CAUSAL PROBABILISTIC PROGRAMMING VIA MAGMADIC DO-NOTATION

MARIO ROMÁN

ABSTRACT. We introduce a denotational semantics for a simple form of causal probabilistic programming, based on magmadic do-notation. We propose to derive the identifiability constructs that causal probabilistic programming requires from the non-associative phenomena of probabilistic inference.

1. INTRODUCTION

1.1. **Example — Simpson's paradox in observational clinical data.** Imagine we have two treatments, named A and B, for the same disease, X. From the observational data, A seems slightly less successful than B: they achieve remission in 81.3% and 83.8% of the cases, respectively.

	Treatment A	*Treatment* B
Variant X	81.3% (61 : 14)	83.8% (88 : 17)
Variant X_1	93.3% (28 : 02)	86.6% (78 : 12)
Variant X_2	73.3% (33 : 12)	66.6% (10 : 05)

FIGURE 1. Remission odds, by treatment and variant.

However, we observe that clinical practice is to subdivide the illness in two variants, X_1 and X_2. After segregating patients (Figure 1), we find that treatment A is separately more successful for X_1 — with a success rate of 93.3% versus 86.6% — and for X_2 — with a success rate of 73.3% versus 66.6%. The explanation for this apparent paradox is that the choice of treatment was being influenced by the variant — while the X_1 variant was mostly treated with B, the more difficult and rare X_2 variant was being systematically sent for treatment A.

This is *Simpson's paradox*, a failure to account for a confounding variable in the observational data. Randomised trials avoid this issue: e.g., if we had truly randomized the application of A or B, we would have been able to compare both under equal conditions.[1]

But, do we always *need* randomised trials to estimate treatment effect?

Date: August 29, 2026.

[1]This is not an entirely fictional scenario. In 1986, an observational study by Charig et al. [CWPW86] reported the relative success of different modalities of kidney stone surgery. Open surgery seemed slightly less successful than a modality of closed surgery. This was a real-world instance of *Simpson's paradox*, as noted in 1994 by Julious and Mullee [JM94], who concluded that "randomised trials are therefore necessary to demonstrate any treatment effect." Similar real-world examples exist for many causal inference problems [Pea09].

1.2. **Example — causal reasoning with observational data.** In particular, in this example, we can estimate a randomised control trial from the given data.

(1) We can estimate the prevalence of the X_1 and X_2 variants. We have 120 patients with the X_1 variant and 60 patients with the X_2 variant (Figure 1).
(2) We can weight the success of the treatment A on each variant by the actual probability of each, $28/30 \cdot 120/180 + 33/45 \cdot 60/180 = 13/15$.
(3) Analogously, we can weight the success of treatment B by the actual probabilities, $78/90 \cdot 120/180 + 10/15 \cdot 60/180 = 12/15$.
(4) As a consequence, a 1:1 randomised control trial of A and B, with 30 patients, would expect $13/15$ to achieve remission with A and $12/15$ to achieve remission with B.

Thus, we have a mathematically valid estimator for a randomized control trial; and no randomized controlled trial was necessary for this estimate. Although the problem of finding estimators under causal assumptions is decidable and solved [SP08, TS10], it is generally difficult to reason about causality [Pea09].

This manuscript introduces a syntax and denotational semantics for a simple form of *causal probabilistic programming*: a programming paradigm for causal inference built over *probabilistic programming*.

1.3. **Probabilistic programming.** Probabilistic programming is a programming paradigm where statistical models are first-class citizens of the language; it facilitates a syntax for probabilistic inference. Probabilistic inference is the problem of estimating a conditional probability from observational data and a statistical model.

While is a mathematically-solved problem, it is conceptually difficult to apply correctly. Probabilistic programming allows us to declaratively specify problems of probabilistic inference. It achieves so via an `observe` operator that imposes a conditional constraint on a distribution (see, for instance, Figure 2).

```
(do (variant treatment outcome) <- data
    () <- (observe treatment 'A)
    return (outcome))
>>> '(((success) 61/75)
>>>   ((failure) 14/75))

(do (variant treatment outcome) <- data
    () <- (observe treatment 'B)
    return (outcome))
>>> '(((success) 88/105)
>>>   ((failure) 17/105))
```

FIGURE 2. Naive conditional estimation of treatment outcome.

However, a naive application of probabilistic inference would fall prey to the confounding variable problem in Section 1.1 (see the results in Figure 2). The question the following program answers is: *"after observing that I have been assigned to treatment A (or B), what are the odds of remission?"* Note that this is different from asking *"what are the odds of remission if I undergo treatment A (or B)?"*

Conditioning on the observation does not help to answer this second question, for what we really need is a causal model that explains out the bias introduced by

deciding treatment based on variant. Endowed with a causal model, causal inference is solvable [SP08], but still conceptually difficult. This conceptual difficulty is eased by the syntax of *causal probabilistic programming*.

1.4. **Causal probabilistic programming.** *Causal probabilistic programming* allows us to declaratively specify problems of causal inference. Causal inference is the problem of estimating effects from observational data and a causal model [Pea09]. As an example, consider that we want to estimate the effect of a 1:1 randomized control trial for the data in Section 1.1. Figure 3 shows the query code: it assigns a treatment arm uniformly and estimates the causal effect of the treatment.

```
(do (arm) <- (uniform '(A) '(B))
    (outcome) <- (intervene data
                  withModel (do
                    variant <- ()
                    treatment <- (variant)
                    outcome <- (variant treatment)
                    return (variant treatment outcome))
                  setting (treatment) to (arm)
                  in (outcome))
    return (arm outcome))
>>> '(((open success) 634983/1525400)
>>>  ((open failure) 127717/1525400)
>>>  ((closed success) 6231/16000)
>>>  ((closed failure) 1769/16000))
```

FIGURE 3. Causal estimation of a randomized control trial.

Causal inference is achieved via the `intervene` operator: it takes a distribution and a causal model and yields the intervened distribution — or an exception when impossible. `Intervene` is not a primitive: it is internally rewritten into non-associative sequencing and `observe`. For instance, Figure 3 becomes Figure 4.

```
(do (arm) <- (uniform '(A) '(B))
    (outcome) <- (do
      (variant) <- (do
        (variant1 treatment1 outcome1) <- data
        return (variant1))
      (outcome) <- (do
        (variant2 treatment2 outcome2) <- data
        () <- (observe treatment treatment2)
        () <- (observe variant variant2)
        return (outcome2))
      return (outcome))
    return (arm outcome))
```

FIGURE 4. Probabilistic rewriting of the causal estimation.

From this point of view, the primitives of our syntax are the (non-associative) `do` and the `observe` primitives from normalized probabilistic programming. Next section discusses the non-associative `do` primitive [DRS26] (Section 2); last section recasts an identifiability algorithm [SP08] in terms of `do` and `observe` (Section 3).

Remark 1.4.1. The causal model can be itself declared using a form of single-output anonymous do-notation, as we do in Figure 3. It must specify a semimarkovian model.

2. Magmadic Do-notation

Do-notation is right-associative by default [Mog91, Wad92]. We instead employ left-associative do-notation. We take semantics in the magmad — the non-associative monad — of normalized distributions (Definition 2.3.1, see [DRS26]).

2.1. **Magmads.** Magmads are the non-associative counterpart of monads.[2] We introduce here Set-based magmads in *relative form* (or *Kleisli form*): magmads in terms of *unit* and *bind*.

Definition 2.1.1 (Magmad)**.** A *(unital) magmad*, $(\mathbf{T}, \eta^{\mathbf{T}}, \beta^{\mathbf{T}})$, consists of an assignment on sets, $\mathbf{T}X$ for each set X, a natural family of functions $\eta^{\mathbf{T}} \colon X \to \mathbf{T}X$ (called *unit*) and a natural family of functions $\beta^{\mathbf{T}} \colon \mathbf{T}X \times (X \to \mathbf{T}Y) \to \mathbf{T}Y$ (called *bind*), satisfying the following two axioms.

(1) $\beta^{\mathbf{T}}(\eta^{\mathbf{T}}(x))(\lambda x.f(x)) = f(x)$, left unitality; and
(2) $\beta^{\mathbf{T}}(t)(\lambda x.\eta^{\mathbf{T}}(x)) = t$, right unitality.

Remark 2.1.2. A monad is a magmad additionally satisfying associativity,

(3) $\beta^{\mathbf{T}}(t)(\lambda x.\beta^{\mathbf{T}}(f(x))(g)) = \beta^{\mathbf{T}}(\beta^{\mathbf{T}}(t)(\lambda x.f(x)))(\lambda y.g(y))$.

2.2. **Magmadic do notation.** Because we do not have a monad but only a magmad, the usual monadic programming language [Mog91] can take two meanings depending on where we associate by default. Moreover, nested `do` does not reduce to a single `do`, as it happens with its associative counterpart.

Definition 2.2.1 (Magmadic do notation)**.** Magmadic do notation is defined recursively by the following rules. The three first rules *(i,ii,iii)* deal with the three base cases. The last rule *(iv)* groups two lines, using $m = \beta(m_1; \lambda x.\beta(m_2; \lambda y.\eta(x, y)))$.

```
(do return (y ...))
```
$\overset{(i)}{=}$ $\eta(y_1, ..., y_m)$

```
(do (x ...) <- m
    return (y ...))
```
$\overset{(ii)}{=}$ $\beta(m; \lambda(x_1, ..., x_n).$
$\eta(y_1, ..., y_m)).$

```
(do (x ...) <- m1
    (y ...) <- m2
    return (z ...))
```
$\overset{(iii)}{=}$ $\beta(m_1; \lambda(x_1, ..., x_n).$
$\beta(m_2; \lambda(y_1, ..., y_m).$
$\eta(z_1, ..., z_k)))$

```
(do (x ...) <- m1
    (y ...) <- m2
    rest ...)
```
$\overset{(iv)}{=}$
```
(do (x ... y ...) <-
      (do (x ...) <- m1
          (y ...) <- m2
          return (x ... y ...))
    rest ...)
```

[2] More precisely, we use *unital* magmads and call them *magmads* thoughout this text.

2.3. **Magmad of normalized distributions.** We recall the *magmad*, or *non-associative monad*, of normalized distributions [DRS26]. A normalized distribution is either an empty formal sum — meaning it "fails", or that it is undefined — or a full distribution. Normalized distributions provide normalized stochastic semantics, but they do not form a monad, only a magmad.

Definition 2.3.1 (Magmad of normalized distributions)**.** A *normalized distribution* over a set is a finite formal sum over its elements whose coefficients are positive and add up to exactly 1 or 0. The *finitary normalized distribution magmad*, $\mathbf{N}\colon \mathsf{Set} \to \mathsf{Set}$, assigns, to each set, the set of normalized distributions over it,

$$\mathbf{N}(X) = \left\{ \sum_{i=0}^{n} \lambda_i \left|x_i\right\rangle \;\middle|\; x_i \in X,\ \lambda_i \in \mathbb{R}^+,\ \sum_{i=0}^{n} \lambda_i = 1 \text{ or } 0 \right\}.$$

Its unit, $\eta^{\mathbf{N}}\colon X \to \mathbf{N}X$, is defined by $\eta^{\mathbf{N}}(x) = 1\left|x\right\rangle$. Its binding, $\beta^{\mathbf{N}}\colon \mathbf{N}X \times (X \to \mathbf{N}Y) \to \mathbf{N}Y$, is defined by

$$\beta^{\mathbf{N}}\left(\sum_{i=0}^{n} \lambda_i \left|x_i\right\rangle ; \left[x_i \mapsto \sum_{j=0}^{m_i} \rho_{ij} \left|y_j\right\rangle\right]\right) = \sum_{i=0}^{n}\sum_{j=0}^{m_i} \frac{\lambda_i \rho_{ij} \left|y_j\right\rangle}{\sum_{i=0}^{n}\sum_{j=0}^{m_i} \lambda_i \rho_{ij}}.$$

Note that, whenever $\sum_{i=0}^{n}\sum_{j=0}^{m_i} \lambda_i \rho_{ij} = 0$, then there must be no summands and the value of the binding is the empty formal sum, 0.

3. Identifiability

Section 3.1 derives conditional distributions from the basic primitives; Section 3 derives the whole Identifiability algorithm.

3.1. **Conditionals from observations.** The conditional distribution $P(x|y)$ is traditionally computed by Bayes rule:

$$P(x|y) = \frac{P(x)P(y|x)}{\sum_{x'} P(x')P(y|x')}.$$

Note that this is precisely the normalization of the distribution $P(x)P(y|x) = P(x,y)$ constrained to a particular value of y. In terms of our probabilistic language, this is the same as computing $P(x,y)$ and then observing a particular value for y.

More generally, the value of $P(a_1, ..., a_n \mid b_1, ..., b_m)$ is obtained by computing $P(a_1, ..., a_n, b_1, ..., b_m)$ and observing particular values for the b_i variables. In the following formula, v_i refers to a temporary variable for the i-th output of the distribution; v_{bi} refers to the temporary variable corresponding to b_i.

```
p(a1 ... an | b1 ... bm)
```

=

```
(do (v1 ... vk) <- p
    () <- (observe b1 vb1)
    ...
    () <- (observe bm vbm)
    return (a1 ... an))
```

3.2. **Identifiability algorithms.** The **Identify** algorithm checks if a set of variables C can be identified inside of a single confounded component T containing them, $C \subseteq T$. It is based on Shpitser and Pearl's identify algorithm [SP08], but differs in that *(i)* it does not use multiplications or divisions, but only do-notation; *(ii)* it computes conditionals separately, using `observe`; and *(iii)* it outputs the

selected variable as a single separated output, $x \in C$, instead of outputting all variables at once as in the original algorithm — this is helpful for the ID algorithm.

Definition 3.2.1 (Causal model)**.** A *(semimarkovian) causal model* consists of a directed acyclic graph with a subset of *visible* variables, such that each non-visible variable does not have any parents and is the parent of exactly two visible variables.

Algorithm 3.2.2 (Identify algorithm, c.f. [SP08])**.** Let G be a causal model inducing a topological partial order $(\leq)$. Let T be a confounded component; let $C \subseteq T$ be a subset of it; and let $x \in C$ be a variable. Let q be a distribution over T.

The following algorithm computes $\text{Identify}(q, G, T, C, x)$.

(1) Compute $A = \text{Ancestors}(C)_{G(T)}$, the ancestors of C in the subgraph given by the component T.
(2) If $A = C$, then return the conditional distribution $q(x \mid \{c < x \mid c \in C\})$.
(3) If $A \neq C$ and $A = T$, then fail.
(4) If $C \subset A \subset T$, then let T' be the confounded component of C in $G(A)$.
    (a) Compute a conditional distribution (by Section 3.1) over the subset using the fact that it is a separated confounded component.

```
q' = (do t1' <- q(t1' | {a : a < t1'})
         ...
         tn' <- q(tn' | {a : a < tn'})
         return (t1' ... tn'))
```

    (b) Recursively apply $\text{Identify}(q', G(T'), T', C, x)$.

Algorithm 3.2.3 (ID algorithm)**.** The following algorithm computes $\text{ID}(p, G, T, S)$.

(1) Compute the ancestors of S in the graph where all the variables in T are removed, $D = \text{Ancestors}(S)_{G(V-T)}$.
(2) For each variable $d_i \in D$, let S_i be its confounded component.
(3) For each variable $d_i \in D$, let D_i be its confounded component in the graph $G(D)$, restricted to variables in D.
(4) For each variable $d_i \in D$, let QS_i be computed as a conditional distribution (by Section 3.1) on a separated confounded component.

```
QSi = (do si1 <- p(si1 | {v | v < si1})
          ...
          sim <- p(sim | {v | v < sim})
          return (si1 ... sim))
```

(5) For each variable in D, identify it using a call to the main Identify algorithm, $\text{Identify}(QS_i, G, S_i, D_i, d_i)$.

```
q = (do d1 <- Identify(QS1, G, S1, D1, d1)
        ...
        dn <- Identify(QSn, G, Sn, Dn, dn)
        return (s1 ... sn))
```

Remark 3.2.4. Shpitser and Pearl have shown that this algorithm is both sound — i.e., when it succeeds, it computes a correct estimator of the intervened distribution — and complete — i.e., it only fails when it is impossible to provide a correct

estimator [SP08]. However, they have done so assuming a particular semantics: that of probability distributions.

3.3. **Identifiability as a derived construct.** We introduce a single construct to magmadic do notation. It represents the result of modeling a distribution p with a causal model G and, after performing an intervention that sets some variables $(v_1, ..., v_n)$ to some values $(x_1, ..., x_n)$, identifying the value of some output variables $(y_1, ..., y_m)$. We use the previous identifiability algorithm.

```
(intervene p withModel G
 setting (v ...) to (x ...)
 in (y ...))
```

$$= \mathrm{ID}(p, G, \{v_1, ..., v_n\}, \{y_1, ..., y_m\})(x_1, ..., x_n)$$

FIGURE 5. Definition of the "intervene" construct.

In this way, every instance of the "intervene" construct is systematically rewritten into the output of the ID algorithm, which contains a probabilistic program estimating the intervention.

4. IMPLEMENTATION

A Racket implementation of the algorithms described here is publicly available [Rom26]. Additionally, the Appendix contains two more examples of automatic rewriting using the identifiability algorithm.

5. RELATED WORK

Causal programming has been the subject of recent theoretical work [WSB+] and implementations [Cc23, TKZ+21], but literature towards a denotational semantics of causal probabilistic programming had not yet explicitly addressed the problem of identifiability.

We introduce a simple syntax and algebraic denotational semantics for causal programming. Our technique is not to consider *causal programming* as a new paradigm requiring new primitives, but to introduce derived constructs on top of non-associative probabilistic programming with normalized semantics [DRS26].

ACKNOWLEDGEMENTS

Parts of this article build upon previous joint work with Márk Széles and Elena Di Lavore [DRS26]. The author thanks Márk Széles and Elena Di Lavore for their encouragement and much discussion on this article; and thanks Wessel de Weijer and Théo Wang for their helpful comments on earlier versions of it.

Funding. Mario Román was supported by the Estonian Research Council grant PRG 3215 ("Post Cartesian Programming - Logic, Probability and Computation with String Diagrams"), and by the Advanced Research + Invention Agency (ARIA) Safeguarded AI Programme.

References

[Cc23] The ChiRho contributors. ChiRho: An experimental language for causal reasoning. Available at https://github.com/BasisResearch/chirho, 2023.

[CWPW86] Clive R Charig, David R Webb, Stephen Richard Payne, and John E Wickham. Comparison of treatment of renal calculi by open surgery, percutaneous nephrolithotomy, and extracorporeal shockwave lithotripsy. *British medical journal (Clinical research ed.)*, 292(6524):879–882, 1986.

[DRS26] Elena Di Lavore, Mario Román, and Márk Széles. Normalized probabilistic semantics is not associative. *arXiv preprint*, 2026. doi:10.48550/arXiv.2510.01131.

[JM94] Steven A Julious and Mark A Mullee. Confounding and Simpson's paradox. *British Medical Journal*, 309(6967):1480–1481, 1994.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

[Pea09] Judea Pearl. *Causality*. Cambridge University Press, 2009.

[Rom26] Mario Román. do: A magmadic metalanguage for probabilistic and causal inference (v0.1.0). https://github.com/mroman42/do, 2026.

[SP08] Ilya Shpitser and Judea Pearl. Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, 9(64):1941–1979, 2008. URL http://jmlr.org/papers/v9/shpitser08a.html.

[TKZ+21] Zenna Tavares, James Koppel, Xin Zhang, Ria Das, and Armando Solar-Lezama. A language for counterfactual generative models. In *International conference on machine learning*, pages 10173–10182. PMLR, 2021.

[TS10] Jin Tian and Ilya Shpitser. On identifying causal effects. *Heuristics, Probability and Causality: A Tribute to Judea Pearl (R. Dechter, H. Geffner and J. Halpern, eds.). College Publications, UK*, pages 415–444, 2010.

[Wad92] Philip Wadler. The essence of functional programming. In Ravi Sethi, editor, *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 19-22, 1992*, pages 1–14. ACM Press, 1992. doi:10.1145/143165.143169.

[WSB+] Theo Wang, Dario Stein, Eli Bingham, John Feser, Ohad Kammar, Michael Lee, and Jeremy Yallop. Typed abstractions for causal probabilistic programming.

Appendix A. Further examples

A.1. **Front-door criterion.** The paradigmatic problem of causal inference estimates the effect of smoking on cancer from some (fake) data suggesting a protective effect. The estimation uses a middle observed variable (*tar in the lungs*) and this technique is usually known as the *front-door criterion* [Pea09]. The query in Figure 6 asks, assuming this (fake) data, what would be the incidence of cancer if 5% of the population were to smoke.

```
(define survey (distribution-table
     ['(smoker tar nocancer)     323]
     ['(smoker tar cancer)        57]
     ['(nonsmoker tar nocancer)    1]
     ['(nonsmoker tar cancer)     19]
     ['(smoker notar nocancer)    18]
     ['(smoker notar cancer)       2]
     ['(nonsmoker notar nocancer) 38]
     ['(nonsmoker notar cancer)  342]))

; Which would be the incidence with a 5% of smokers?
(define (incidence-from-habits habits)
  (intervene survey withModel
    (do gene <- ()
        smoking <- (gene)
        tar <- (smoking)
        cancer <- (gene tar)
        visibles (smoking tar cancer))
   setting (smoking) to (smoking-habits) in (cancer)))

(do (newHabits) <- (distribution
                    ['(smoker)     5/100]
                    ['(nonsmoker) 95/100])
    (incidence) <- (incidence-from-habits newHabits)
    return (incidence))
```

Figure 6. Smoking example.

Internally, the code computing incidence gets rewritten as in Figure 7.

```
(define (incidence-from-habits habits)
  (do
    (tar) <- (do
        (smoking1 tar2 cancer3) <- survey
        () <- (observe smoking smoking1)
        return (tar2))
    (cancer) <- (do
        (cancer11 smoking12) <- (do
            (smoking) <- (do
                (smoking5 tar6 cancer7) <- survey
                return (smoking5))
            (cancer) <- (do
                (smoking8 tar9 cancer10) <- survey
                () <- (observe tar tar9)
                () <- (observe smoking smoking8)
                return (cancer10))
            return (cancer smoking))
        return (cancer11))
    return (cancer)))
```

FIGURE 7. Automatic rewriting of Figure 6.

A.2. **Nested causal effects.** Our final example corresponds to Pearl's *napkin problem*, which is not usually provided with a real-world interpretation. Its purpose is to show that the rewriting algorithm does handle nested cases and translates them to probabilistic programming primitives.

```
(intervene p
  withModel (do
               u1 <- ()
               u2 <- ()
               w <- (u1 u2)
               z <- (w)
               x <- (z u1)
               y <- (x u2)
               return (w z x y))
  setting (x) to ('i) in (y))
```

FIGURE 8. "Napkin" problem example.

```
(do
  (x16 y17) <- (do
      (x13 w14 y15) <- (do
          (w) <- (do
              (w1 z2 x3 y4) <- p
              return (w1))
          (x) <- (do
              (w5 z6 x7 y8) <- p
              () <- (observe z z6)
              () <- (observe w w5)
              return (x7))
          (y) <- (do
              (w9 z10 x11 y12) <- p
              () <- (observe x x11)
              () <- (observe z z10)
              () <- (observe w w9)
              return (y12))
          return (x w y))
      return (x13 y15))
  () <- (observe 'i x16)
  return (y17))
```

FIGURE 9. Automatic rewriting of Figure 8.