construct new paths by following one directed path to a shared node and continuing along the reverse of the second directed path. This node will always be a collider.

Theorem 2.3.5. *Let (u_1, v_1, l_1) and (u_2, v_2, l_2) be two directed paths that intersect. Let x be the intersection point guaranteed by Theorem 2.3.3. Let P be the path from u_1 to u_2 constructed from following the first path from u_1 to x and continuing along the reverse of the second path to u_2. Then, P is acyclic, and x is a collider in P.*

Proof. By Theorem 2.3.4, we can assume the two directed paths are both acyclic. Note that using the notation of Theorem 2.3.3, $P = (u_1, u_2, l_1' +\!+ [x] +\!+ \texttt{rev}(l_2'))$. Since there is no overlap between l_1 and l_2', P is acyclic.

Furthermore, since both original paths are directed, both edges into x in the resulting path are incoming. Thus, we prove that x satisfies the collider definition by performing casework on the structures of l_1' and l_2' and using `colliders_vs_edges_in_path`, as stated in Section 2.2.1. □

Chapter 3

Semantics of Causal Models

Formalizing a system involves both syntactic and semantic components. The syntactic aspect focuses on the structural rules and relationships, such as how nodes and edges in a DAG interact and satisfy certain properties. Semantics, on the other hand, provide meaning to these structures by specifying how values propagate through the graph. In causal modeling, semantics help us understand not only the presence of relationships but also what these relationships imply about the behavior of variables under interventions or counterfactual scenarios. Integrating semantics into the formalization bridges the gap between abstract causal reasoning and a more practical interpretation, enabling deeper insights and intuitive understanding.

3.1 Function-Based Formal Semantics

Causal models tell us the relationships that may exist between nodes. We choose to capture those relationships using `nodefun`s, which assign a value to each node based on the values of its parents and an additional *unobserved term*. 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. However, a `nodefun` is of course free to ignore or allow little influence to the unobserved term.

We represent a causal model with an overarching `graphfun`, which maps each node in the graph to its corresponding `nodefun`:

```
Definition nodefun (X: Type): Type := X * list X -> X.
Definition graphfun {X: Type}: Type := node -> nodefun X.
```

Here, `X` denotes the type of values assigned to the nodes, e.g., `bool`, `nat`, or more complex types like `list nat` or finite enumerations.

To illustrate how a `graphfun` captures meaningful causal relationships, consider a simple example modeling how a student's sleep, study hours, and concentration influence their test

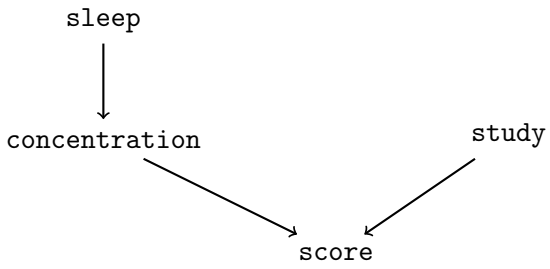


Figure 3.1.1: This causal model encodes the intuition that amount of sleep affects concentration level, and test score is affected by both concentration and study time.

score. Suppose our causal graph is as shown in Figure 3.1.1. Then, we might have the following `graphfun`, where $\texttt{X} = \mathbb{R}$:

```
Definition g_student: graphfun := fun w =>
    match w with
    | sleep => fun (e, _) => e
    | study => fun (e, _) => max 0.0 (e +. 3.0)
    | concentration => fun (e, [sleep]) => (2.0 * sqrt sleep) + e
    | score => fun (e, [concentration; study])
               => (0.6 * concentration) + (0.4 * study) + e
    | _ => fun _ => 0.0
    end.
```

Here, `sleep` and `study` are exogenous, `concentration` depends on `sleep`, and `score` is a weighted sum of `concentration` and `study`. Each variable is also influenced by an unobserved term `e` (e.g., stress, natural ability, distractions).

To track the values assigned to nodes during evaluation, we define the following types:

```
Definition assignment {X: Type}: Type := node * X.
Definition assignments {X: Type}: Type := list assignment.
```

An `assignments X` object functions like a dictionary, mapping nodes to their values. Internally, it is a list of node-value tuples where a lookup returns the most recent match.

In practical applications, a causal model will likely have known values for some of its nodes but not all. To evaluate others, we define the function `find_value`, which computes the value of a node u in a given graph $\mathcal{G}$ with graph function g, using unobserved-term assignments U:

```
Definition find_value {X: Type} (G: graph) (g: graphfun)
                      (u: node) (U: assignments X): option X
```

Since each node's value depends on its parents, and the graph is assumed to be acyclic, a topological sort provides an evaluation order that guarantees termination, as established in Theorem 2.1.1.

The implementation of `find_value` proceeds in stages. The core function is:

```
Definition get_value_of_node {X: Type} (u: node) (G: graph) (g: graphfun)
                             (U A: assignments X): option X
```

The function `get_value_of_node` takes in A, a set of precomputed values. It searches for the parent values of u within A. If any parent is not already assigned, it returns `None`. Otherwise, it evaluates `graphfun` g on u, then evaluates the resulting `nodefun` on the precomputed parent values and the unobserved term assigned to u in U.

We then define `get_values`, which computes the topological sort of the graph and iteratively computes the value of each node in order using `get_value_of_node`. The resulting list of assignments accumulates values as they are computed. After going through the entire topological sort, `get_values` returns the computed values of all nodes as a set of assignments. Finally, the `find_value` function invokes `get_values` and looks up the value for the desired node.

While it is cleanest to refer to the value of a node u with the `find_value` function, i.e., `find_value`$(\mathcal{G}, g, u, U)$, this notation is generally not useful in proofs, since it unfolds to simply `get_assigned_value`(V, u), where V is the result to the call to `get_values`$(\mathcal{G})$. Instead, most proof work must proceed by reasoning directly about `get_value_of_node`. Fortunately, the following theorem shows that the `find_value` and `get_value_of_node` functions are equivalent as long as the parent values are correctly supplied:

Theorem 3.1.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic graph with graph function g, and let U be an assignment of unobserved terms for the nodes in $\mathcal{V}$. For $u \in \mathcal{V}$, if P maps parents p of u to* `find_value`(G, g, p, U)*, then* `find_value`$(G, g, u, U) =$ `get_value_of_node`(u, G, g, U, P).

Proof. While this theorem is quite intuitive, it is difficult in a mechanized proof due to the repeated calls and changing arguments to `get_value_of_node` in `get_values`. We cannot simply perform induction on the topological sort of $\mathcal{G}$ because after removing the first node to get a shorter topological sort, any child of that node may now have a different `nodefun` or value.

Instead, we divide the proof into two primary lemmas. Let V be the result of the `get_values` function, which returns the values assignments for the nodes in $\mathcal{V}$ in topological order. Suppose u is the i-th node of the topological sort of $\mathcal{G}$. Let V_i be the first i elements of V.

1. `get_value_of_node`$(u, G, g, U, V_i) =$ `get_value_of_node`(u, G, g, U, P).
2. `find_value`$(G, g, u, U) =$ `get_value_of_node`(u, G, g, U, V_i).

The first statement is a result of using induction on the parents of u, as well as several lemmas about the topological sort of $\mathcal{G}$. The second statement is a result of using induction on i to show that the `get_values` function preserves index; in other words, the assignment for u in V must be at index i. The result then follows. □

With this foundation established, we now shift focus away from implementation-specific details and toward how these semantics can be used to define and reason about conditional

independence. For the remainder of this thesis, we adopt more abstract notation and move toward a semantics-driven formalism.

In particular, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG. For any U of type `assignments X`, such as the unobserved-terms assignments for nodes of $\mathcal{V}$, we can think of U as a function $U : \mathcal{V} \to \mathtt{X}$. We write $U(w)$ for the value that w is assigned within U, for all $w \in \mathcal{V}$.

Furthermore, each `nodefun` can be computed with knowledge of the unobserved-terms assignments for $\mathcal{V}$ and the structure of $\mathcal{G}$, since it can then determine the unobserved term and parents of the desired node. Thus, for simplicity, let a function f encode the information about $\mathcal{G}$'s `graphfun` and each `nodefun`, and let f_U denote the function $f_U : \mathcal{V} \to \mathtt{X}$, which takes in a node, finds its `nodefun`, and computes its value using U as the unobserved-terms assignments for $\mathcal{V}$. The function f_U thus captures the entire semantics of the causal model under given assignments of unobserved terms.

3.2 Defining Conditional Independence

The semantic framework enables us to define 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, which depends on the values of its parents and an unobserved term. Therefore, 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 3.2.1. We say that a world modeled by unobserved-terms assignments U **properly conditions on $\boldsymbol{Z}$** if, for the given assignments A_Z, $f_U(z) = A_Z(z)$ for all $z \in Z$.

In particular, if u and v are conditionally independent, then in a world where $f(u) = \alpha$ and all nodes in Z are properly conditioned, if we were to modify the world 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 of the world can be absorbed into the unobserved-terms assignments, U. With this setup, we attempt our first formal semantic definition of conditional independence.

Definition Attempt 3.2.2. Let $\mathcal{G}$ be a graph with graph function f, and let u and v be nodes in $\mathcal{G}$. We say u and v are **conditionally independent given $\boldsymbol{Z}$** if given values α, β and conditioned assignments A_Z, then for all unobserved assignments U_α and U_β such that

1. $f_{U_\alpha}(u) = \alpha$, and f_{U_α} properly conditions on Z.
2. $f_{U_\beta}(u) = \beta$, and f_{U_β} properly conditions on Z.

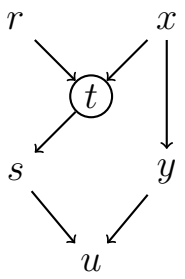


Figure 3.2.1: In the above, $Z = \{t\}$, and $\mathsf{Anc}^*_Z(u) = \{u, s, y, x\}$. Note that although x has a blocked path to u through t, we only require the existence of any unblocked path.

the value of v stays constant, i.e., $f_{U_\alpha}(v) = f_{U_\beta}(v)$.

While this definition captures the idea that changing u's value should not affect v if they are conditionally independent, it fails to isolate *why* $f(v)$ changed. Specifically, the change in $f(v)$ might be attributable not to $f(u)$ but to other aspects of the changing unobserved-terms assignments. For example, if u and v are disconnected nodes (and thus should be conditionally independent for any Z), we could still have $U_\alpha(v) \neq U_\beta(v)$, causing a difference in $f(v)$ that is entirely unrelated to u.

Attempt 3.2.2 is therefore too permissive, making it impossible to attribute observed differences in $f(v)$ specifically to the change in $f(u)$. To address this issue, we need to restrict how U_α and U_β are allowed to differ. The goal is to make a change minimally in a way that would 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 a conditioned node. We thus define the following notion.

Definition 3.2.3. 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 $\mathsf{Anc}^*_Z(u)$.

An example is shown in Figure 3.2.1.

This restriction helps us ensure that when we compare two different worlds, represented by different unobserved-terms assignments, those differences are localized to the nodes that could actually affect u. In doing so, we rule out irrelevant sources of variation.

Definition Attempt 3.2.4. Let $\mathcal{G}$ be a graph with graph function f, and let u and v be nodes in $\mathcal{G}$. We say u and v are **conditionally independent given Z** if given values α, β and conditioned assignments A_Z, then for all unobserved assignments U_α and U_β such that

1. $f_{U_\alpha}(u) = \alpha$, and f_{U_α} properly conditions on Z.

2. $f_{U_\beta}(u) = \beta$, and f_{U_β} properly conditions on Z.

3. U_β differs from U_α for only $a \in \mathsf{Anc}_Z^*(u)$.

the value of v stays constant, i.e., $f_{U_\alpha}(v) = f_{U_\beta}(v)$.

A third condition has been added from Attempt 3.2.2, where we now only allow changes to unblocked ancestors of u in order to affect $f(u)$. This definition improves on Attempt 3.2.2 by ensuring that any change in the unobserved assignment from U_α to U_β must be localized to nodes that can causally influence u.

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. For example, suppose we have a simple collider structure $u \to c \leftarrow v$, and $Z = \{c\}$. Note that $u, v \in \mathsf{Pa}(c)$. Thus, suppose we define $f(c) := f(u) \oplus f(v)$. Then, we would expect for u and v to be *dependent* conditioned on Z. For example, if $A_Z(c) = 1$, and we change the value of u from 0 to 1, then the value of v would have to change from 1 to 0. However, Attempt 3.2.4 does not allow us to observe this dependence, since the effect of u on v in this case is indirect through conditioned node c, whose value must be held fixed.

This example highlights a key insight: determining conditional independence 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. We refer to this second stage as *"reparative propagation."*

Definition Attempt 3.2.5. Let $\mathcal{G}$ be a graph with function f, and let u and v be nodes in $\mathcal{G}$. We say u and v are **conditionally independent given $\boldsymbol{Z}$** if given values α, β and conditioned assignments A_Z, then for all unobserved assignments U_α, U_β, and U'_β such that

1. $f_{U_\alpha}(u) = \alpha$, and f_{U_α} properly conditions on Z.

2. $f_{U_\beta}(u) = \beta$, and U_β differs from U_α for only $a \in \mathsf{Anc}_Z^*(u)$.

3. $f_{U'_\beta}(u) = \beta$, $f_{U'_\beta}$ properly conditions on Z, and U'_β differs from U_β for only

$$a \in \bigcup_{\substack{z \in Z \\ f_{U_\beta}(z) \neq A_Z(z)}} \mathsf{Anc}_Z^*(z).$$

the value of v stays constant, i.e., $f_{U_\alpha}(v) = f_{U'_\beta}(v)$.

This attempt brings us closer to a robust definition by allowing a two-stage process: first, we change u's value via its unblocked ancestors (step 2), then we adjust any conditioned nodes that are no longer correctly fixed (step 3).

However, Attempt 3.2.5 fails to account for two cases that may arise due to indirect effects among nodes in Z. First, consider the case of two consecutive nodes in Z, such as in the path

$u \to z_1 \to z_2 \leftarrow a$, where $Z = \{z_1, z_2\}$. It is clear that conditioning on z_2 is redundant, since conditioned on z_1, z_2's value is fixed. If changing $f(u)$ affects z_1, then it will also affect z_2. However, restoring z_1 to its correct value should automatically fix z_2. Attempt 3.2.5 allows for an alternate repair of z_2 via a, introducing potential noncausal paths of influence.

Second, consider a sequential case of conditioned nodes, such as in the path $u \to z_1 \leftarrow x \to z_2 \leftarrow y$, where $Z = \{z_1, z_2\}$. Repairing z_1 in the propagation step via x may now cause changes to z_2. Note that the repair of z_2 is disallowed in step 3 because z_2 was initially correct under U_β. However, after z_2 is perturbed, U'_β would no longer properly condition on Z. Thus, the propagation step in Attempt 3.2.5 does not allow for full repair of the graph.

To handle these cases, we generalize the reparative-propagation assignments U'_β into a *sequence* of assignments of unobserved terms, where each incrementally restores the conditioning set using only changes justified by the previous set of assignments.

Definition 3.2.6. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with graph function f, and let $u, v \in \mathcal{V}$. We say u and v are **conditionally independent given Z** if given values α, β and conditioned assignments A_Z, then for all unobserved-terms assignments U_α, $U_\beta =: U_0$, and $U_1, ..., U_\ell$ such that

1. $f_{U_\alpha}(u) = \alpha$, and f_{U_α} properly conditions on Z.
2. $f_{U_\beta}(u) = \beta$, and U_β differs from U_α for only $a \in \mathsf{Anc}^*_Z(u)$.
3. For all $i = 1, ..., \ell$, U_i differs from U_{i-1} for only

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

   For $i = 1$, we let $U_{i-2} = U_\alpha$.
4. $0 \leq \ell \leq |\mathcal{V}|$.
5. $f_{U_\ell}(u) = \beta$, and f_{U_ℓ} properly conditions on Z (note if $\ell = 0$, then this condition applies to $U_0 = U_\beta$).

the value of v stays constant, i.e., $f_{U_\alpha}(v) = f_{U_\ell}(v)$.

In this formulation, the unobserved-terms assignments contained in U_β trigger a change to $f(u)$, and the subsequent (possibly empty) sequence of unobserved-terms assignments repairs any disrupted values in the conditioning set Z. In particular, each U_i is responsible for repairing changes to conditioned nodes that were affected by U_{i-1}.

While this definition introduces some complexity, it reflects the full structure of the causal model and guarantees that conditioned nodes remain truly fixed. Since at least one conditioned node must be repaired for each i, we can bound the length of the sequence by the number of conditioned nodes, which is of course upper-bounded by the total number of nodes

$|\mathcal{V}|$. It is worth considering whether this sequence can be *cyclic*, in that a repaired node is unconditioned again in a later set of assignments and must be repaired again. While cyclic sequences are not forbidden by Definition 3.2.6, the definition must hold for all sequences satisfying the criteria, so redundant or oscillating repair paths can be ignored. It is in fact possible to derive a tighter bound on ℓ, which we explore in Section 4.5.

Ultimately, Definition 3.2.6 encapsulates the three phases of the process laid out in the Definition Attempts: the initialization in step 1, the catalyst update in step 2, and the reparative propagation in step 3, ending with a properly conditioned set of unobserved-terms assignments that still has u evaluating to β.

3.3 Conditional Independence $\iff$ *d*-Separation

Using the notion of conditional independence established in Definition 3.2.6, we now present the central result of this thesis:

Theorem 3.3.1. *The notions of conditional independence 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 conditionally independent given Z if and only if they are d-separated given Z in $\mathcal{G}$.*

In the Coq formalization, Theorem 3.3.1 is stated generically over all types `X` for which node values may be drawn. To ensure soundness of the semantics, we assume the following constraints on `X`:

- `X` contains at least two distinct elements.
- `X` is equipped with a Boolean equality function `eqb` satisfying reflexivity and symmetry.

Additionally, $\mathcal{G}$ must satisfy the following well-formedness conditions:

- Each node in $\mathcal{V}$ and each edge in $\mathcal{E}$ appears at most once.
- All nodes referenced in any edge must appear in $\mathcal{V}$.
- The conditioning set Z (whose type is `nodes` in the formalization) contains no duplicates.

We prove Theorem 3.3.1 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 conditional independence. We prove the backward direction by showing that if u and v are not semantically conditionally independent, i.e., there exist unobserved assignments satisfying the conditions of Definition 3.2.6 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 detail in Chapters 4 and 5, respectively. Assuming the equivalence is true, we next examine situations in which the semantic definition offers benefits over the purely structural perspective offered by d-separation.

3.4 Advantages of Semantic Conditional Independence

The semantic definition of conditional independence provides several advantages over the traditional graphical approach of d-separation. While d-separation is a useful and well-established syntactic criterion, its rules can often appear opaque, and it can be difficult to gain intuition about *why* a pair of nodes are or are not conditionally independent. Our semantics-based definition, by contrast, directly captures the idea of how changes propagate through a causal model and makes explicit the mechanisms by which one variable may influence another. By relying on this higher-level understanding, we also hope to avoid the need to analyze very technical aspects of graphs, such as mediators, colliders, and confounders, in exhaustive detail for every proof.

As a toy example, consider the case where we wish to determine whether two nodes u and v are conditionally independent given an empty conditioning set $Z = \emptyset$. From the d-separation perspective, we must verify that every undirected path from u to v is blocked by the empty set. Since there are no nodes in Z, no noncollider on a path can satisfy the blocking condition, while any collider satisfies the criterion. Thus, u and v are independent if and only if *every path from u to v contains a collider*. While correct, this result is not especially intuitive and requires manipulating a set of graph rules to arrive at a conclusion about independence.

From the perspective of our semantic definition, the conclusion is both simpler and more intuitive. Since no nodes are conditioned, no reparative propagation steps will occur in a sequence of unobserved-terms assignments. Therefore, in order for a change in the value of u to result in a change in v, the change must occur directly between U_α and U_β without any further propagation. In this case, such a change is only possible if u and v share an unblocked ancestor. So u and v are conditionally independent if and only if the sets of their unblocked ancestors are disjoint. This condition is more intuitive because it aligns with the idea that if two nodes do not share a source of variation, then they should not influence one another.

Another key advantage of the semantic definition is that it incorporates not just the structure of the graph but also more specific relationships between its nodes via its `graphfun`. While the definition of conditional independence must hold across all possible graph functions, in practice, we may have prior knowledge about how certain nodes behave. For example, suppose a node depends deterministically on just one of its parents or ignores its unobserved term. In such cases, we can rule out a large space of possible graph functions from consideration, making the semantic definition strictly stronger than d-separation: while d-separation must assume the worst-case influence of all parents and confounding paths, the semantic perspective can exploit additional structure or constraints in the system.

The most important advantage is how the semantic definition enhances our intuition for causal reasoning. While d-separation can tell us that two nodes are independent, it does little to tell us why. By contrast, the semantic definition makes it possible to reason explicitly about which changes propagate, which ones do not, and why certain conditional relationships hold in terms of actual values and dependencies. For example, when debugging a causal model or trying to explain the effect of an intervention, the semantic approach gives a clearer

picture of what must remain fixed to preserve certain outcomes, or how an intervention at one node might ripple through the system. This insight is especially valuable in complex settings such as counterfactual analysis or experimental design, where understanding why a conclusion holds is often as important as the conclusion itself.

Although the semantic definition is more expressive, d-separation remains computationally simple to check using tools such as our formalization or existing systems like Causal Fusion [11]. In practice, the two can be used together, d-separation for fast initial screening of independence relationships and the semantic definition for the understanding of these relationships in richer detail.

Chapter 4

Forward Direction: Conditional Independence Implies d-Separation

We now prove the forward direction of Theorem 3.3.1, restated below for clarity.

Lemma 4.1. For a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two different nodes $u, v \in \mathcal{V}$, and a conditioning set $Z \subseteq \mathcal{V}$ with $u, v \notin Z$, if u and v are conditionally independent given Z, then they are d-separated given Z in $\mathcal{G}$.

We proceed by proving the 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, under this condition, u and v are not conditionally independent given Z. Concretely, we will construct a graph function f and unobserved-terms assignments $U_\alpha, U_\beta, U_1, ..., U_\ell$ that satisfy the initialization, catalyst, and reparative-propagation steps of Definition 3.2.6 but result in different values for $f_{U_\alpha}(v)$ and $f_{U_\ell}(v)$, thus violating conditional independence.

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, effectively propagating the value from u to v. 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 conditionally independent and complete the contrapositive proof.

4.1 Constructing a Function to Equate Node Values

We begin by constructing a function f that equates the values of all noncollider nodes along a d-connected path. We examine simple cases before generalizing to an arbitrary path.

4.1.1 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 3.2.6. 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 again ensure all nodes evaluate to the same value.

4.1.2 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 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.

In a more general case, 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 4.1.1, 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.

4.1.3 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, so any effect of the value of u on the value of v must occur through c, their shared child node.

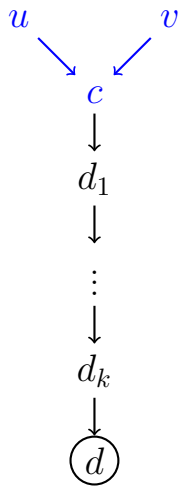


Figure 4.1.1: Assuming the path highlighted in blue is d-connected, then c must have a descendant in Z. Assuming $c \notin Z$, it must have a descendant $d \in Z$ and a descendant path to d.

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

$$f(c) := \begin{cases} x & f(u) = f(v) \\ y & \text{else.} \end{cases}$$

Then, 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 4.1.1. Using the mediator-chain strategy of Section 4.1.1, 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 $f(d) = f(c)$. 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 $f(u) = f(v)$.

4.1.4 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 any setting of unobserved terms that properly conditions on Z.

Specifically, we partition the nodes in $\mathcal{V}$ into six sets:

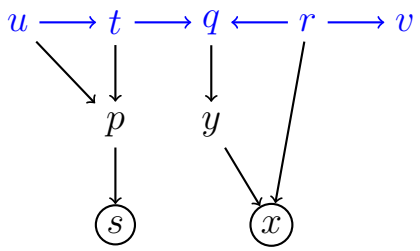


Figure 4.1.2: In the above graph, consider the d-connected path from u to v highlighted in blue, where $Z = \{s, x\}$. Then, the partition is as follows: $S_1 = \{u, r\}$, $S_2 = \{t, v\}$, $S_3 = \{q\}$, $S_4 = \{y, x\}$, $S_5 = \{s\}$, $S_6 = \{p\}$.

- S_1 (sources): nodes on P whose neighbors on the path are not parents
- S_2 (transmitters): every node on P with exactly one neighboring parent on the path
- S_3 (colliders): every node on P whose two neighboring nodes on the path are both parents
- S_4 (descendants): nodes along descendant paths from colliders in P
- S_5 (Z-residual): conditioned nodes not included in $S_1 \cup S_2 \cup S_3 \cup S_4$
- S_6 (residual): all remaining nodes.

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. An example is shown in Figure 4.1.2.

For each collider c, note that $c \in S_3$. If $c \notin Z$, then 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.

In order to construct f^{path} to equate all values of noncollider nodes on P, we will define a specific `nodefun` for each node based on the set into which it falls.

An issue arises upon introducing the set S_4: 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 `nodefun`. However, nodes in S_4 may also be on P or appear on multiple descendant paths. This complication prevents the clean assignment of `nodefun`s. We address this problem in Section 4.2.

Momentarily assuming that all descendant paths are disjoint and do not intersect P, the sets defined above indeed form a disjoint partition on $\mathcal{V}$. We can then define a `nodefun` for each set:

- S_1 (sources): use unobserved term directly.

```
Definition f_unobs (X: Type) (val: X * list X): X := fst val.
```

- S_2 (transmitters): use value of parent on P, where i is the index of the designated parent node in the transmitter's parent list.

```
Definition f_parent_i (X: Type) (i: nat) (val: X * list X): X :=
    nth_default (fst val) (snd val) i.
```

- S_3 (colliders): enforce equality of two parents on P, where $x = A_Z(d)$, d is the descendant of the collider in Z (possibly the collider itself), $y \neq x$, and i and j are the indices of the two parents on P in the collider's parent list.

```
Definition f_equate_ij (X: Type) `{EqType X} (i j: nat) (x y: X)
                       (val: X * (list X)): X :=
    if eqb (nth_default (fst val) (snd val) i)
           (nth_default (fst val) (snd val) j) then x else y.
```

- S_4 (descendants): use `f_parent_i X i` as in S_2, where the parent in the descendant path is at index i in the descendant's parent list.

- S_5 (Z-residual): force to their assigned values in A_Z.

```
Definition f_constant (X: Type) (res: X) (val: X * (list X)): X := res.
```

- S_6 (residual): assign an arbitrary `nodefun`, since these nodes play no role.

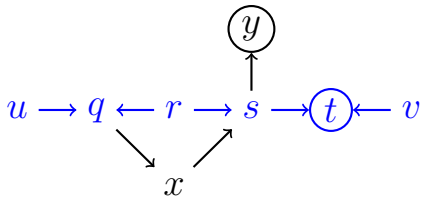
For example, using the graph in Figure 4.1.2, we would assign

$$
\begin{aligned}
f^{\mathsf{path}}(t) &:= f^{\mathsf{path}}(u) \\
f^{\mathsf{path}}(v) &:= f^{\mathsf{path}}(r) \\
f^{\mathsf{path}}(x) &:= f^{\mathsf{path}}(y) \\
f^{\mathsf{path}}(y) &:= f^{\mathsf{path}}(q) \\
f^{\mathsf{path}}(q) &:= \begin{cases} A_Z(x) & f(t) = f(r) \\ \rho & \text{else,} \end{cases}
\end{aligned}
$$

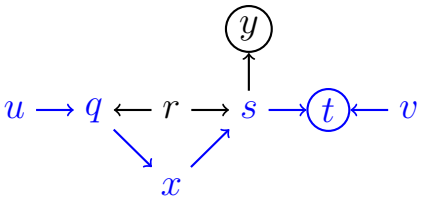
for some $\rho \neq A_Z(x)$. We see that if f^{path} properly conditions on Z, then all noncollider values along the path are equal.

We will formally show in Section 4.4 that f^{path} equates all noncollider values under appropriate unobserved-terms assignments. 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. S_3 nodes enforce equal values on their two neighboring nodes since they must pass down their values to their conditioned descendants.

This structure will ensure the propagation of value from u to v in any d-connected path, allowing us to construct a sequence of assignments that violates conditional independence.

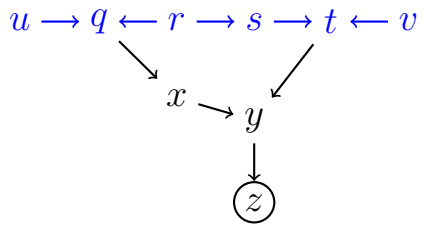


(a) The path highlighted in blue is a d-connected path, but the descendant path of collider q overlaps with the path at s.

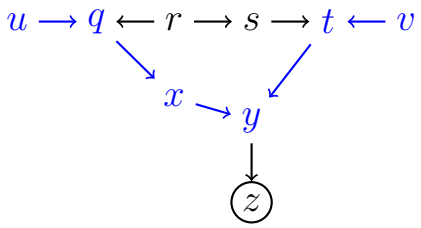


(b) The alternate path in blue is still d-connected, since q is now a mediator. This path is a clean d-connected path.

Figure 4.2.1: Given a d-connected path from u to v where a descendant path of a collider intersects the path itself, we can construct an alternate d-connected path that satisfies Definition 4.2.1.



(a) The path highlighted in blue is a d-connected path, but the two descendant paths of colliders q and t overlap each other at y.



(b) The alternate path in blue is a clean d-connected path, since q and t are now mediators, and y has a descendant path to z.

Figure 4.2.2: Given a d-connected path from u to v where the descendant paths of two colliders intersects each other, we can construct an alternate d-connected path that satisfies Definition 4.2.1.

4.2 Finding Disjoint Descendant Paths

In the previous section, we defined S_4, the set of nodes that appear on the descendant path from any collider in the d-connecting path from u to v. However, while d-connectedness guarantees the existence of such descendant paths, it does not prevent problematic *overlaps*, such as:

- A collider's descendant path might intersect the path itself (see Figure 4.2.1a).
- Descendant paths from two different colliders could intersect (see Figure 4.2.2a).

These situations pose issues for our construction of f^{path}, which requires handling each node in a descendant path with a specific function corresponding to its predecessor in the descendant path. This logic breaks if a node is shared between multiple descendant paths (with multiple predecessors) or if the node is simultaneously a path node (e.g., an S_2 node). For instance, in Figure 4.2.1a, node s would be assigned conflicting roles.

To address this issue, we define a stricter notion of a d-connected path:

Definition 4.2.1. Let $Z \subseteq \mathcal{V}$, and let P be a path. Let $c_1, ..., c_k$ be the colliders in P. We say P is a **clean *d*-connected path** if P is d-connected given Z, and for each c_i, either $c_i \in Z$, or $c_i \notin Z$ and there exists a path $Q_i := (c_i, d_n, [d_1, ..., d_{n-1}])$, where $n \geq 1$, satisfying the following conditions:

1. $d_n \in Z$.
2. $d_i \notin Z$ for $i < n$.
3. $d_i \notin P$ for $i = 1, ..., n$.
4. For all $c_j \neq c_i$ such that $c_j \notin Z$, Q_i does not intersect Q_j.

Our formulation of f^{path} requires a clean d-connected path. Fortunately, when descendant paths intersect either the path or each other, we can construct an alternative clean path using the strategies shown in Figures 4.2.1b and 4.2.2b. These ideas motivate the following result.

Theorem 4.2.2. *If u and v are d-connected, then there exists a clean d-connected path from u to v.*

The proof involves extensive case analysis on the structure of the path and descendant relationships. Since the proof is already fully mechanized in Coq, we present here only a high-level sketch to convey the main ideas.

We introduce the following helper function, which determines the direction of a path immediately after a specific node:

Definition 4.2.3. For an acyclic path $P = (u, v, l)$ and a node w, let $\mathsf{dir}_P(w) \in \{\leftarrow, \rightarrow, \perp\}$ represent the direction of the path immediately after the node w in P. Concretely, if $w \notin P$ or $w = v$, then $\mathsf{dir}_P(w) = \perp$. Otherwise, let w' be the node immediately following w on P. Then,

$$\mathsf{dir}_P(w) := \begin{cases} \leftarrow & (w', w) \in \mathcal{E} \\ \rightarrow & (w, w') \in \mathcal{E}. \end{cases}$$

Note that since we restrict attention to acyclic paths within acyclic graphs, there is no ambiguity in the dir function, since each node appears at most once in the path, and for any adjacent pair of nodes w, w', only one of (w', w) or (w, w') can be an edge in the graph. This notion plays a crucial role in the proof; for example, for a directed path P, every intermediate node w satisfies $\mathsf{dir}_P(w) = \rightarrow$. More generally, in any acyclic path P, if $\mathsf{dir}_P(w) = \rightarrow$, then w cannot be a collider on P.

To prove Theorem 4.2.2, we proceed by induction. At each step, we require the path to satisfy specific structural conditions in order to construct the induction step from the hypothesis. Thus, we establish a stronger, more precisely formulated version of Theorem 4.2.2 that incorporates these additional constraints.

Lemma 4.2.4. *Let $P = (u, v, l)$ be a d-connected path. Then there exists a clean d-connected path $P' = (u, v, l')$ such that for any node w satisfying either of:*

I. $w \in P'$ *and* $w \notin P$.

II. $\mathsf{dir}_P(w) \neq \mathsf{dir}_{P'}(w)$.

the following conditions hold:

1. *One of:*

   (a) $w \notin Z$, *and there exists a directed path* (w, d, p) *with* $d \in Z$.

   (b) $w \in Z$, *and* $\mathsf{dir}_{P'}(w) \in \{\leftarrow, \perp\}$.

2. *If* $\mathsf{dir}_{P'}(w) = \rightarrow$, *then* $w \notin Z$.

Proof sketch. We perform induction on the length of P. In particular, we identify the node immediately after u in P and invoke the induction hypothesis to construct a clean d-connected path from that node to v. We then prepend u and analyze the overlaps that are introduced:

- Node u could overlap the induction path or its descendant paths.
- If the addition of u makes the next node a collider, then its new descendant path could intersect the induction path or any of its descendant paths.

For each of these intersections, we construct alternate paths as in Figures 4.2.1b and 4.2.2b.

Conditions 1 and 2 on the induction path allow us to track how how path direction and collider status changed from P. We must reason about the orientations of the intersection nodes in the original and new paths to ensure that the new path remains d-connected, acyclic, clean, and satisfying Conditions 1 and 2.

Intuitively, nodes that satisfy Condition I are those that originally appeared on a descendant path of a collider in the original path. For example, node x in Figures 4.2.1b and 4.2.2b falls into this category. These nodes satisfy Conditions 1(a) and 2.

In the case of nodes appearing on the second path of two intersecting descendant paths, such as y in Figure 4.2.2b, the rerouted path P' enters with direction $\leftarrow$, so Condition 2 does not apply. Depending on whether the node is the conditioned descendant or not, it will satisfy either 1(b) or 1(a), respectively.

Condition II is satisfied by nodes that were colliders in the original path and for which the rerouting process altered the path structure to avoid overlaps. This set includes two types of colliders: those whose descendant paths intersected the original path and the first of a pair of colliders whose descendant paths intersected each other. In both cases, the collider w satisfies Condition 1(a) through its original descendant path. Additionally, since $\mathsf{dir}_P(w) = \leftarrow$ and $\mathsf{dir}_{P'}(w) = \rightarrow$,as with node q in Figures 4.2.1b and 4.2.2b, we check Condition 2, which is satisfied because $w \notin Z$ (since it has a descendant path). For the second collider w' in such a pair, the path direction does not change at w', i.e., $\mathsf{dir}_P(w') = \mathsf{dir}_{P'}(w')$, since the rerouted path reuses the same outgoing edge from w'.

At a high level, by preserving the roles and directions of nodes carefully and redirecting overlaps only when safe, we can maintain the required structure and ensure all conditions of cleanliness hold. The mechanized proof in Coq is available for full rigor. □

It is clear that with Lemma 4.2.4, Theorem 4.2.2 is true.

We can now carry out our implementation of f^{path}, since we can assume by Theorem 4.2.2 that there exists a clean d-connected path between u and v.

In Coq, we must be able to express not just the path portion of a clean d-connected path but also the disjoint descendant paths and the collider that each corresponds to so that we can access the nodes in the descendant paths to assign specific `nodefuns`.

We again use the `assignments` type, this time mapping collider c_i to a tuple `nodes * node`. Let D be of type `assignments (nodes * node)`. If $c_i \in Z$, then D at c_i is simply $([], c_i)$. Otherwise, D at c_i is a tuple (p_i, d_i), where p_i represents the (possibly empty) intermediate nodes on the directed path from p_i to d_i, and $d_i \in Z$.

We can then formalize Definition 4.2.1 to be a `Prop`, given such a set D representing the descendant paths of clean d-connected path (u, v, l) in graph $\mathcal{G}$ with conditioning set Z:

```
1  Definition descendant_paths_disjoint (D: assignments (nodes * node))
2                      (u v: node) (l: nodes) (G: graph) (Z: nodes): Prop :=
3    forall (c: node), In c (find_colliders_in_path (u, v, l) G)
4        -> get_assigned_value D c = Some ([], c) /\ In c Z
5           \/
6           exists (p: nodes) (d: node), get_assigned_value D c = Some (p, d)
7             /\ In d Z /\ is_directed_path_in_graph (c, d, p) G = true
8                       /\ acyclic_path_2 (c, d, p)
9             /\ overlap (c :: p) Z = false
10            /\ overlap (p ++ [d]) (u :: l ++ [v]) = false
11            /\ forall (c' d': node) (p': nodes),
12               c =? c' = false /\ get_assigned_value D c' = Some (p', d')
13               -> overlap (c :: p ++ [d]) (c' :: p' ++ [d']) = false.
```

In the above, line 4 is the case that the collider c is conditioned on, and thus we do not need a descendant path. Lines 6-8 describe the alternate case, that instead there is an acyclic, directed path from c to a conditioned descendant d along p. Line 9 ensures that d is the first node in the path that is in Z, and line 10 ensures that the descendant path does not intersect the path from u to v itself. Lines 11-13 ensure that no two different descendant paths given by D overlap each other.

Definition 4.2.5. Given a clean d-connected path $P = (u, v, l)$ conditioned on Z, then D is a **descendant map** for P if `descendant_paths_disjoint`$(D, u, v, l, \mathcal{G}, Z)$ holds.

It is clear by Definition 4.2.1 that any clean d-connected path will have at least one descendant map.

4.3 Existence of Node Set Assignments

As described in Section 4.1.4, f^{path} relies on a partition of $\mathcal{V}$ into $S_1, S_2, ..., S_6$. However, in order to implement f^{path}, we need to know the specific indices to which to bind the nodes in

S_2, S_3, and S_4. For example, for $w \in S_2$, we must have the structure $p_w \to w$ in the path, and we wish to assign w the `nodefun f_parent_i X i`, where p_w is at index i in $\mathsf{Pa}(w)$, the list of w's parents.

Specifically, define the following assignments A_2, A_3, A_4, corresponding to the nodes of S_2, S_3, and S_4 respectively:

- A_2: `assignments nat`. Maps node $w \in S_2$ to i, corresponding to the index of w's single parent in the path in w's parent list.
- A_3: `assignments (nat * nat * X * X)`. Maps node $w \in S_3$ to (i, j, x, y), where i and j are the indices of w's two parents in the path in w's parent list, x is the value of w's conditioned descendant given by A_Z, and y is some value unequal to x.
- A_4: `assignments nat`. Maps node $w \in S_4$ to i, corresponding to the index of the previous node in the descendant path in w's parent list.

Specifically, for the clean d-connected path from u to v, given the set S_1 and the assignments A_2, A_3, A_4, as well as some arbitrary default `nodefun` h_{def}, and the conditioned assignments A_Z, we can implement the `graphfun` f^{path}:

```
Definition g_path (X: Type) `{EqType X} (S1: nodes) (A2: assignments nat)
                  (A3: assignments (nat * nat * X * X)) (A4: assignments nat)
                  (AZ: assignments X) (default: nodefun X)
                  (w: node): nodefun X :=
  match member S1 w with
  | true => f_unobs X
  | false =>
          match get_assigned_value A2 w with
          | Some i => f_parent_i X i
          | None =>
                match get_assigned_value A3 w with
                | Some (i, j, x, y) => f_equate_ij X i j x y
                | None =>
                      match get_assigned_value A4 w with
                      | Some i => f_parent_i X i
                      | None =>
                            match get_assigned_value AZ w with
                            | Some x => f_constant X x
                            | None => default
                            end
                      end
                end
          end
  end.
```

$$f^{\mathsf{path}} := \mathtt{g\_path}(S_1, A_2, A_3, A_4, A_Z, h_{\text{def}}). \tag{4.3.1}$$

We now must show the existence of these sets and assignments for arbitrary d-connected nodes u and v. For the following lemmas, let P be a clean d-connected path conditioned on Z between u and v in $\mathcal{G}$. Let S_1, S_2, S_3, S_4, S_5, and S_6 be the sets described by the partition for f^{path} given P. Let D be any descendant map for P, as described in Definition 4.2.5.

Lemma 4.3.1. *There exists A_2, assignments for the nodes of S_2 to $\mathbb{N}$, satisfying that for all $(w, i) \in A_2$, there exists a node a such that a is the i-th node in $\mathsf{Pa}(w)$, and either $a \to w$ or $w \leftarrow a$ is a subpath in P.*

Proof. We perform strong induction on the length of P. In the base case, P is simply an edge between u and v. Suppose P is $u \to v$. Then, $S_2 = \{v\}$. Since u is a parent of v, we can determine i_u such that u is the i_u-th node in $\mathsf{Pa}(v)$. Let $A_2 := \{v : i_u\}$. Similarly, if P is $u \leftarrow v$, then $S_2 = \{u\}$. Let i_v be the index of v in $\mathsf{Pa}(u)$. Then, let $A_2 := \{u : i_v\}$.

For the induction step, suppose the statement is true for paths of length k, where $2 \leq k < n$. Suppose P has length $n > 2$. Let w be the node following u in P. We consider different cases for the edge orientations at the start of the path surrounding w:

Case 1: $u \leftarrow w \cdots v$. By the induction hypothesis, there is a set A_2' corresponding to S_2' for the path $w \cdots v$. Note that if the path continues into w, e.g., $w \leftarrow \cdots v$, then $w \in S_2'$, and $w \in S_2$. If the path continues out of w, e.g., $w \to \cdots v$, then $w \in S_1'$, and $w \in S_1$. Thus, $S_2 = \{u\} \cup S_2'$. Let i_w be the index of w in $\mathsf{Pa}(u)$. Then, we define $A_2 := \{u : i_w\} \cup A_2'$.

Case 2: $u \to w \to \cdots v$. By the induction hypothesis, there is a set A_2' corresponding to S_2' for the path $w \to \cdots v$. Note that $w \in S_1'$, but $w \in S_2$. Furthermore, $u \in S_1$. Thus, $S_2 = \{w\} \cup S_2'$. Let i_u be the index of u in $\mathsf{Pa}(w)$. Then, we define $A_2 := \{w : i_u\} \cup A_2'$.

Case 3a: $u \to w \leftarrow v$. Consider the case of a three-node path, in which w is a collider. Then, $S_2 = \emptyset$, so let $A_2 := \emptyset$.

Case 3b: $u \to w \leftarrow w' \cdots v$. Now suppose there are more than three nodes in the path. By the induction hypothesis, there is a set A_2' corresponding to S_2' for the path $w' \cdots v$. Note that $w \in S_3$, and $u \in S_1$. Thus, $S_2 = S_2'$. So, we define $A_2 := A_2'$.

Thus, we can define A_2 as desired for a path of length n for all cases of edge orientations. □

Lemma 4.3.2. *There exists A_3, assignments for the nodes of S_3 to $\mathbb{N} \times \mathbb{N} \times \mathtt{X} \times \mathtt{X}$, satisfying that for all $(w, (i, j, x, y)) \in A_3$, there exist nodes a, b, such that a and b are the i-th and j-th nodes in $\mathsf{Pa}(w)$, respectively, and $a \to w \leftarrow b$ is a subpath in P. Furthermore, there exist list of nodes p and node d such that $(w, (p, d)) \in D$, $A_Z(d) = x$, and $x \neq y$.*

Proof. Again, we proceed via strong induction on the length of P. For the base case, P has only two endpoints and no intermediate nodes, so $S_3 = \emptyset$. Thus, $A_3 := \emptyset$.

For the induction step, we follow the same setup as Lemma 4.3.1: suppose the statement is true for paths of length k, where $2 \leq k < n$ and that P has length n. Let w be the node following u in P. By the induction hypothesis, there is a set A_3' corresponding to S_3' for the path $w \cdots v$. Suppose w is not a collider in P. Then, $S_3 = S_3'$, and D is also a descendant map for the path $w \cdots v$. Thus, we define $A_3 := A_3'$.

Now suppose w is a collider in P. We consider two cases:

Case 1: $u \to w \leftarrow v$. Here, P is the three-node path in which w is a collider. Then, $S_3 = \{w\}$. Since u and v are parents of w, they must appear in $\mathsf{Pa}(w)$. Let i_u and i_v be the indices of u and v in $\mathsf{Pa}(w)$, respectively. If $w \in Z$, let $x := A_Z(w)$. Otherwise, there exists p, d such that $(w, (p, d)) \in D$. Let $x := A_Z(d)$. Since we assume that `X` has at least two distinct elements, we can choose a y such that $y \neq x$. Let $A_3 := \{w : (i_u, i_v, x, y)\}$.

Case 2: $u \to w \leftarrow w' \cdots v$. Now there are more than three nodes in the path. Consider the path $w' \cdots v$. It is clear that D is also a descendant map for this path, since any overlaps in the descendant paths with each other or with the path itself would also occur for P. Then, by the induction hypothesis, there exists A'_3 corresponding to S'_3 for the path $w' \cdots v$. Note that $S_3 = \{w\} \cup S'_3$. Thus, we only need to add on an entry for w to A'_3, and we do so using the same methods as Case 1, defining $A_3 := \{w : (i_u, i_v, x, y)\} \cup A'_3$. □

Lemma 4.3.3. *There exists A_4, assignments for the nodes of S_4 to $\mathbb{N}$, satisfying that for all $(w, i_a) \in A_4$, there exists i_D, nodes c, d, a and list of nodes p, such that $(c, (p, d)) \in D$ where $c \neq d$, w is at index i_D in path (c, d, p), a is at index $i_D - 1$ in (c, d, p), and a is the i_a-th node in $\mathsf{Pa}(w)$.*

Proof. If $w \in S_4$, then w must belong to the descendant path of some collider in D. We define the following function which, given a descendant path (c, d, p), assigns the correct index bindings for the nodes on that path.

```
Fixpoint get_A4_nodes_for_path (c d: node) (p: nodes) (G: graph):
                               option (assignments nat) :=
  match p with
   | [] => match index (find_parents d G) c with
           | Some i => Some [(d, i)]
           | None => None
           end
   | h :: t => match index (find_parents h G) c with
               | Some i => match get_A4_nodes_for_path h d t G with
                           | Some r => Some ((h, i) :: r)
                           | None => None
                           end
               | None => None
               end
  end.
```

Induction on the length of p will show that `get_A4_nodes_for_path`$(c, d, p, \mathcal{G})$ exists as long as (c, d, p) is a directed path. Then, given D, we can go through each node of S_3 (colliders) and append the results of `get_A4_nodes_for_path` together to get A_4. Induction on the length of the path (which affects the number of nodes in S_3) shows that this result exists as long as D is a descendant map.

By construction, it is easy to see that any $(w, i_a) \in A_4$ computed as described above must correspond to a node in a descendant path and the index of the previous node in the path in $\mathsf{Pa}(w)$. Mechanistically, induction on the colliders of P will show that this result is true. $\square$

Thus, we have shown the existence of A_2, A_3, and A_4 for arbitrary clean d-connected path P from u to v. We can then define the `graphfun` f^{path} using Equation 4.3.1, which we will now show equates all values of noncollider nodes.

4.4 Equating Node Values and Conditioning on Z

We will now show that for a specific choice of unobserved-terms assignments U, f^{path} indeed equates noncollider values and properly conditions on Z. We will then select a sequence of unobserved-terms assignments to satisfy the requirements of Definition 3.2.6.

Definition 4.4.1. For any α, we say U is **source-fixed to α** if $U(w) = \alpha$ for all sources $w \in S_1$.

Let P be a clean d-connected path given Z and A_Z from u to v in $\mathcal{G}$. Let $S_1, ..., S_6$ be the partition as described in Section 4.1.4. Let U be any unobserved-terms assignments source-fixed to α. Choose some arbitrary `nodefun` h_{def}. Let A_2, A_3, A_4 be as given in Lemmas 4.3.1, 4.3.2, and 4.3.3, respectively. Let f^{path} be as described in Equation 4.3.1.

Theorem 4.4.2. *For all noncollider nodes $w \in P$, $f_U^{\mathsf{path}}(w) = \alpha$.*

Proof. Note that all noncollider nodes $w \in P$ must be in S_1 or S_2. It is clear that for all $w \in S_1$, $f_U^{\mathsf{path}}(w) = \alpha$, since $f_U^{\mathsf{path}}(w) = U(w) = \alpha$ (recall that the `nodefun` corresponding to a source is `f_unobs`).

Now consider any transmitter $w \in S_2$. Let i be the mapping of w given by A_2. Let a be the i-th node in $\mathsf{Pa}(w)$. By Lemma 4.3.1, either $a \to w$ or $w \leftarrow a$ is a subpath of P. Note that since the `nodefun` corresponding to w is `f_parent_i`, $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(\mathsf{Pa}(w)_i) = f_U^{\mathsf{path}}(a)$.

Consider first the case that $a \to w$ is a subpath of P. We induct on the index of the subpath in the path. For the base case, a and w are the first and second nodes of P, respectively. Then, $a \in S_1$, so $f_U^{\mathsf{path}}(a) = \alpha$. Thus, $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(a) = \alpha$, as desired.

For the induction hypothesis, suppose that all S_2 nodes coming before w in P evaluate to α. Note that since a has an arrow out to w, a is not a collider. Thus, $a \in S_1 \cup S_2$. If $a \in S_1$, then $f_U^{\mathsf{path}}(a) = \alpha$. If $a \in S_2$, then by the induction hypothesis, $f_U^{\mathsf{path}}(a) = \alpha$ as well. Thus, $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(a) = \alpha$, as desired.

Consider now the case that $w \leftarrow a$ is a subpath of P. Here, we induct on the index of the subpath $a \to w$ in the *reverse* of P. For the base case, a and w are the first and second nodes, respectively, in the reverse path; thus w and a are the second-to-last and last nodes, respectively, in P. Then, $a \in S_1$, so $f_U^{\mathsf{path}}(a) = \alpha$. Thus, $f_U^{\mathsf{path}}(w) = \alpha$.

For the induction hypothesis, suppose that all S_2 nodes coming before w in the reverse of P (or *after* w in P) evaluate to α. Again, a is not a collider, so $a \in S_1 \cup S_2$. If $a \in S_1$, then

$f_U^{\mathsf{path}}(a) = \alpha$. If $a \in S_2$, then $f_U^{\mathsf{path}}(a) = \alpha$ by the induction hypothesis. Thus, $f_U^{\mathsf{path}}(w) = \alpha$, as desired.

Thus, we have shown that all nodes $w \in S_1 \cup S_2$ (the noncollider nodes in P) all evaluate to α. □

We have now shown that for all noncollider nodes $w_i, w_j \in P$, $f_U^{\mathsf{path}}(w_i) = f_U^{\mathsf{path}}(w_j)$. Importantly, this result means that $f_U^{\mathsf{path}}(u) = f_U^{\mathsf{path}}(v)$, which intuitively tells us that u and v cannot be conditionally independent, as their values are equal. However, we still must show a sequence of unobserved-terms assignments that can propagate a change in $f_U^{\mathsf{path}}(u)$ all the way to v. We show this construction in Section 4.5.

Of course, f^{path} is of no use if it does not properly condition on Z given assignments A_Z. Thus, we must show that all nodes in Z evaluate to the correct values. Recall that U is source-fixed to α.

Theorem 4.4.3. *For all nodes $z \in Z$, $f_U^{\mathsf{path}}(z) = A_Z(z)$.*

Proof. First, we show that for all nodes $w \in S_3$ (colliders), if w maps to (i, j, x, y) in A_3, then $f_U^{\mathsf{path}}(w) = x$. Let a and b be the i-th and j-th nodes of $\mathsf{Pa}(w)$, respectively. Then, by Lemma 4.3.2, $a \to w \leftarrow b$ is a subpath of P. Since the `nodefun` corresponding to w is `f_equate_ij`,

$$f_U^{\mathsf{path}}(w) := \begin{cases} x & f_U^{\mathsf{path}}(a) = f_U^{\mathsf{path}}(b) \\ y & \text{else.} \end{cases}$$

Since a and b both have arrows out towards w, neither can be in S_3. Thus, both are in $S_1 \cup S_2$ (noncollider nodes in P). Then, by Theorem 4.4.2, $f_U^{\mathsf{path}}(a) = f_U^{\mathsf{path}}(b)$. Thus, $f_U^{\mathsf{path}}(w) = x$.

Now, consider any $z \in Z$. By construction of the partition, $z \in S_3 \cup S_4 \cup S_5$. Suppose $z \in S_5$ (one of the residual nodes in Z). Then, by definition, $f_U^{\mathsf{path}}(z) = A_Z(z)$. Furthermore, if $z \in S_3$ is a collider, then for descendant map D, we must have $(z, ([], z)) \in D$, since z does not need a descendant path. By Lemma 4.3.2, $A_Z(z) = x$. Then, by the above paragraph, $f_U^{\mathsf{path}}(z) = x = A_Z(z)$, as desired.

It remains to show that $f_U^{\mathsf{path}}(z) = A_Z(z)$ if $z \in S_4$ is the conditioned descendant of a collider. More specifically, $(c, (p, z)) \in D$ for some $c \in S_3$. We will show that for all nodes $w \in S_4$ in the path (c, z, p), $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(c)$. Let i be the mapping of w given by A_4. Let a be the i-th node in $\mathsf{Pa}(w)$. Since the `nodefun` corresponding to w is `f_parent_i`, $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(\mathsf{Pa}(w)_i) = f_U^{\mathsf{path}}(a)$.

We proceed via induction on the index of w in the path. For the base case, we assume w is the first node of p. Then, $a = c$, so $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(c)$. For the induction hypothesis, we assume all nodes in (c, z, p) prior to w evaluate to $f_U^{\mathsf{path}}(c)$. We can then apply the induction hypothesis on a, and we thus have $f_U^{\mathsf{path}}(w) = f_U^{\mathsf{path}}(a) = f_U^{\mathsf{path}}(c)$, as desired.

Then, since z is a node in the path (c, z, p), we have $f_U^{\mathsf{path}}(z) = f_U^{\mathsf{path}}(c)$. We furthermore know that $f_U^{\mathsf{path}}(c) = x$, where (i_c, j_c, x, y) is the mapping of c given by A_3. Lemma 4.3.2 says that $x = A_Z(z)$, so $f_U^{\mathsf{path}}(c) = A_Z(z)$. Thus, $f_U^{\mathsf{path}}(z) = A_Z(z)$, as desired. □

We have thus shown that with a source-fixed U, we have a function that properly conditions on Z and forces the values of all noncollider nodes, and importantly, the values of u and v, to be equal. We now demonstrate how this construction can be used to violate the definition of conditional independence.

4.5 The Sequence of Unobserved-Terms Assignments

We aim to use f^{path} to prove that u and v are *not* conditionally independent given Z, as described in Definition 3.2.6. We know that for any unobserved-terms assignments U that are source-fixed to some α, f_U^{path} conditions on Z and forces $f_U^{\mathsf{path}}(u) = f_U^{\mathsf{path}}(v)$. Choose some $\alpha \neq \beta$. Then, if U_α is source-fixed to α, we will have $f_{U_\alpha}^{\mathsf{path}}(u) = f_{U_\alpha}^{\mathsf{path}}(v) = \alpha$. If we can create a sequence $U_\beta, U_1, ..., U_\ell$ satisfying the requirements of Definition 3.2.6 such that U_ℓ is source-fixed to β, then $f_{U_\ell}^{\mathsf{path}}(u) = f_{U_\ell}^{\mathsf{path}}(v) = \beta$, and notably $f_{U_\alpha}^{\mathsf{path}}(v) \neq f_{U_\ell}^{\mathsf{path}}(v)$, proving that u and v are not conditionally independent given Z.

Consider the simple case of P being a single directed edge $u \to v$. Then, $S_1 = \{u\}$. We can simply define $U_\alpha = \{u : \alpha\}$ and $U_\beta = \{u : \beta\}$, where the assignments for other nodes are arbitrary for both U_α and $U_\beta = U_0$. Note that U_α and U_β indeed differ only for members of $\mathsf{Anc}_Z^*(u)$ (in particular, only u). Here, $\ell = 0$, and note that U_α and U_ℓ both properly condition on Z by Theorem 4.4.3, and furthermore $f_{U_\alpha}^{\mathsf{path}}(v) \neq f_{U_\ell}^{\mathsf{path}}(v)$, so u and v are not conditionally independent.

Now consider the case that P is a simple fork $u \leftarrow w \to v$. Here, $S_1 = \{w\}$. Define $U_\alpha = \{w : \alpha\}$, where other nodes are assigned arbitrarily. Note that changing the unobserved term of u will not cause any change in $f^{\mathsf{path}}(u)$, since $f^{\mathsf{path}}(u)$ depends on the value of w. However, w is an unblocked ancestor of u. Thus, we can define

$$U_\beta(w') := \begin{cases} \beta & w' = w \\ U_\alpha(w') & \text{else.} \end{cases}$$

Note that U_α and U_β are both source-fixed (to α and β, respectively), so if we let $\ell = 0$, we once again see that U_α and $U_\beta = U_\ell$ satisfy the requirements from Definition 3.2.6, and thus u and v are not conditionally independent.

Now, consider the case that P has a single collider: $u \to w \leftarrow v$. Suppose $w \in Z$, and (i, j, x, y) is the mapping of w given by A_3. Note that $A_Z(w) = x$ and $x \neq y$ by Lemma 4.3.2. Here, $S_1 = \{u, v\}$. Define $U_\alpha = \{u : \alpha, v : \alpha\}$. Define

$$U_\beta(w') := \begin{cases} \beta & w' = u \\ U_\alpha(w') & \text{else.} \end{cases}$$

Note that $f_{U_\beta}^{\mathsf{path}}(u) = U_\beta(u) = \beta$, but we are no longer properly conditioning on Z. In particular, $f_{U_\beta}^{\mathsf{path}}(v) = U_\beta(v) = \alpha$. Recall that

$$f_{U_\beta}^{\mathsf{path}}(w) := \begin{cases} x & f_{U_\beta}^{\mathsf{path}}(u) = f_{U_\beta}^{\mathsf{path}}(v) \\ y & \text{else,} \end{cases}$$

so $f_{U_\beta}^{\mathsf{path}}(w) = y \neq A_Z(w)$, since $\alpha \neq \beta$.

Thus, we define

$$U_1(w') := \begin{cases} \beta & w' = v \\ U_\beta(w') & \text{else.} \end{cases}$$

Note that U_1 differs from U_β for only $v \in \mathsf{Anc}_Z^*(w)$, which satisfies Condition 3 of Definition 3.2.6. We let $\ell = 1$, and note that U_ℓ is source-fixed to β. Thus, $f_{U_\ell}^{\mathsf{path}}(v) = \beta \neq f_{U_\alpha}^{\mathsf{path}}(v)$, so u and v are not conditionally independent.

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 conditionally independent.

Specifically, suppose there are ℓ sources, and suppose they are organized in order of their appearance in P, such that $S_1 = [s_0, s_1, ..., s_\ell]$. Choose some $\alpha \neq \beta$. Define U_α to be any unobserved-terms assignments source-fixed to α. Then define the following sequence of unobserved-terms assignments:

$$\begin{aligned} U_\beta &= \begin{cases} \beta & w = s_0 \\ U_\alpha(w) & \text{else} \end{cases} \\ U_1 &= \begin{cases} \beta & w = s_1 \\ U_\beta(w) & \text{else} \end{cases} \\ &\vdots \\ U_i &= \begin{cases} \beta & w = s_i \\ U_{i-1}(w) & \text{else} \end{cases} \\ &\vdots \\ U_\ell &= \begin{cases} \beta & w = s_\ell \\ U_{\ell-1}(w) & \text{else} \end{cases} \end{aligned} \tag{4.5.1}$$

Note that for all edge orientations of P, S_1 will have at least one node. Thus, the sequence will contain at least U_β, as needed for Definition 3.2.6.

Note that U_α satisfies Condition 1 of Definition 3.2.6. Furthermore, $f_{U_\alpha}^{\mathsf{path}}(v) = \alpha \neq \beta = f_{U_\ell}^{\mathsf{path}}(v)$ by Theorem 4.4.2. Thus, if we show that the sequence satisfies the conditions of Definition 3.2.6, then we will show that u and v are not conditionally independent.

To do so, we must establish relationships between the nodes of S_1 with other nodes in the path, particularly nodes of which they are unblocked ancestors.

Lemma 4.5.1. *The first node in P that is a member of S_1 is an unblocked ancestor of u.*

Proof. Proceed via induction on the length of P. For the base case, P consists of only u and v. If P is $u \to v$, then u is the first member of S_1, and $u \in \mathsf{Anc}_Z^*(u)$. If P is $u \leftarrow v$, then v is the first member of S_1, and it is clear that $v \in \mathsf{Anc}_Z^*(u)$, since $v \notin Z$ by assumption.

For the induction hypothesis, we assume that for paths of length n, the first node in S_1 is an unblocked ancestor of the first node in the path. Suppose P has length $n+1$, where $n \geq 2$. If the edge out of u is outwards, then u is the first member of S_1, and $u \in \mathsf{Anc}_Z^*(u)$. Otherwise, P is $u \leftarrow w \cdots v$ for some node w. Let P' be the subpath $w \cdots v$ with respective partitions S_1', S_2', etc. Note that $u \in S_2$, so the first node in P' in S_1', a, will also be the first node in S_1. By the induction hypothesis, a is an unblocked ancestor of w. Note that since P is d-connected, $w \notin Z$, since w is not a collider. Thus, $a \in \mathsf{Anc}_Z^*(u)$. □

Lemma 4.5.2. *If $x, y \in S_1$ are consecutive sources (no nodes between them on P are in S_1), then there exists a node $z \in Z$ such that $x \in \mathsf{Anc}_Z^*(z)$ and $y \in \mathsf{Anc}_Z^*(z)$.*

Proof. Intuitively, x and y must both have arrows out; in other words, $x \rightarrow \cdots \leftarrow y$ is a subpath of P. Then, at some point in the subpath, there must be a collider. Since P is d-connected, this collider has a conditioned descendant $z \in Z$, of which x and y are unblocked ancestors.

To formally prove the statement, we induct on the length of the path. Note that P must have length at least 3 in order to have two nodes in S_1. Thus, for the base case, P is $u \rightarrow w \leftarrow v$ (the other arrow orientations result in only one node in S_1), where $u = x$ and $v = y$. If $w \in Z$, then let $z = w$. It is clear that $x \in \mathsf{Anc}_Z^*(w)$ and $y \in \mathsf{Anc}_Z^*(w)$. Otherwise, w must have a conditioned descendant $z \in Z$ such that there is a directed path from w to z that does not go through other nodes in Z. Then, it is clear that $x \in \mathsf{Anc}_Z^*(z)$ and $y \in \mathsf{Anc}_Z^*(z)$.

Suppose the statement is true for paths P' of length n and corresponding set of nodes S_1'. Suppose P has length $n+1$, where $n \geq 3$, so P is $u \leftrightarrow w_1 \leftrightarrow w_2 \cdots v$. Let P' be the subpath $w_1 \cdots v$. If P has an arrow into u, then $u \in S_2$, so x, y are still consecutive nodes in S_1' corresponding to P'. Thus, we can apply the induction hypothesis on P' to get the desired z.

Now consider the case that P has an arrow out of u. If $u \neq x$, then x and y are again still consecutive nodes in S_1' corresponding to P', so we again can apply the induction hypothesis on P'. Suppose $u = x$. We first consider the case that P is $u \rightarrow w_1 \rightarrow w_2 \cdots v$. Then, note that the first node of S_1' will be w_1, and the second will be y. Thus, w_1 and y are consecutive nodes in S_1', so we can find a $z \in Z$ such that $w_1 \in \mathsf{Anc}_Z^*(z)$ and $v \in \mathsf{Anc}_Z^*(z)$. Then, $u \in \mathsf{Anc}_Z^*(z)$ since $u \rightarrow w_1$ is an edge and $w_1 \notin Z$, since w_1 is a mediator.

We now consider the case that P is $u \rightarrow w_1 \leftarrow w_2 \cdots v$. By Lemma 4.5.1, $y \in \mathsf{Anc}_Z^*(w_1)$. Since w_1 is a collider, it has a conditioned descendant $z \in Z$. Applying the same logic as the base case, we have that $u = x \in \mathsf{Anc}_Z^*(z)$, as desired. □

We now show that the sequence satisfies Condition 3 of Definition 3.2.6.

Theorem 4.5.3. *For the sequence defined in Equation 4.5.1, each two consecutive unobserved-terms assignments U_{i-1}, U_i in the sequence differ from each only for*

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

where we let $U_{i-2} = U_\alpha$ *if* $i = 1$.

Proof. Note that the statement applies only to paths with at least two sources, since ℓ must be at least 1 for U_{i-2} to make sense. Consider U_i and U_{i-1} for some $i \geq 1$. By definition, they differ from each other only for s_i. By Lemma 4.5.2, there is a node $z \in Z$ such that $s_{i-1}, s_i \in \mathsf{Anc}_Z^*(z)$. Note that by definition, $U_{i-2}(s_{i-1}) = \alpha \neq \beta = U_{i-1}(s_i)$. Thus, the statement holds.

In order to prove the above formally, we proceed via induction on the length of S_1, proving a key lemma that S_1' corresponding to the subpath $s_i \cdots v$ of P is the subset of S_1 containing nodes beginning from s_i and continuing to the end of the subpath. □

We can now prove Lemma 4.1, restated below for convenience.

Lemma 4.1. For a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two different nodes $u, v \in \mathcal{V}$, and a conditioning set $Z \subseteq \mathcal{V}$ with $u, v \notin Z$, if u and v are conditionally independent given Z, then they are d-separated given Z in $\mathcal{G}$.

Proof. We proceed via the contrapositive. Suppose that u and v are not d-separated, so there exists a clean d-connected path P from u to v. Let $U_\alpha, U_\beta, U_1, ..., U_\ell$ be defined as in Equation 4.5.1. We show that all conditions of Definition 3.2.6 are satisfied, using graph function f^{path}:

1. Since U_α is source-fixed to α, $f_{U_\alpha}^{\mathsf{path}}(u) = \alpha$, and the function properly conditions on Z by Theorems 4.4.2 and 4.4.3, respectively.

2. By construction, U_β differs from U_α for only s_0, where $s_0 \in \mathsf{Anc}_Z^*(u)$ by Lemma 4.5.1. To show that $f_{U_\beta}^{\mathsf{path}}(u) = \beta$, we consider the edge orientation associated with u on P: if P has an edge out of u, then $u \in S_1$, so $f_{U_\beta}^{\mathsf{path}}(u) = U_\beta(u) = \beta$. Otherwise, $u \in S_2$. Via induction on the length of the path, we show that the chain of S_2 nodes starting at u will terminate at s_0, and all nodes along the chain will take on the value of s_0, which is β.

3. By Theorem 4.5.3, this condition is satisfied.

4. Note that $\ell = |S_1| - 1$, and $|S_1| \leq |\mathcal{V}|$, so the condition is satisfied.

   For a tighter bound, we note that at most every other node in P can belong to S_1, since between two consecutive nodes, one must be the parent of the other, so at most one can be in S_1. Thus, $|S_1| \leq \frac{1}{2}|P|$. Thus, for constructing sequences to violate conditional independence, we can bound ℓ by half the length of the shortest clean d-connected path from u to v.

5. Since U_ℓ is source-fixed to β, $f_{U_\ell}^{\mathsf{path}}(u) = \beta$, and the function properly conditions on Z by Theorems 4.4.2 and 4.4.3, respectively.

By Theorem 4.4.2, $f_{U_\alpha}^{\mathsf{path}}(v) = \alpha \neq \beta = f_{U_\ell}^{\mathsf{path}}(v)$. Thus, u and v are not conditionally independent. □

Chapter 5

Backward Direction: d-Separation Implies Conditional Independence

We now prove the backward direction of Theorem 3.3.1, again restated below for clarity.

Lemma 5.1. For a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two different nodes $u, v \in \mathcal{V}$, and a conditioning set $Z \subseteq \mathcal{V}$ with $u, v \notin Z$, if u and v are d-separated given Z, then they are conditionally independent.

We prove the contrapositive: assume that u and v are not conditionally independent, so there exists some graph function f and sequence of unobserved-terms assignments $U_\alpha, U_\beta, U_1, ..., U_\ell$ satisfying the conditions of Definition 3.2.6, such that $f_{U_\alpha}(v) \neq f_{U_\ell}(v)$. 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.

5.1 Change Originates From Unblocked Ancestors

While in Chapter 4, 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 $a \in \mathsf{Pa}(v)$. 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 $a' \in \mathsf{Pa}(a)$. 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 $z \in Z$.

Thus, we see that a change in $f(v)$ between two different unobserved-terms assignments must originate from an unblocked ancestor of v. Specifically, some unblocked ancestor of v must have a different unobserved term in the two assignments. We formally describe this conclusion in the following lemma:

Lemma 5.1.1. *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 $a \in \mathsf{Anc}^*_Z(w)$ such that $U(a) \neq U'(a)$.*

Proof. We proceed via strong induction on the index of w in the topological sort of $\mathcal{G}$. For the base case, suppose w is the first node in the topological sort. Then, it has no parents, so $f_U(w)$ and $f_{U'}(w)$ depend only on $U(w)$ and $U'(w)$, respectively. Thus, it must be true that $U(w) \neq U'(w)$, so we can simply let $a = w$, where clearly $w \in \mathsf{Anc}^*_Z(w)$.

For the induction hypothesis, assume that the statement is true for all nodes with index at most i in the topological sort of $\mathcal{G}$, and suppose the index of w is $i+1$. If $U(w) \neq U'(w)$, we can once again simply let $a = w$. Assume $U(w) = U'(w)$. Then, there must be some $w' \in \mathsf{Pa}(w)$ such that $f_U(w') \neq f_{U'}(w')$, since $f_U(w) \neq f_{U'}(w)$. Since $w' \in \mathsf{Pa}(w)$, the index of w' in the topological sort of $\mathcal{G}$ must be less than $i+1$. Thus, by the induction hypothesis, there exists a node $a \in \mathsf{Anc}^*_Z(w')$ such that $U(a) \neq U'(a)$. Note that since f_U and $f_{U'}$ both properly condition on Z, it must be true that $w' \notin Z$, since $f_U(w') \neq f_{U'}(w')$. Thus, $a \in \mathsf{Anc}^*_Z(w)$, as desired. □

Lemma 5.1.1 already leads us towards a d-connected path to v because it points us to a specific unblocked ancestor of v. Unblocked ancestors are useful for constructing d-connected paths because they ensure that all mediators on the directed path are not conditioned on and furthermore that the ancestor itself, which could be a mediator or a confounder if the path is further extended, is also not conditioned on. We will see how to use unblocked ancestors to discover a d-connected path from a sequence of unobserved-terms assignments.

5.2 *d*-Connected Paths For Short Sequences

We sketch the processes of finding a d-connected path between u and v for short sequences of unobserved-terms assignments that satisfy the conditions of Definition 3.2.6 but change the value of v.

We begin with a lemma that finds the existence of a d-connected path given a shared unblocked ancestor of two nodes:

Lemma 5.2.1. *For distinct nodes w_1, w_2, if there is a node a such that $a \in \mathsf{Anc}^*_Z(w_1) \cap \mathsf{Anc}^*_Z(w_2)$, then one of the following is true:*

1. *There is a d-connected, acyclic directed path (w_1, w_2, l) such that $w_1 \notin Z$.*
2. *There is a d-connected, acyclic directed path (w_2, w_1, l) such that $w_2 \notin Z$.*
3. *There is a d-connected, acyclic path $(w_1, w_2, \texttt{rev}(l_1)\texttt{++}[a']\texttt{++}l_2)$ such that (a', w_1, l_1) and (a', w_2, l_2) are both directed.*

Proof. Since a is a shared unblocked ancestor of w_1, w_2, we use Definition 3.2.3 to produce the following cases:

1. $a = w_1$, shown in Figure 5.2.1a. Then, since w_1, w_2 are distinct, $a \neq w_2$, so there is a directed path to w_2 that does not go through any member of Z. Then, the path is d-connected, since all intermediate nodes are mediators. Furthermore, since $a \in \mathsf{Anc}^*_Z(w_2)$ and $a \neq w_2$, $a = w_1 \notin Z$, as desired. Thus, Condition 1 is satisfied.

2. $a \neq w_1$, $a = w_2$, shown in Figure 5.2.1b. By symmetric logic as the case above, Condition 2 is satisfied.

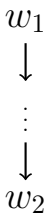
3. $a \neq w_1, a \neq w_2$. Then, there are directed paths from a to u and from a to v, both of which do not go through members of Z. It is clear that $a \notin Z$. If the two directed paths do not overlap, shown in Figure 5.2.1c, then we simply concatenate them and satisfy Condition 3. If they do overlap, shown in Figure 5.2.1d, then we find the first intersection a' of the reverse paths given by Theorem 2.3.3, and we take the path from w_1 to a' via the reverse of l_1, then to w_2 via l_2. By Theorem 2.3.3, this path is acyclic. Since the original directed paths do not pass through Z, this path is d-connected. Furthermore, it is clear that the two paths making up the resulting path are directed, since they are subpaths of the original directed paths.

Thus, one of the conditions is satisfied in all cases. □

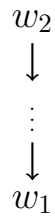
Suppose $\ell = 0$, so we change the value of v between U_α and U_β. Then, since U_α and U_β both properly condition on Z, Lemma 5.1.1 tells us that there exists a node $a \in \mathsf{Anc}^*_Z(v)$ such that $U_\alpha(a) \neq U_\beta(a)$. However, U_α and U_β are constrained such that they only differ for values in $\mathsf{Anc}^*_Z(u)$. Thus, $a \in \mathsf{Anc}^*_Z(u) \cap \mathsf{Anc}^*_Z(v)$. We can then apply Lemma 5.2.1, seeing that in all three cases, there is a d-connected path between u and v, and thus u and v are d-connected.

Now, consider the case that $\ell = 1$, so $f_{U_\alpha}(v) \neq f_{U_1}(v)$. Again, by Lemma 5.1.1, there exists a node $a \in \mathsf{Anc}^*_Z(v)$ such that $U_\alpha(a) \neq U_1(a)$. If $U_\beta(a) = U_1(a)$, then we follow the same steps as $\ell = 0$ above to see that u and v are d-connected. Otherwise, if $U_\beta(a) \neq U_1(a)$, then there must exist $z \in Z$ such that $a \in \mathsf{Anc}^*_Z(z)$, and there is an $a' \in \mathsf{Anc}^*_Z(z)$ such that $U_\alpha(a') \neq U_\beta(a')$. Then, $a' \in \mathsf{Anc}^*_Z(u)$. We can then construct the path shown in Figure 5.2.2a. We again must consider the possibility that the directed paths making up the path overlap each other, which we handle formally in Section 5.4. 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 must overlap at some node c, and they then can 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 5.2.2b.

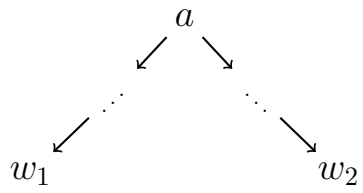
We have already taken advantage of many components of d-connectedness, such as mediators and confounders being unconditioned and colliders either being in the conditioning set or having conditioned descendants. However, thus far we have only constructed examples involving paths with at most one collider and two confounders. To establish full equivalence, we expect to rely on d-connected paths of arbitrary structure with any number of mediators, confounders, and colliders, since intuitively, every d-connected path from u to v should



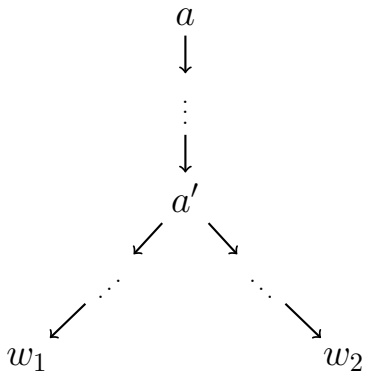
(a) $a = w_1$ and has a directed path to w_2.



(b) $a = w_2$ and has a directed path to w_1.

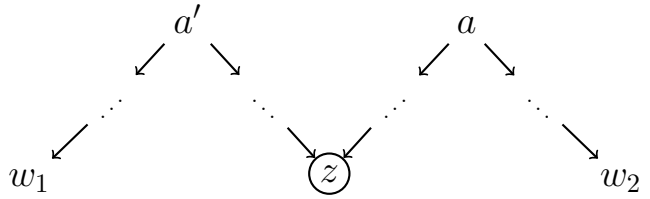


(c) The two directed paths from a to w_1 and w_2 do not overlap.

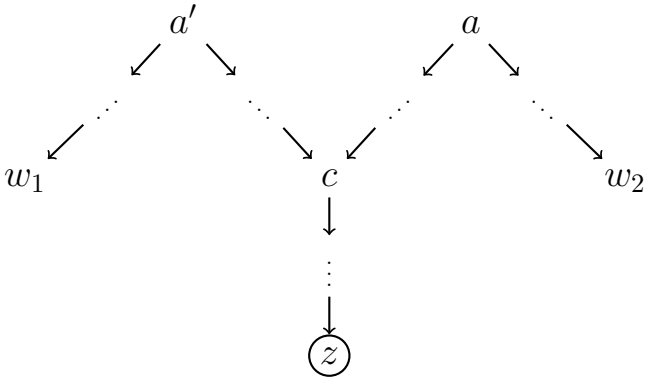


(d) The two directed paths from a to w_1 and w_2 overlap at a'.

Figure 5.2.1: Given that $a \in \mathsf{Anc}^*_Z(w_1) \cap \mathsf{Anc}^*_Z(w_2)$, there are four possible cases for the path structure between w_1 and w_2, resulting in either a directed path or a single-confounder path.



(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.

Figure 5.2.2: For any w_1, w_2 such that w_2's value is affected by a change in w_1 with a sequence of unobserved-terms assignments U_α, U_β, U_1, we can construct a d-connected path from w_1 to w_2 using $z \in Z$ and shared ancestors a', a.

correspond to some valid sequence of unobserved-terms assignments that alters the value of v without violating conditioning on Z. This observation highlights why our definition of semantic conditional independence requires a *sequence* of assignments rather than a single comparison: 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 arbitrary-length sequences that satisfy the constraints in Definition 3.2.6, thereby leveraging the full expressive power of d-connectedness.

5.3 Generalizing to Arbitrary-Length Sequences

As illustrated in the previous section, the change in $f(v)$ originates from u and propagates through a chain of intermediate assignments. At a high level, each transition from U_i to U_{i+1} introduces a change via a different unblocked ancestor whose effect passes through a shared descendant $z \in Z$. Each such transition introduces a collider to the overall path, and the full path formed by concatenating the directed paths from unblocked ancestors to descendants in Z is d-connected. The existence of this path ultimately shows that u and v must be d-connected.

A natural strategy for generalizing this logic to arbitrary-length sequences $U_\alpha, U_\beta, U_1, ..., U_\ell$ is to attempt induction on ℓ. However, this approach does not directly succeed. Suppose the induction hypothesis were that if a sequence of unobserved-terms assignments of length n demonstrates conditional dependence between two nodes, then there exists a d-connected path between them. Now suppose we have a sequence of length $n+1$ for u and v. One might hope to apply the inductive hypothesis to the subsequence $U'_\alpha = U_\beta, U'_\beta = U_1, ..., U'_n = U_{n+1}$ to obtain a d-connected path from some $z \in Z$ to v, and then prepend a d-connected path from u to z, which exists via the constraints on U_α, U_β, and U_1. However, this approach fails because the initial step U_β is constrained to differ from U_α only on unblocked ancestors of u, while later steps may involve changes across ancestors of many different nodes $z_1, \ldots, z_k \in Z$. Since there is no guarantee of a d-connected path from each of these z_i to v, we cannot inductively extend the argument as hoped.

Instead, we could attempt to apply the induction hypothesis to the prefix $U_\alpha, U_\beta, U_1, ..., U_n$ to produce a d-connected path from u to $z \in Z$, then append a d-connected path from z to v. However, we encounter another asymmetry: U_{n+1} is special in that it must fully recondition on Z, while any intermediate U_i need not. This asymmetry once again prevents a clean inductive step.

Note that we do not assume that the given sequence is minimal in that the value of v necessarily changes only at the final step. It is possible that the value of v has already changed after the changes made in U_β, for example. However, we can see that if the value of v changes *after* U_β, then some $z \in Z$ must have been affected during the propagation, and its change enabled the downstream effect on v.

To make this reasoning precise, we now define functions to identify the set of $z \in Z$ whose values are affected during the sequence. These nodes serve as intermediaries in the

propagation chain and will be used to construct an explicit d-connected path from u to v.

```
Fixpoint find_unblocked_ancestors_in_Z_contributors {X: Type} `{EqType X}
        (G: graph) (Z: nodes) (AZ: assignments X) (S: nodes): nodes :=
  match AZ with
  | [] => []
  | (z, x) :: AZ' => if overlap (find_unblocked_ancestors G z Z) S
            then z :: find_unblocked_ancestors_in_Z_contributors G Z AZ' S
            else find_unblocked_ancestors_in_Z_contributors G Z AZ' S
  end.

Fixpoint get_conditioned_nodes_that_change_in_seq {X: Type} `{EqType X}
            (L: list (assignments X)) (Z: nodes) (AZ: assignments X)
            (G: graph): nodes :=
  match L with
  | U1 :: L' => match L' with
                | U2 :: U3 :: L''' =>
    find_unblocked_ancestors_in_Z_contributors G Z AZ
        (unblocked_ancestors_that_changed_A_to_B (nodes_in_graph G) U1 U2)
    ++ get_conditioned_nodes_that_change_in_seq L' Z AZ G
                | _ => []
                end
  | _ => []
  end.
```

In the above, `unblocked_ancestors_that_changed_A_to_B`$(\mathcal{V}, U_1, U_2)$ returns every node with an unobserved term differing between U_1 and U_2. Let

$$\Delta_Z(L) := \texttt{get\_conditioned\_nodes\_that\_change\_in\_seq}(L, Z, A_Z, \mathcal{G}).$$

If $L = [U_\alpha, U_\beta, U_1, ..., U_\ell]$, then $\Delta_Z(L)$ gives the subset of Z whose values were affected by changes in the sequence $U_1, \ldots, U_\ell$.

We now prove a result similar to Lemma 5.1.1 tailored to sequences:

Lemma 5.3.1. *For any graph function f and sequence of unobserved-terms assignments $U_\alpha, U_\beta, U_1, ..., U_\ell$ satisfying the conditions of Definition 3.2.6, if $f_{U_\alpha}(v) \neq f_{U_\ell}(v)$, then there exists a node $a \in \mathsf{Anc}^*_Z(v)$ such that one of the following is true:*

1. *$a \in \mathsf{Anc}^*_Z(u)$.*
2. *There exists $z \in Z$ such that $a \in \mathsf{Anc}^*_Z(z)$ and $z \in \Delta_Z(U_\alpha, U_\beta, U_1, ..., U_\ell)$.*

Proof. By Lemma 5.1.1, we know that there is a node $a \in \mathsf{Anc}^*_Z(v)$ such that $U_\alpha(a) \neq U_\ell(a)$. If $a \in \mathsf{Anc}^*_Z(u)$, then Condition 1 is satisfied. Suppose $a \notin \mathsf{Anc}^*_Z(u)$.

Then, $U_\alpha(a) = U_\beta(a)$, so $\ell \geq 1$. To show that Condition 2 above is satisfied for any $U_\alpha, U_\beta, U_1, ..., U_\ell$ satisfying Condition 3 of Definition 3.2.6, we perform induction on ℓ. Note that for the rest of the proof, we no longer require that the sequence satisfy Condition 2 of Definition 3.2.6.

For the base case, $\ell = 1$. Then, $U_\beta(a) \neq U_1(a)$, so there must be a node $z \in Z$ and a node a' such that $a \in \mathsf{Anc}^*_Z(z)$, $a' \in \mathsf{Anc}^*_Z(z)$, and $U_\alpha(a') \neq U_\beta(a')$. Then, letting

$$S := \texttt{unblocked\_ancestors\_that\_changed\_A\_to\_B}(\mathcal{V}, U_\alpha, U_\beta),$$

we have that

$$z \in \texttt{find\_unblocked\_ancestors\_in\_Z\_contributors}(\mathcal{G}, Z, A_Z, S),$$

and thus $z \in \Delta_Z(U_\alpha, U_\beta, U_1)$.

For the induction hypothesis, we assume that the statement is true for $\ell = n$. Suppose the sequence has length $\ell = n + 1$, where $n \geq 2$. If $U_\beta(a) \neq U_1(a)$, proceed as in the base case. Otherwise, apply the induction hypothesis to the sequence $U'_\alpha = U_\beta, U'_\beta = U_1, ...$, which tells us that there must be a node $z \in Z$ such that $a \in \mathsf{Anc}^*_Z(z)$ and $z \in \Delta_Z(U_\beta, U_1, U_2, ..., U_{n+1})$. Then, by definition of Δ_Z, we have that $z \in \Delta_Z(U_\alpha, U_\beta, U_1, ..., U_{n+1})$, as desired. □

We now can pinpoint the change in v's value to a specific conditioned node z. We now relate syntactic structure in the sequence to the nodes in $\Delta_Z(L)$.

Lemma 5.3.2. *For a node $z \in Z$ and $L = [U_\alpha, U_\beta, U_1, ..., U_\ell]$, $z \in \Delta_Z(L)$ if and only if there exists a subsequence U_i, U_{i+1}, U_{i+2} of L such that*

$$z \in \texttt{find\_unblocked\_ancestors\_in\_Z\_contributors}(\mathcal{G}, Z, A_Z, S),$$

where $S :=$ $\texttt{unblocked\_ancestors\_that\_changed\_A\_to\_B}(\mathcal{V}, U_i, U_{i+1})$. *In words, z was affected by changes between U_i and U_{i+1}.*

Proof. In the mechanized proof, we would proceed via induction on ℓ. However, it is clear from the Coq function definitions that the statement is true. □

This lemma isolates the part of the sequence that causes a particular z to change. It is possible for this change to come directly from U_β; if not, it is caused by reparative changes initiated by a different $z' \in Z$ in a previous transition. The following lemma sets up this recursive structure:

Lemma 5.3.3. *Let $L = [U_\alpha, U_\beta, U_1, ..., U_\ell]$ satisfy Condition 2 of Definition 3.2.6. Suppose U_i, U_{i+1}, U_{i+2} is the subsequence corresponding to $z \in \Delta_Z(L)$ as given by Lemma 5.3.2. Furthermore, suppose there exists U' such that U', U_i, U_{i+1} is a subsequence of L. Then, there exists a node a and a node $z' \in Z$ such that*

$$a \in \mathsf{Anc}^*_Z(z) \cap \mathsf{Anc}^*_Z(z')$$

and

$$z' \in \texttt{find\_unblocked\_ancestors\_in\_Z\_contributors}(\mathcal{G}, Z, A_Z, S'),$$

where $S' :=$ $\texttt{unblocked\_ancestors\_that\_changed\_A\_to\_B}(\mathcal{V}, U_{i-1}, U_i)$.

Proof. We perform induction on L. For the base case, suppose $U' = U_\alpha$, $U_i = U_\beta$, and $U_{i+1} = U_1$. Then, since z is affected by changes from U_β to U_1, there must be a node $a \in \mathsf{Anc}^*_Z(z)$ such that $U_\beta(a) \neq U_1(a)$. Then, there must be a node $z' \in Z$ such that $a \in \mathsf{Anc}^*_Z(z')$, and z' is affected by changes from U_α to U_β, as desired.

For the induction step, we assume that the result holds for $L' = [U_\beta, U_1, ..., U_\ell]$. If $U' = U_\alpha$, $U_i = U_\beta$, and $U_{i+1} = U_1$ again, we follow the same steps as above. If not, the result follows directly from the induction hypothesis. □

5.4 Concatenating Paths from Unobserved-Terms Assignments

Recall that our goal is to construct a d-connected path from u to v using the existence of a sequence of unobserved-terms assignments $U_\alpha, U_\beta, U_1, ..., U_\ell$ satisfying the conditions of Definition 3.2.6, under the assumption that $f_{U_\alpha}(v) \neq f_{U_\ell}(v)$.

The case where $\mathsf{Anc}^*_Z(u) \cap \mathsf{Anc}^*_Z(v) \neq \emptyset$ yields a d-connected path via Lemma 5.2.1. In the case where $\mathsf{Anc}^*_Z(u) \cap \mathsf{Anc}^*_Z(v) = \emptyset$, Lemma 5.3.1 guarantees the existence of some $z \in Z$ and $a \in \mathsf{Anc}^*_Z(z) \cap \mathsf{Anc}^*_Z(v)$ such that z is affected by changes in the sequence. If z is influenced directly by U_β, we can construct a d-connected path from u to z. Otherwise, Lemma 5.3.3 provides a recursive structure in which the change to z propagates through another node $z' \in Z$. Repeated applications of Lemmas 5.3.2 and 5.3.3, together with induction, allow us to construct a d-connected path from u to z. Finally, we concatenate this path with a d-connected path from z to v.

We now formalize the construction of a d-connected path from u to such a node $z \in Z$. Note that we will eventually have to concatenate this path with a d-connected path from z to v, in which z will have to be a collider to guarantee d-connectedness, as shown in Theorem 2.3.1. Thus, we require the additional constraint that the final edge is into z.

Lemma 5.4.1. *Let $z \in Z$, $z \in \Delta_Z(U_\alpha, U_\beta, U_1, ..., U_\ell)$. Then, there is a path $P_{u,z}$ from u to z that is acyclic, d-connected given Z, and into z at the last edge.*

Proof. By Lemma 5.3.2, there must be a subsequence U', U'', U''' of $U_\alpha, U_\beta, U_1, ..., U_\ell$ such that z was affected by changes between U' and U''. Let i be the index of the subsequence in the sequence (the i-th entry of the sequence is U', the $(i+1)$-th is U'', and the $(i+2)$-th is U'''). Note further that this index does not have to be unique; we do not require that the sequence of unobserved-terms assignments are minimal. We can choose any index i that satisfies the requirements.

Now, we proceed via strong induction on i. For the base case, suppose that $i = 0$, so $U' = U_\alpha, U'' = U_\beta, U''' = U_1$. Then, there must exist some a such that $a \in \mathsf{Anc}^*_Z(z)$, and $U_\alpha(a) \neq U_\beta(a)$. Thus, $a \in \mathsf{Anc}^*_Z(u)$. Since $z \in Z$, one of Conditions 1 or 3 of Lemma 5.2.1 must be true. Either way, it is clear that the path goes into z, is d-connected, and is acyclic.

For the induction hypothesis, suppose that for any $z' \in Z$ that is affected by changes between assignments appearing at index at most some n, where $n \geq 0$, there is a path $P_{u,z'}$

from u to z' that is acyclic, d-connected given Z, and into z'. Assume $i = n + 1 > 0$. Then, there must be some U that immediately precedes the subsequence U', U'', U'''; in other words, U appears at index $(i-1)$ in the sequence. Then by Lemma 5.3.3, there is a $z' \in Z$ and a node a such that $a \in \mathsf{Anc}^*_Z(z) \cap \mathsf{Anc}^*_Z(z')$, and z is affected by changes between U and U'. It is clear that $(i-1)$ is a valid index of the subsequence U, U', U'' in the sequence. Thus, we apply the induction hypothesis to get a path $P_{u,z'}$ that goes into z'.

Furthermore, since $z, z' \in Z$, Condition 3 of Lemma 5.2.1 must be true, giving us some single-confounder path $P_{z',z}$ from z' to z. We aim to concatenate these two paths; however, we must consider the possibility that they intersect at some point. Luckily, the casework here is much easier than in Section 4.2.

If $P_{u,z'}$ and $P_{z',z}$ do not intersect, we concatenate them directly. Since the last edge of $P_{u,z'}$ is into z', z' is a collider, and the path remains d-connected because $z' \in Z$.

If u lies on $P_{z',z}$, then we simply take the subpath of $P_{z',z}$ starting from u and continuing to z. Since $P_{z',z}$ is acyclic, d-connected, and into z, so is this subpath. If z lies on $P_{u,z'}$, then we simply take the subpath of $P_{u,z'}$ starting from u and continuing to z. This subpath must be acyclic and d-connected. Since $z \in Z$, and $P_{u,z'}$ is d-connected, z must be a collider in the path. Thus, the subpath is also into z.

Otherwise, let x be the first intersection point from Theorem 2.3.3. Construct the new path by taking $P_{u,z'}$ to x, then continuing along $P_{z',z}$. Let this composite path be $P_{u,z}$. This path is acyclic by Theorem 2.3.3 and still into z. Note that since $x \in P_{z',z}$, it must be a mediator or confounder in $P_{z',z}$ and thus not in Z. In $P_{u,z}$, if x is a mediator or confounder, then the path is d-connected, since $x \notin Z$. If x is a collider in $P_{u,z}$, then we consider where x falls in $P_{z',z}$. In particular, if x falls before the confounder, then x has a descendant path to z'. Otherwise, x has a descendant path to z. Either way, $P_{u,z}$ is d-connected.

All the above cases result in paths $P_{u,z}$ satisfying the requirements in the lemma statement. □

We now have all the pieces to proceed with the proof of Lemma 5.1.

Lemma 5.1. For a causal model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two different nodes $u, v \in \mathcal{V}$, and a conditioning set $Z \subseteq \mathcal{V}$ with $u, v \notin Z$, if u and v are d-separated given Z, then they are conditionally independent.

Proof. We prove the contrapositive. Suppose that u and v are not conditionally independent, so there exists some graph function f, some $\alpha \neq \beta$, and a sequence of unobserved-terms assignments U_α, U_β, U_1, ..., U_ℓ, where $\ell \geq 0$ satisfying the conditions of Definition 3.2.6, such that $f_{U_\alpha}(v) \neq f_{U_\ell}(v)$. We will show that u and v are not d-separated given Z. We follow the steps outlined at the beginning of this section.

By Lemma 5.3.1, there exists an $a \in \mathsf{Anc}^*_Z(v)$ such that either $a \in \mathsf{Anc}^*_Z(u)$, or there exists $z \in Z$ such that $a \in \mathsf{Anc}^*_Z(z)$ and $z \in \Delta_Z(U_\alpha, U_\beta, U_1, ..., U_\ell)$. Consider the first case. Then, by Lemma 5.2.1, there is a d-connected path from u to v, and thus u and v are not d-separated.

In the second case, we know by Lemma 5.4.1 that there is a d-connected, acyclic path $P_{u,z}$ from u to z that goes into z at the last edge. Furthermore, since z and v share unblocked

ancestor a, and $z \in Z$, one of Conditions 2 of 3 of Lemma 5.2.1 must hold, and thus there is a d-connected path $P_{z,v}$ from z to v that goes into z at the first edge.

We now consider the structure of the path from z to v and how it intersects with the previously constructed path from u to z.

1. If $P_{u,z}$ and $P_{z,v}$ do not intersect anywhere, then we can simply concatenate them to get $P_{u,v}$. Since $P_{u,z}$ is into z, z becomes a collider in $P_{u,v}$, so $P_{u,v}$ is d-connected.

2. If u lies on $P_{z,v}$ or v lies on $P_{u,z}$, we take the appropriate subpath, which remains a d-connected path.

3. If the first overlap of $P_{u,v}$ and the reverse of $P_{z,v}$ given by Theorem 2.3.3 is some node $x \notin \{u, v\}$, then we let $P_{u,v}$ be the path $P_{u,z}$ until x, then switch to the path $P_{z,v}$ until v. This path is acyclic by Theorem 2.3.3. Since $x \in P_{z,v}$, it is a mediator or confounder and thus not in Z. Thus if x is a mediator or confounder in $P_{u,v}$, then $P_{u,v}$ is still d-connected. Suppose x is a collider in $P_{u,v}$.

   Consider the structure of $P_{z,v}$ as given by Lemma 5.2.1 (it must satisfy Condition 2 or Condition 3). If $P_{z,v}$ is a directed path from v to z, then the descendant path from x to z along the reverse of $P_{z,v}$ ensures that $P_{u,v}$ is d-connected.

   If $P_{z,v}$ is a single-confounder path from z to v, then x must come before the confounder in $P_{z,v}$, since otherwise, the edge on the right of x in $P_{u,v}$ would be outwards, and thus x would not be a collider. Thus, the descendant path from x to z along the reverse of $P_{z,v}$ still ensures that $P_{u,v}$ is d-connected.

Thus, in all possible overlapping cases, there is a path $P_{u,v}$ from u to v that is d-connected given Z, so u and v are not d-separated, as desired. □

We now can establish the equivalence of conditional independence and d-separation given Z.

Theorem 3.3.1. *The notions of conditional independence 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 conditionally independent given Z if and only if they are d-separated given Z in $\mathcal{G}$.*

Proof. Follows directly from Lemmas 4.1 and 5.1. □

Chapter 6

Future Work

While this thesis establishes a foundational framework for reasoning about causal models in Coq, it also opens the door to many avenues of further exploration. This chapter outlines promising directions for future development, ranging from completing correctness proofs for existing functions to applying the framework to more advanced ideas in causal inference. In particular, the final sections explore how formal semantics can inform counterfactual reasoning, probabilistic extensions, and ultimately the formal verification of experimental-design validity.

6.1 Finish Correctness Proofs

There are still graph-theoretic functions that have not yet had their correctness fully proven due to the implementation complexity introduced by Coq's purely functional nature. In particular, functions that involve recursion over graph structures, such as pathfinding or topological sorting, require several intermediate steps and auxiliary lemmas to manage both termination and logical soundness.

One major proof target is the correctness of the `find_all_paths_from_start_to_end` function, which is central to reasoning about causal models. The correctness specification is stated below:

```
Theorem paths_start_to_end_correct: forall (p: path) (u v: node) (G: graph),
    (is_path_in_graph p G = true) /\ (path_start_and_end p u v = true)
                                  /\ (acyclic_path p = true)
    <-> In p (find_all_paths_from_start_to_end u v G).
```

Proving this theorem likely requires decomposing the implementation into a series of lemmas that correspond to each step in the pathfinding pipeline, as described in Section 2.1.1. Analogous reasoning can then be reused to prove the correctness of the similar functions for finding directed paths, finding descendants and ancestors, and cycle detection.

The other important unproven result is the topological-sort correctness theorem. While we have already proven the existence of a topological sort for acyclic graphs and shown many key properties, we have not yet proven the positional constraints between nodes:

```
Theorem topo_sort_correct: forall (G: graph) (u v: node) (sorted: nodes),
  G_well_formed G = true /\ topological_sort G = Some sorted
                         /\ edge_in_graph (u, v) G = true
  -> exists (i j: nat),
       Some i = index sorted u /\ Some j = index sorted v /\ i < j.
```

This theorem states that any topological sort must respect the edge directionality of the input graph. While its truth is well-known in the field of algorithms, proving it in Coq will require careful management of the graph and its edges, serving as a valuable exercise in formal algorithmic reasoning.

Importantly, although the correctness proofs for these functions are not yet complete, their absence does not threaten the soundness of the overall system; the existence of algorithms satisfying these theorems is already established in classical theory, so we are confident that the specifications of the theorems themselves are accurate. If anything were to fail in proof, it would more likely reveal an issue with the implementation rather than with the theorem specification. Nonetheless, completing these correctness proofs is an essential step toward building a fully verified system on trustworthy foundational components.

6.2 Modeling Counterfactuals

Counterfactuals play a crucial role in causal inference, allowing us to explore hypothetical scenarios and understand the consequences of different actions. As discussed in Section 1.2.1, the twin-network model is intuitive and aligns closely with the traditional DAG representation of causal models, but it has a difficult preprocessing step that was introduced later on to overcome an error that resulted from the model failing to account for deterministic dependencies between twin nodes. The original error could have been avoided with a formalization, which would have enforced greater rigor in the development of the correct twin network and the checks for d-separation. In addition, the formalization would be useful to understand the conceptual difficulties that come with the preprocessing step fully.

We have implemented the twin-network method up to just before the preprocessing step, which has formally replicated the error in Example 11.3.3 of *Causality* [3]. The graph involved in the example was the model of sequential randomness shown in Figure 1.2.1. The conditional independence in question was between X_1 and Y^* conditioned on Z and $X_0 = x_0^*$. In the twin network shown in Figure 1.2.2a, the path $X_1 \leftarrow H \leftarrow U_H \rightarrow H^* \rightarrow Z^* \rightarrow Y^*$ d-connects X_1 to Y^*, which led to the conclusion that the two nodes are not independent. However, Z^* is deterministically related to Z. Shpitser and Pearl describe in detail the algorithm for determining this relation [6], but intuitively, Z^* depends only on H^* and x_0^*; $X_0 = x_0^*$ since we assume that the naturally occurring value of X_0 is x_0^*, and H and H^* are also equivalent, since they represent the same value and are both only affected by U_H. Figure 1.2.2b shows the correct twin network post-preprocessing.

This error has now been verified within Coq:

```
Example sequential_twin_network_error:
    d_separated_bool X1 Y' sequential_twin [Z;X0] = false.
```

where `Y'` is Y^*.

Additionally, we have formally proven that the conditional-independence relationships remain identical in the duplicated twin graph before any unobserved confounders are added. In the process, we also proved that many other causal-model properties, such as colliders, descendants, and paths, are preserved in the twin graph. Thus, we have validated the correctness of the initial stages of the formalization.

The next step is to implement and verify the preprocessing step of the twin-network model in Coq, which would address the dependency issue by merging specific twin nodes that share deterministic relationships [6]. This preprocessing step is critical, as it would have prevented the replicated error from occurring in the first place. The implementation would ensure that the twin-network model can accurately represent counterfactual scenarios without ambiguity. Determining the appropriate notion of correctness for this algorithm and completing its proof would make the natural and intuitive twin-network model also formally sound and robust.

6.3 Inducing Paths and *d*-Separation

As described in Section 1.2.2, inducing paths are a powerful concept in causal modeling, particularly when dealing with latent variables, which are variables that are not directly observed but may still influence the relationships between observed variables. Theorem 1.2.2 states an equivalence between the existence of an inducing path over a subset O and d-connectedness of a and b conditioned on every subset of $O \setminus \{a, b\}$. This equivalence is important because it links structural properties of causal graphs (inducing paths) with the theoretical property of conditional independence (d-separation).

In our Coq formalization, Theorem 1.2.2 takes the form:

```
Theorem d_separation_and_inducing_paths:
    forall (G: graph) (O: nodes) (a b: node),
    contains_cycle G = false ->
    (forall Z: nodes, subset Z (set_subtract O [a; b]) = true
        -> d_separated_bool a b G Z = false)
    <-> exists (U: path), path_start_and_end U a b = true
        /\ inducing_path U G O.
```

The forward direction, showing that if a and b are d-connected given all subsets of $O \setminus \{a, b\}$, then there exists an inducing path between them, has already been proven. This proof leverages numerous auxiliary lemmas about acyclic graphs and set-theoretic operations.

An important next step is to complete the backward direction of the theorem. The proof strategy breaks into two main cases depending on whether the inducing path begins with an arrow out of a or into a, corresponding to whether the resulting d-connected path is out of or into a. In the former, we must reason about the structure of colliders along the path. If

no colliders exist, then the path is already d-connected regardless of the conditioning set. If colliders do appear, a more nuanced analysis is required. In particular, the key fact used in the proof, originally stated without elaboration, is the following: in a DAG, every node in a path is either an ancestor of one of the path's endpoints or an ancestor of a collider [9].

While this claim appears nontrivial at first glance, working through the semantics of d-connectedness in our framework makes its intuition much clearer. Specifically, using the notation of Chapter 4, all nodes on a d-connected path fall into one of the sets S_1, S_2, or S_3: sources, transmitters, or colliders, respectively. If the node is past the first source and before the last source, then it must lie on a path to or from a collider (i.e., be an ancestor of one). Otherwise, it must be an ancestor of an endpoint. Thus, the semantic lens developed in this thesis provides a natural and intuitive route toward formalizing this result, an insight that would likely have remained opaque without the semantic framework.

Formalizing this observation and using it to complete the backward proof of Theorem 1.2.2 would solidify the connection between semantic reasoning and classical graphical notions involving latent variables and independence. It would also pave the way for mechanized reasoning about maximal ancestral graphs and other advanced representations that rely on inducing paths as a central concept.

6.4 More Semantic Equivalences

The central contribution of this thesis is the development of a semantic definition of conditional independence and the formal proof of its equivalence with the classical syntactic definition of d-separation. This result is not just a correctness theorem; it provides a new lens through which to understand what d-separation actually means by growing it from a purely graph-theoretic rule into a semantically rich property of how node values respond to changes in other nodes under constraints. It also opens the door to extending the same semantic treatment to other syntactic constructs in causal inference.

One natural next target is the `do` operator. Syntactically, applying $\texttt{do}(a := \alpha)$ to a graph means removing all incoming edges to node a, thus cutting it off from its usual causal influences. Semantically, however, this operation should correspond to a transformation of the graph function; rather than computing a from its parents and unobserved term, the function for a should be replaced with the constant function $f(a) := \alpha$. We conjecture that these two views of intervention (syntactic graph surgery vs. semantic value override) are equivalent and that this equivalence can be proven formally in Coq.

Another area where a semantic view would be especially useful is counterfactual reasoning, a more subtle and error-prone component of causal inference. The standard approach using twin networks can be difficult to interpret and reason about. In this thesis, we identified specific ambiguities in the preprocessing step of the model, which led to errors even in canonical sources such as *Causality* [3]. A semantic framing could describe counterfactual worlds through modified graph functions and value propagations. The axioms of effectiveness, composition, and consistency [13] map well onto the function-based semantics already built

in this framework. In a formal equivalence between semantic counterfactuals and twin-network–based reasoning, the preprocessing step in the twin-network model could be explained not merely as a technical fix but as a semantic necessity to model dependencies between twin nodes correctly.

One striking aspect of causal models is that much of their expressive power does not rely on probabilities at all. Our formalization, for example, says nothing about probabilities yet supports many parts of causal reasoning. This abstraction is powerful, especially given that real-world experiments are inherently probabilistic and rarely yield deterministic conclusions. However, probabilities remain a critical part of causal inference, particularly for evaluating confidence and estimating causal effects. An extension of this work would incorporate probability distributions into the formal framework, allowing us to reason about probabilistic core results such as the backdoor criterion, frontdoor criterion, and rules of do-calculus. For example, proving that the backdoor adjustment formula holds under a semantic definition of the backdoor criterion would provide a compelling synthesis of probabilistic reasoning and formal verification.

Establishing these connections would further validate the semantic framework and allow us to improve the interpretability of causal inference tools.

6.5 Formally Verifying Experimental-Design Validity

This thesis is the first research thrust in a longer-term project to develop a formal framework for verifying the validity of experimental designs. We believe that this is a promising but underexplored area; while experimental science produces results at an accelerating pace, the infrastructure for rigorously validating those results across disciplines has not kept up. Establishing a mechanized way of validating experimental reasoning is crucial to addressing this gap.

A key insight motivating this direction is that causal models can inform the validity of experimental designs. They provide structure that can help identify key variables to measure, detect potential confounders, and guide decisions on how to control for biases, ensuring that the observed effects are due to the treatment or intervention rather than other factors. They can also help researchers design experiments that can isolate causal effects, predict the outcomes of different interventions, and improve the robustness and generalizability of findings.

This direction of work aligns closely with another ongoing research effort: PLanet [16], a domain-specific language for formalizing experimental design that generates assignment plans for experiments given user constraints. The language supports complex patterns such as counterbalancing and Latin squares, enabling researchers to specify experimental structures. Combining PLanet's design-specification capabilities with our semantic causal-modeling framework could lead to powerful new verification tools. For instance, we might formally prove that counterbalancing removes a confounding influence by showing that, semantically, the design transformation results in two variables becoming conditionally independent. That is,

experimental design would be understood not only in terms of assignments and randomization but also in terms of the causal model it induces, verified formally in Coq to ensure that experimental intent matches causal-inference validity.

Ultimately, this work lays the groundwork for building a full-stack formal system handling experimental-design specification, causal-model generation, and formally proven analysis of causal claims. This level of rigor could greatly enhance reproducibility, interpretability, and validity in experimental science.

References

[1] The Coq Development Team. *The Coq Proof Assistant*. Version 8.18. 2024. URL: http://coq.inria.fr.

[2] J. Pearl and D. Mackenzie. *The Book of Why: The New Science of Cause and Effect*. 1st. USA: Basic Books, Inc., 2018. ISBN: 046509760X.

[3] J. Pearl. *Causality*. 2nd ed. Cambridge University Press, 2009.

[4] A. Balke and J. Pearl. "Probabilistic Evaluation of Counterfactual Queries". In: Feb. 2022, pp. 237–254. ISBN: 9781450395861. DOI: 10.1145/3501714.3501733.

[5] A. Naimi, S. Cole, and E. Kennedy. "An Introduction to G Methods". In: *International journal of epidemiology* 46 (Dec. 2016). DOI: 10.1093/ije/dyw323.

[6] I. Shpitser and J. Pearl. "Complete Identification Methods for the Causal Hierarchy". In: *J. Mach. Learn. Res.* 9 (June 2008), pp. 1941–1979. ISSN: 1532-4435.

[7] C. Hazlett and A. Rohde. "Shadow and Within-Stratum Graphs: Tools for Teaching DAGs and Potential Outcomes Together". Work in progress, not yet submitted. 2025.

[8] T. S. Richardson and J. M. Robins. "Single World Intervention Graphs : A Primer". In: *Second UAI Workshop on Causal Structure Learning* (2013).

[9] P. Spirtes. "Building causal graphs from statistical data in the presence of latent variables". In: *Logic, Methodology and Philosophy of Science IX*. Ed. by D. Prawitz, B. Skyrms, and D. Westerståhl. Vol. 134. Studies in Logic and the Foundations of Mathematics. Elsevier, 1995, pp. 813–829. DOI: https://doi.org/10.1016/S0049-237X(06)80075-3. URL: https://www.sciencedirect.com/science/article/pii/S0049237X06800753.

[10] J. Zhang. "Causal Reasoning with Ancestral Graphs". In: *J. Mach. Learn. Res.* 9 (June 2008), pp. 1437–1474. ISSN: 1532-4435.

[11] E. Bareinboim, J. D. Correa, and C. Jeong. *Causal Fusion*. URL: https://causalfusion.net/app.

[12] E. Bareinboim and J. Pearl. "Causal inference and the data-fusion problem". In: *Proceedings of the National Academy of Sciences* 113.27 (2016), pp. 7345–7352. DOI: 10.1073/pnas.1510507113. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1510507113. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1510507113.

[13] J. Tian and J. Pearl. *Probabilities of Causation: Bounds and Identification*. 2013. arXiv: 1301.3898 [cs.AI]. URL: https://arxiv.org/abs/1301.3898.

[14] S. Lau, T. Bourgeat, C. Pit-Claudel, and A. Chlipala. "Specification and Verification of Strong Timing Isolation of Hardware Enclaves". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 1121–1135. ISBN: 9798400706363. DOI: 10.1145/3658644.3690203. URL: https://doi.org/10.1145/3658644.3690203.

[15] J. Pearl. "Causal inference in statistics: An overview". In: *Statistics Surveys* 3 (2009), pp. 96–146. DOI: 10.1214/09-SS057. URL: https://doi.org/10.1214/09-SS057.

[16] L. Bielicke, A. Zhang, S. Tyagi, E. Berger, A. Chlipala, and E. Jun. *PLanet: Formalizing Experimental Design*. 2025. arXiv: 2505.09094 [cs.HC]. URL: https://arxiv.org/abs/2505.09094.