Towards an equational calculus of interventions

Shubh Agrawal
Northeastern Universtiy
Boston, MA, USA
agrawal.shub@northeastern.edu

Jialu Bao
Northeastern Universtiy
Boston, MA, USA
jialu8ao@gmail.com

Steven Holtzen
Northeastern Universtiy
Boston, MA, USA
s.holtzen@northeastern.edu

Abstract

A key aspect of understanding complex probabilistic phenomena is determining the *causal effect* that one random quantity has on another, i.e., determining the extent to which setting the value of one random quantity causes the value of another to change. The notion of "setting the value" of a random variable is known as an *intervention*. A key reasoning challenge in this setting is that practical causal models often have *latent variables* and *unknown mechanisms*: hidden factors influence the observable variables in some way, but the values of the latent variables and the precise causal relationships are unknown. It might seem that reasoning about causal effects with these unknowns is impossible, but Pearl [5] showed that it can be done by *rewriting the model* into one that no longer depends on the hidden latent variables. In this work, we seek (1) a programming model that enables programmers to express models with latent variables in them; and (2) a formal equational description of a rewriting theory that enables programmers to confidently perform causal reasoning.

1 Motivation: valid observational studies

A classic example of causal reasoning due to Pearl [5] goes as follows: suppose we want to know the causal effect of smoking on having cancer. Answering such questions is relatively straightforward if one has a complete description of the relationship between the potential causes of cancer as a probabilistic program, like this:

Genotype $\leftarrow$ **flip** 0.3;
Smokes $\leftarrow$ **flip** (**if** Genotype **then** 0.4 **else** 0.7)
Tar $\leftarrow$ **flip** (**if** Smokes **then** 0.6 **else** 0.1)
Cancer $\leftarrow$ **return** Genotype $\vee$ Tar;
return $\langle$Genotype, Smokes, Tar, Cancer$\rangle$

Given such a program, we can easily determine what effect it would have on Cancer if we set the value of Smokes by performing "surgery" to convert it to the form we want:

Genotype $\leftarrow$ **flip** 0.3;
Smokes $\leftarrow$ **return** $\hat{s}$
Tar $\leftarrow$ **flip** (**if** Smokes **then** 0.8 **else** 0.3)
Cancer $\leftarrow$ **return** Genotype $\vee$ Tar;
return Cancer

Above we performed two operations: first, we replaced the expression that computed Smokes with free variable $\hat{s}$. Then, we "projected" the fourth component of the result to obtain the probability of Cancer in the intervened-on model. To determine the strength of the causal effect of Smokes on Cancer, we run inference twice: once where we substitute **true** for $\hat{s}$, and once where we substitute **false**. Then, we compare the resulting distributions. In this situation, where the relevant relationships between variables are all known, we call the causal effect *fully identifiable*.

Causal reasoning in the fully identifiable setting is well-studied and there are off-the-shelf tools to enable probabilistic programs to perform these reasoning tasks [6]. However, full identifiability is a very strong requirement that is often not fulfilled in practice.

More typically, one has some degree of partial information about the model: *output variables* are returned by the model, and modeler is assumed to have access to a joint probability distribution over these variables, while the remaining *latent variables* cannot be observed directly. We can make assumptions about the *abstract* relationships between the variables, but do not know the precise functional form of the relationship. For the smoking situation, that could be represented with a program like this, which we will denote $O()$:

Genotype $\leftarrow f_G()$;
Smokes $\leftarrow f_S$(Genotype);
Tar $\leftarrow f_T$(Smokes);
Cancer $\leftarrow f_C$(Genotype, Tar);
return $\langle$Smokes, Tar, Cancer$\rangle$

The relationships between variables are now abstract, uninterpreted functions f_G, f_S, f_T, and f_C. The new intervened-on program, denoted $\mathcal{I}(\hat{s})$, is this:

Genotype $\leftarrow f_G()$;
Smokes $\leftarrow$ **return** $\hat{s}$
Tar $\leftarrow f_T$(Smokes);
Cancer $\leftarrow f_C$(Genotype, Tar);
return Cancer

Our goal can now be stated precisely: to compute the strength of a causal effect in the presence of latent variables, we must infer the distribution $\mathcal{I}(\hat{s})$ using only the distribution $O()$ as a black box. To do this, we propose a rewriting theory for *rewriting* $\mathcal{I}(\hat{s})$ *into an identifiable program* – that

is, one which uses $O()$ as a subprogram and has no other abstract functions. The fact that such a program rewrite is possible is quite surprising, and the process of safely performing these rewrites can be quite complex. Our aim is to provide an equational theory for indentifying interventional queries in the context of probabilistic programs, similar to the *do-calculus* in the context of traditional, pen-and-paper probability reasoning [5], and existing equational rules in the context of categorical probability [3].

2 Equational reasoning for interventions

For specifying and computing the probabilistic models, we define a monadic probabilistic programming language. We then provide a system of equational rewriting rules for terms in the language. We use the rewriting rules to transform a program representing the intervened-on model, whose distribution cannot be computed directly, into a program that can be computed directly by using the observational study as a subprogram.

Our probabilistic programming language is mostly standard (the full definition is deferred to Appendix A). We highlight a few choices. First, the language is typed with Booleans, finite products and two monadic type constructors – the usual discrete subdistribution monad $\mathcal{SD}$ and the distribution monad $\mathcal{D}$. For any type τ, we have $\mathcal{D}\,\tau$ being a subtype of $\mathcal{SD}\,\tau$ – recall that a subdistribution $\mu : \mathrm{Val} \to [0,1]$ satisfies $\sum_{a \in \mathrm{Val}} \mu(a) \leq 1$ while a distribution μ sums to exactly 1. Second, the language can convert a term $e : \mathcal{SD}\,\tau$ into a term $\mathbf{norm}\,(e)$ of type $\mathcal{D}\,\tau$ by normalizing. In addition, the language provides $\mathbf{flip}\,(\theta)$ for introducing randomness and $\mathbf{observe}\,e_1 = e_2$ for unnormalized Bayesian conditioning. Third, the language also supports first-order functions in order to represent abstract relationships between variables.

Our rewriting theory consists of judgments of the form $\Gamma \vdash e_1 \equiv e_2 : \tau$. Here, we highlight two notable rules in Figure 1; we defer the rest to Appendix B. These rules are indirectly validated due to equations given by Jacobs et al. [3]; ongoing work is to further develop a denotational semantics for proving the soundness of our theory directly.

A crucial concept in probabilistic reasoning is *disintegration*, which breaks the process generating a joint distribution into two steps: $\Pr(y, z \mid x) = \Pr(y \mid x) \cdot \Pr(z \mid y, x)$. This principle is captured in our equational theory by rule Disintegration. A program e which computes a joint (sub)distribution over τ_1, τ_2 is equivalent to the program which runs e twice independently, conditions on their first components being equal, and returns the first and second component from the first and second runs, respectively.

The next rule is the crux of identifying interventional queries: Observation-Intervention-Exchange. This rule converts a computation e into the Dirac delta distribution on y', given as input; in order for this to be sound, we need that e has full support – this ensures that y' is indeed in the support of e so that the conditioned subdistribution is not everywhere-zero. The full support condition can be proved in a number of ways: equationally (using rule Full-Support), denotationally, or by assumption. While it may seem unintuitive, we typically use this rule from right-to-left. This allows us to convert an intervened-on program into one that involves conditioning over a program whose distribution we already know.

We use these two rules, and a number of others, to rewrite the program $\mathcal{I}(\hat{s})$ into one which uses $O()$ as a subprogram and no other unknown mechanisms. In order to fulfill the premise of Observation-Intervention-Exchange when we apply it, we need the additional assumption that $O()$ has full support. This is sufficient to prove that each internal abstract function also has full support. The full equivalence is given in Appendix C.

3 Discussion & related work

We have presented an outline for an equational theory for soundly reasoning about interventions in causal models with latent variables. These equational rules are heavily inspired by and build directly on the string diagram approach introduced by Jacobs et al. [3]. In our view, the main deficiency of this approach is that it uses very low-level manipulations that are difficult to understand. In the future, we aim to develop higher-level abstractions that build on this low-level equational theory that are more ergonomic to use, and to potentially explore automation.

We believe that our equational approach to causal reasoning improves on the state-of-the-art for causal reasoning in probabilistic programs by incorporating latent variables and unknown causal mechanisms. Prior work for interventional reasoning in probabilistic programming languages focuses primarily on the fully identifiable setting [1, 2, 4, 6, 7]. We hope that this equational theory can bring probabilistic programming languages one step closer to being a platform for general-purpose and verifiably correct causal reasoning.

References

[1] Ria Das, Joshua B Tenenbaum, Armando Solar-Lezama, and Zenna Tavares. 2023. Combining functional and automata synthesis to discover causal reactive programs. *Proceedings of the ACM on Programming Languages* 7, POPL (2023), 1628–1658.

[2] Duligur Ibeling and Thomas Icard. 2020. Probabilistic reasoning across the causal hierarchy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10170–10177.

[3] Bart Jacobs, Márk Széles, and Dario Stein. 2025. Compositional Inference for Bayesian Networks and Causality. *Mathematical Foundations of Programming Semantics* (2025).

[4] Rafael Kiesel, KILIAN RÜCKSCHLO, and Felix Weitkämper. 2023. "What if?" in Probabilistic Logic Programming. *Theory and Practice of Logic Programming* 23, 4 (2023), 884–899.

[5] Judea Pearl. 2009. *Causality.* Cambridge university press.

[6] Zenna Tavares, James Koppel, Xin Zhang, Ria Das, and Armando Solar-Lezama. 2021. A language for counterfactual generative models. In *International conference on machine learning*. PMLR, 10173–10182.

Disintegration

$$\frac{\Gamma \vdash e : \mathcal{SD}\,(\tau_1 \times \tau_2)}{\Gamma \vdash e \equiv \begin{pmatrix} \langle y_1, \_ \rangle \leftarrow e; \\ z_2 \leftarrow \mathbf{norm}\,(\langle y_2, z_2 \rangle \leftarrow e; \_ \leftarrow \mathbf{observe}\, y_1 = y_2; \mathbf{return}\, z_2)\,; \\ \mathbf{return}\, \langle y_1, z_2 \rangle \end{pmatrix} : \mathcal{SD}\,(\tau_1 \times \tau_2)}$$

Observation-Intervention-Exchange

$$\frac{x : \tau_1 \vdash e : \mathcal{D}\,\tau_2 \qquad e\ \mathbf{full\ support}}{x : \tau_1, y' : \tau_2 \vdash \mathbf{norm}\,(y \leftarrow e; \_ \leftarrow \mathbf{observe}\, y = y'; \mathbf{return}\, y) \equiv \mathbf{return}\, y' : \mathcal{D}\,\tau_2}$$

Figure 1. Selected equational rules

[7] Sam Witty, Alexander Lew, David Jensen, and Vikash Mansinghka. 2019. Bayesian causal inference via probabilistic program synthesis. *arXiv preprint arXiv:1910.14124* (2019).

A Language Definition and Typing Rules

$$
\begin{aligned}
\text{Expr} \ni e ::= {} & x \mid f(e) \mid \mathbf{return}\, e \mid x \leftarrow e_1;\ e_2 \\
& \mid \mathbf{observe}\, e_1 = e_2 \mid \mathbf{norm}\,(e) \mid \mathbf{flip}\,\theta \\
& \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3 \\
& \mid \langle\rangle \mid \langle e_1, e_2\rangle \mid \pi_1\, e \mid \pi_2\, e \\
\text{Type} \ni \tau ::= {} & \mathbb{B} \mid \mathcal{SD}\,\tau \mid \mathcal{D}\,\tau \mid \tau_1 \times \tau_2 \mid \mathbb{1} \\
\text{ValCtx} \ni \Gamma ::= {} & \cdot \mid \Gamma,\ x : \tau \\
\text{FnCtx} \ni \Sigma ::= {} & \cdot \mid \Sigma,\ f : \tau_1 \to \tau_2
\end{aligned}
$$

Figure 2. Language grammar

$$\frac{x : \tau \in \Gamma}{\Sigma; \Gamma \vdash x : \tau} \qquad \frac{f : \tau_1 \to \tau_2 \in \Sigma \qquad \Sigma; \Gamma \vdash e : \tau_1}{\Sigma; \Gamma \vdash f(e) : \tau_2} \qquad \frac{\Sigma; \Gamma \vdash e : \tau}{\Sigma; \Gamma \vdash \mathbf{return}\, e : \mathcal{D}\,\tau} \qquad \frac{\Sigma; \Gamma \vdash e_1 : \mathcal{SD}\,\tau_1 \qquad \Sigma; \Gamma, x : \tau_1 \vdash e_2 : \mathcal{SD}\,\tau_2}{\Sigma; \Gamma \vdash x \leftarrow e_1;\ e_2 : \mathcal{SD}\,\tau_2}$$

$$\frac{\Sigma; \Gamma \vdash e_1 : \mathcal{D}\,\tau_1 \qquad \Sigma; \Gamma, x : \tau_1 \vdash e_2 : \mathcal{D}\,\tau_2}{\Sigma; \Gamma \vdash x \leftarrow e_1;\ e_2 : \mathcal{D}\,\tau_2} \qquad \frac{\Sigma; \Gamma \vdash e_1 : \tau \qquad \Sigma; \Gamma \vdash e_2 : \tau}{\Sigma; \Gamma \vdash \mathbf{observe}\, e_1 = e_2 : \mathcal{SD}\,\mathbb{1}} \qquad \frac{\Sigma; \Gamma \vdash e : \mathcal{SD}\,\tau}{\Sigma; \Gamma \vdash \mathbf{norm}\,(e) : \mathcal{D}\,\tau} \qquad \frac{\Sigma; \Gamma \vdash e : \mathcal{D}\,\tau}{\Sigma; \Gamma \vdash e : \mathcal{SD}\,\tau}$$

$$\frac{}{\Sigma; \Gamma \vdash \mathbf{flip}\,\theta : \mathcal{D}\,\mathbb{B}} \qquad \frac{}{\Sigma; \Gamma \vdash \mathbf{true} : \mathbb{B}} \qquad \frac{}{\Sigma; \Gamma \vdash \mathbf{false} : \mathbb{B}} \qquad \frac{\Sigma; \Gamma \vdash e_1 : \mathbb{B} \qquad \Sigma; \Gamma \vdash e_2 : \tau \qquad \Sigma; \Gamma \vdash e_3 : \tau}{\Sigma; \Gamma \vdash \mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3 : \tau}$$

$$\frac{}{\Sigma; \Gamma \vdash \langle\rangle : \mathbb{1}} \qquad \frac{\Sigma; \Gamma \vdash e_1 : \tau_1 \qquad \Sigma; \Gamma \vdash e_2 : \tau_2}{\Sigma; \Gamma \vdash \langle e_1, e_2\rangle : \tau_1 \times \tau_2} \qquad \frac{\Sigma; \Gamma \vdash e : \tau_1 \times \tau_2}{\Sigma; \Gamma \vdash \pi_1\, e : \tau_1} \qquad \frac{\Sigma; \Gamma \vdash e : \tau_1 \times \tau_2}{\Sigma; \Gamma \vdash \pi_2\, e : \tau_2}$$

Figure 3. Typing rules

B Equational Theory

The equational theory has rules of the form $\Gamma \vdash e_1 \equiv e_2 : \tau$, presented in Figure 4.

Our first two rules, Monad-Unit-L and Monad-Unit-R are standard for monadic computations. Swap can be complex in general, but we provide the simple case where the subexpressions clearly cannot refer to one another. Drop-L and Drop-R allow adding or removing superfluous computations when they are proper distributions; this is to ensure that the added computation does not re-weight the resulting subdistribution. Norm-Bind-Dist and Norm-Bind-Eqn says that normalization respects monadic bind when the outer computation returns a proper distribution. The former proves the premise via the typing rules, and the latter equationally. Norm-Pair says that normalization respects products. Norm-Idempotent says that normalizing twice is the same as normalizing once. Observe-Swap intuitively captures the idea that observe has a global effect; as such, observing before or after a computation is equivalent. Finally, Full-Support provides the equational definition for proving that an expression has full support.

$$\textsc{Monad-Unit-L}\quad \frac{\Gamma \vdash y : \tau' \qquad \Gamma, x : \tau' \vdash e : \mathcal{SD}\,\tau}{\Gamma \vdash x \leftarrow \mathbf{return}\, y; e \equiv e[y/x] : \mathcal{SD}\,\tau} \qquad \textsc{Monad-Unit-R}\quad \frac{\Gamma \vdash e : \mathcal{SD}\,\tau}{\Gamma \vdash x \leftarrow e; \mathbf{return}\, x \equiv e : \mathcal{SD}\,\tau}$$

$$\textsc{Swap}\quad \frac{\Gamma \vdash e_1 : \mathcal{SD}\,\tau_1 \qquad y \notin \Gamma \qquad \Delta \vdash e_2 : \mathcal{SD}\,\tau_2 \qquad x \notin \Delta \qquad x \neq y}{\Gamma, \Delta \vdash x \leftarrow e_1; y \leftarrow e_2; \mathbf{return}\,(x, y) \equiv y \leftarrow e_2; x \leftarrow e_1; \mathbf{return}\,(x, y) : \mathcal{SD}\,(\tau_1 \times \tau_2)}$$

$$\textsc{Drop-L}\quad \frac{\Gamma \vdash e_1 : \mathcal{SD}\,\tau_1 \qquad \Gamma \vdash e_2 : \mathcal{D}\,\tau_2 \qquad y \notin \Gamma}{\Gamma \vdash e_1 \equiv y \leftarrow e_2; x \leftarrow e_1; \mathbf{return}\, x : \mathcal{SD}\,\tau_1} \qquad \textsc{Drop-R}\quad \frac{\Gamma \vdash e_1 : \mathcal{SD}\,\tau_1 \qquad \Gamma \vdash e_2 : \mathcal{D}\,\tau_2 \qquad x \neq y}{\Gamma \vdash e_1 \equiv x \leftarrow e_1; y \leftarrow e_2; \mathbf{return}\, x : \mathcal{SD}\,\tau_1}$$

$$\textsc{Norm-Bind-Dist}\quad \frac{\Gamma, x : \tau_1 \vdash e_2 : \mathcal{D}\,\tau_2}{\Gamma \vdash \mathbf{norm}\,(x \leftarrow e_1;\ e_2) \equiv x \leftarrow \mathbf{norm}\,(e_1);\ e_2 : \mathcal{D}\,\tau_2} \qquad \textsc{Norm-Bind-Eqn}\quad \frac{\Gamma, x : \tau_1 \vdash y \leftarrow e_2;\ \mathbf{return}\,\langle\rangle \equiv \mathbf{return}\,\langle\rangle : \mathcal{SD}\,\tau_2}{\Gamma \vdash \mathbf{norm}\,(x \leftarrow e_1;\ e_2) \equiv x \leftarrow \mathbf{norm}\,(e_1);\ e_2 : \mathcal{D}\,\tau_2}$$

$$\textsc{Norm-Pair}\quad \frac{}{\Gamma \vdash \mathbf{norm}\,(x \leftarrow e_1;\ y \leftarrow e_2;\ \mathbf{return}\,\langle x, y\rangle) \equiv x \leftarrow \mathbf{norm}\,(e_1);\ y \leftarrow \mathbf{norm}\,(e_2);\ \mathbf{return}\,\langle x, y\rangle : \mathcal{D}\,(\tau_1 \times \tau_2)}$$

$$\textsc{Norm-Idempotent}\quad \frac{\Gamma \vdash e_1 : \mathcal{SD}\,\tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \mathcal{SD}\,\tau_2}{\Gamma \vdash \mathbf{norm}\,(x \leftarrow e_1;\ \mathbf{norm}\,(e_2)) \equiv \mathbf{norm}\,(x \leftarrow e_1;\ e_2) : \mathcal{D}\,\tau_2}$$

$$\textsc{Observe-Swap}\quad \frac{x : \tau_1 \vdash e : \mathcal{SD}\,\tau_2}{x : \tau_1, x' : \tau_1 \vdash \_ \leftarrow \mathbf{observe}\, x = x'; e \equiv y \leftarrow e; \_ \leftarrow \mathbf{observe}\, x = x'; \mathbf{return}\, y : \mathcal{SD}\,\tau_2}$$

$$\textsc{Full-Support}\quad \frac{x : \tau_1, y' : \tau_2 \vdash \mathbf{norm}\,(y \leftarrow e; \mathbf{observe}\, y = y') \equiv \mathbf{return}\,\langle\rangle : \mathcal{D}\,\mathbb{1}}{e\ \mathbf{full\ support}}$$

Figure 4. Equivalences

C Smoking Example

Recall that our goal is to rewrite $\mathcal{I}(\hat{s})$ into a program which uses O as a subprogram and no other unknown functions. Throughout, we will freely apply substitutions.

First, we apply Monad-Unit-L:

$$\begin{aligned}\mathcal{I}(\hat{s}) \equiv\ & \text{Genotype} \leftarrow f_G();\\ & \text{Tar} \leftarrow f_T(\hat{s});\\ & \text{Cancer} \leftarrow f_C(\text{Genotype}, \text{Tar});\\ & \mathbf{return}\,\text{Cancer}\end{aligned}$$

Next, we apply Drop-R and Swap:

$$\begin{aligned}\mathcal{I}(\hat{s}) \equiv\ & \text{Genotype} \leftarrow f_G();\\ & \text{Smokes} \leftarrow f_S(\text{Genotype});\\ & \text{Tar} \leftarrow f_T(\hat{s});\\ & \text{Cancer} \leftarrow f_C(\text{Genotype}, \text{Tar});\\ & \mathbf{return}\,\text{Cancer}\end{aligned}$$

Next, we use Disintegration with $e := \text{Genotype} \leftarrow f_G(); \text{Smokes} \leftarrow f_S(\text{Genotype}); \textbf{return}\ \langle \text{Smokes}, \text{Genotype} \rangle$:

$$\begin{aligned}
\mathcal{I}(\hat{s}) \equiv\ & \text{Genotype} \leftarrow f_G(); \\
& \text{Smokes} \leftarrow f_S(\text{Genotype}); \\
& \text{Genotype}'' \leftarrow \textbf{norm} \begin{pmatrix} \text{Genotype}' \leftarrow f_G(); \\ \text{Smokes}' \leftarrow f_S(\text{Genotype}'); \\ _ \leftarrow \textbf{observe}\ \text{Smokes} = \text{Smokes}'; \\ \textbf{return}\ \text{Genotype}' \end{pmatrix} \\
& \text{Tar} \leftarrow f_T(\hat{s}); \\
& \text{Cancer} \leftarrow f_C(\text{Genotype}'', \text{Tar}); \\
& \textbf{return}\ \text{Cancer}
\end{aligned}$$

Next, we use Swap with $x := \text{Genotype}'', y := \text{Tar}$:

$$\begin{aligned}
\mathcal{I}(\hat{s}) \equiv\ & \text{Genotype} \leftarrow f_G(); \\
& \text{Smokes} \leftarrow f_S(\text{Genotype}); \\
& \text{Tar} \leftarrow f_T(\hat{s}); \\
& \text{Genotype}'' \leftarrow \textbf{norm} \begin{pmatrix} \text{Genotype}' \leftarrow f_G(); \\ \text{Smokes}' \leftarrow f_S(\text{Genotype}'); \\ _ \leftarrow \textbf{observe}\ \text{Smokes} = \text{Smokes}'; \\ \textbf{return}\ \text{Genotype}' \end{pmatrix} \\
& \text{Cancer} \leftarrow f_C(\text{Genotype}'', \text{Tar}); \\
& \textbf{return}\ \text{Cancer}
\end{aligned}$$

Next, we use Norm-Bind-Dist with $e_2 := f_C(\text{Genotype}'', \text{Tar})$; the premise is satisfied because $f_C : \mathbb{B} \times \mathbb{B} \to \mathcal{D}\,\mathbb{B}$:

$$\begin{aligned}
\mathcal{I}(\hat{s}) \equiv\ & \text{Genotype} \leftarrow f_G(); \\
& \text{Smokes} \leftarrow f_S(\text{Genotype}); \\
& \text{Tar} \leftarrow f_T(\hat{s}); \\
& \text{Cancer} \leftarrow \textbf{norm} \begin{pmatrix} \text{Genotype}' \leftarrow f_G(); \\ \text{Smokes}' \leftarrow f_S(\text{Genotype}'); \\ _ \leftarrow \textbf{observe}\ \text{Smokes} = \text{Smokes}'; \\ f_C(\text{Genotype}', \text{Tar}); \end{pmatrix} \\
& \textbf{return}\ \text{Cancer}
\end{aligned}$$

Let us call the right hand side of the above equation $\mathcal{I}'(\hat{s})$. We will look at three of its subexpressions and rewrite them individually.

First, we look at the first two lines of $\mathcal{I}'(\hat{s})$. The first variable Genotype is not used except as the input of f_S when computing Smokes, so by Drop-R,

$$\begin{aligned}
& \text{Genotype} \leftarrow f_G(); \\
& \text{Smokes} \leftarrow f_S(\text{Genotype}); \\
& \textbf{return}\ \text{Smokes}
\end{aligned} \quad \equiv \quad \begin{aligned}
& \langle \text{Smokes}, \text{Tar}, \text{Cancer} \rangle \leftarrow O(); \\
& \textbf{return}\ \text{Smokes}
\end{aligned}$$

Second, we look at the assignment to Tar. Here we apply a sequence of rules. The first equivalence is by Monad-Unit-L. The second is by Observation-Intervention-Exchange with $y := \text{Smokes}$, $y' := \hat{s}$, and e is some projection of $O()$. e has full support because $O()$ has full support by assumption. The third is by Norm-Bind-Dist. The final equivalence is by Observe-Swap, where we use the fact that $f_T(\text{Smokes})$ gets assigned to the second component of $O()$.

$$f_T(\hat{s}) \quad \equiv \quad s' \leftarrow \textbf{return}\ \hat{s}; f_T(s')$$

$$
\begin{aligned}
&\equiv \quad s' \leftarrow \mathbf{norm}\begin{pmatrix}\langle \text{Smokes}, \_, \_\rangle \leftarrow O();\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \hat{s};\\ \mathbf{return}\ \text{Smokes}\end{pmatrix};\\
&\qquad f_T(s')\\
&\equiv \quad \mathbf{norm}\begin{pmatrix}\langle \text{Smokes}, \_, \_\rangle \leftarrow O();\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \hat{s};\\ f_T(\text{Smokes})\end{pmatrix}\\
&\equiv \quad \mathbf{norm}\begin{pmatrix}\langle \text{Smokes}, \text{Tar}, \_\rangle \leftarrow O();\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \hat{s};\\ \mathbf{return}\ \text{Tar}\end{pmatrix}
\end{aligned}
$$

Finally, we look at the assignment to Cancer. First, we apply Monad-Unit-L. Next, we use Observation-Intervention-Exchange with $y' := \text{Tar}, y := \text{Tar}', e := f_T(\text{Smokes}')$; the full support premise is proven by the assumption on f_T. Next, we apply Norm-Bind-Dist, whose assumption is given by the type of f_C. We then collapse the inner norm using Norm-Idempotent. We then use Observe-Swap multiple times to move the observe statements after the assignments. Finally, we have a subexpression corresponding exactly to O, which we substitute in.

$$
\begin{aligned}
\mathbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \text{Smokes}';\\ f_C(\text{Genotype}', \text{Tar});\end{pmatrix} &\equiv \quad \mathbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \text{Smokes}';\\ t \leftarrow \mathbf{return}\ \text{Tar};\\ f_C(\text{Genotype}', t);\end{pmatrix}\\
&\equiv \quad \mathbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \text{Smokes}';\\ t \leftarrow \mathbf{norm}\begin{pmatrix}\text{Tar}' \leftarrow f_T(\text{Smokes}');\\ \_ \leftarrow \mathbf{observe}\ \text{Tar}' = \text{Tar};\\ \mathbf{return}\ \text{Tar}'\end{pmatrix};\\ f_C(\text{Genotype}', t);\end{pmatrix}\\
&\equiv \quad \mathbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \text{Smokes}';\\ \mathbf{norm}\begin{pmatrix}\text{Tar}' \leftarrow f_T(\text{Smokes}');\\ \_ \leftarrow \mathbf{observe}\ \text{Tar}' = \text{Tar};\\ f_C(\text{Genotype}', \text{Tar}');\end{pmatrix};\end{pmatrix}\\
&\equiv \quad \mathbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \_ \leftarrow \mathbf{observe}\ \text{Smokes} = \text{Smokes}';\\ \text{Tar}' \leftarrow f_T(\text{Smokes}');\\ \_ \leftarrow \mathbf{observe}\ \text{Tar}' = \text{Tar};\\ f_C(\text{Genotype}', \text{Tar}');\end{pmatrix}
\end{aligned}
$$

$$\equiv \quad \textbf{norm}\begin{pmatrix}\text{Genotype}' \leftarrow f_G();\\ \text{Smokes}' \leftarrow f_S(\text{Genotype}');\\ \text{Tar}' \leftarrow f_T(\text{Smokes}');\\ \text{Cancer}' \leftarrow f_C(\text{Genotype}', \text{Tar}');\\ \_ \leftarrow \textbf{observe}\, \text{Smokes} = \text{Smokes}';\\ \_ \leftarrow \textbf{observe}\, \text{Tar}' = \text{Tar};\\ \textbf{return}\, \text{Cancer}'\end{pmatrix}$$

$$\equiv \quad \textbf{norm}\begin{pmatrix}\langle \text{Smokes}', \text{Tar}', \text{Cancer}'\rangle \leftarrow O();\\ \_ \leftarrow \textbf{observe}\, \text{Smokes} = \text{Smokes}';\\ \_ \leftarrow \textbf{observe}\, \text{Tar}' = \text{Tar};\\ \textbf{return}\, \text{Cancer}'\end{pmatrix}$$

Now, we can substitute these three equivalences into $\mathcal{I}'(\hat{s})$. We get this final program:

$$\begin{aligned}\mathcal{I}(\hat{s}) \equiv\ & \langle \text{Smokes}, \_, \_\rangle \leftarrow O();\\ & \text{Tar} \leftarrow \textbf{norm}\begin{pmatrix}\langle \text{Smokes}', \text{Tar}', \_\rangle \leftarrow O();\\ \_ \leftarrow \textbf{observe}\, \text{Smokes}' = \hat{s};\\ \textbf{return}\, \text{Tar}'\end{pmatrix};\\ & \text{Cancer} \leftarrow \textbf{norm}\begin{pmatrix}\langle \text{Smokes}', \text{Tar}', \text{Cancer}'\rangle \leftarrow O();\\ \_ \leftarrow \textbf{observe}\, \text{Smokes} = \text{Smokes}';\\ \_ \leftarrow \textbf{observe}\, \text{Tar}' = \text{Tar};\\ \textbf{return}\, \text{Cancer}'\end{pmatrix}\\ & \textbf{return}\, \text{Cancer}\end{aligned}$$

Notice that the flow of information between the subprograms is preserved by these substitutions. Further, the only unknown subexpression is $O()$, whose distribution we have access to. Thus, our original program $\mathcal{I}(\hat{s})$ is identifiable.