Typed Abstractions for Causal Probabilistic Programming

Theo Wang*
theo.wang@spc.ox.ac.uk
University of Oxford
United Kingdom

Dario Stein*
dario.stein@ru.nl
Radboud University, Nijmegen
Netherlands

Eli Bingham
eli@basis.ai
Basis
United States

John Feser
jack@basis.ai
Basis
United States

Ohad Kammar
ohad.kammar@ed.ac.uk
University of Edinburgh
United Kingdom

Michael Lee
michael.lee@cl.cam.ac.uk
University of Cambridge
United Kingdom

Jeremy Yallop
jeremy.yallop@cl.cam.ac.uk
University of Cambridge
United Kingdom

1 Introduction and Contributions

Causal inference plays a central role in empirical science and there is increasing interest in applying causal concepts to machine learning (e.g. surveys [11, 20], frameworks [4]), including applications such as counterfactual token generation for LLMs [6, 19]. The standard treatment uses Pearl's *structural causal models* (SCMs), which are inexpressive: we wish to apply causal queries to a wider range of programs.

Enter *causal probabilistic programming*: fully-fledged probabilistic programming with first-class causal primitives. Several existing languages fit this paradigm; OmegaC [24] offers an intervention operator, at the price of a nonstandard evaluation strategy and dynamic scoping. MultiVerse [17] and ChiRho [2] build on Pyro [5, 18]. ChiRho seamlessly interacts with Pyro and PyTorch, leading to very performant code. However, its causal mechanism is entangled with PyTorch vectorization and Pyro's custom effect handler system.

This talk distills the core ChiRho design as a Haskell library, disentangling the dependence on vectorization and effect handlers, adding static types, and clarifying the semantics.

2 Causal Reasoning

Pearl's systematic treatment of causality [16] introduces a hierarchy of three 'rungs' of causal questions:

I (observational) "seeing" – is X associated with Y?

II (interventional) "doing" – does X cause Y?

III (counterfactual) "imagining" – given that we saw Y, was X its cause?

Probabilistic programming, by construction, resides on rung I. Just as probabilistic programs encode probabilistic models and observations, we want to use *causal probabilistic programs* to encode causal models and queries.

As Pearl's rungs are mathematically distinct [16], we generally cannot answer a rung II question with only rung I information, barring further structure or assumptions. That is, causal programs need to expose more of their *intension* than probabilistic ones. How to do this in a principled way, i.e. only exposing the right amount of intension, is one focus of this talk.

2.1 Causal reasoning, by inspection

Our running causal query example is written in Haskell using a monadic inference library such as MonadBayes or LazyPPL [7, 21–23]:

```
uel <- bernoulli(0.8)
battery <- bernoulli(0.9)
let carStarts = fuel && battery
return carStarts
```

A car requires fuel and battery power to start. We wonder: Given that the car did *not* start this morning, would it have started if it had been refueled the night before? By manual inspection, we can transform this counterfactual query into a Bayesian one, namely: what is the probability that the

*Both authors contributed equally to this research.

battery was at fault?

$$p(\text{battery}|\neg(\text{battery} \wedge \text{fuel})) = \frac{0.9 \cdot 0.2}{1 - 0.9 \cdot 0.8} = \frac{9}{14} \approx 64\%$$

Note that in our model, battery and fuel were modelled as independent. Under different causal assumptions (for example, battery depending on fuel) the answer would be different.

2.2 Causal reasoning, systematically

Inspection works well for simple examples, but to answer counterfactual queries *systematically*, we can apply a series of program transformations consistent with Pearl's do-calculus. The *twinning transformation* (e.g. [10, 16]) splits the model into two correlated copies or 'worlds': the factual world and the (prime-suffixed) counterfactual world:

```
uel <- bernoulli(0.8)
battery <- bernoulli(0.9)
let fuel' = fuel
let battery' = battery
let carStarts = fuel && battery
let carStarts' = fuel' && battery'
return carStarts'
```

The splitting treats the variables `fuel` and `battery` as *exogenous*, i.e. *shared* between both worlds. We can perform a *do-intervention* by assigning `fuel' = true` in the counterfactual branch. We then use the underlying Bayesian facilities to condition on the real-world observations (the car did not start), thereby obtaining an updated prediction in the counterfactual world. The answer to our counterfactual query is thus computed by the transformed probabilistic program:

```
uel <- bernoulli(0.8)
battery <- bernoulli(0.9)
let fuel' = True -- do-intervention
let battery' = battery
let carStarts = fuel && battery
let carStarts' = fuel' && battery'
condition(carStarts == False) -- conditioning
return carStarts'
```

2.3 Causal reasoning, automatically

Manually transforming programs becomes infeasible for more complex causal models. We wish to express causal probabilistic models in a first-class way that lets such transformations be performed automatically, safely and compositionally. The ChiRho language provides features that support these transformations. This section presents the core ChiRho functionality, distilled as a Haskell library.

For Rung I, ChiRho inherits Pyro's probabilistic programming features. For the two higher rungs, we introduce the following abstractions:

Rung II. To manage interventions, we track *intervention points*, which are typed identifiers of program locations where interventions are allowed. We define a monad `Caus` representing a causal model. `Caus` is a reader monad; a model is a suspended computation that is executed under *intervention instructions*, which specify intervention point behavior.

Rung III. To manage counterfactuals, we track expressions which differ by *world* (0=factual, 1=counterfactual). The type `MVal a` represents such *multi-values*. Elements (n, v) pair a finite list n of names with a map $v : 2^n \to a$. Each name in n refers to a branching point, and an assignment $w = (w_1, \ldots, w_n) \in 2^n$ identifies a specific world, i.e. a choice of branch for each branching point, e.g. in

$$M_1 = ([], \{() : a\}), \qquad M_2 = (['b'], \{(0) : a_0, (1) : a_1\}),$$

M_1 is an unbranched multi-value, while M_2 contains a branching named $'b'$; in the world where $'b' = 0$, it takes (factual) value a_0, and for $'b' = 1$ it takes (counterfactual) value a_1.

ChiRho represents multi-values as tensors with one dimension per branching point, and uses *broadcasting* to share information across different worlds. In Haskell, we make this broadcasting explicit via an applicative structure [14] on `MVal`. For example, computing a binary function f over M_1 and M_2 results in the branched value

$$f\langle\$\rangle M_1 \langle*\rangle M_2 = (['b'], \{(0) : f(a, a_0), (1) : f(a, a_1)\}) \quad (1)$$

where broadcasting results in a being shared across the factual and counterfactual worlds.

Worked example. We can now formulate our example causal probabilistic model using the new causal primitives.

```
1 arModel :: MonadDistribution m
2   => InterventionPoint m Bool
3   -> Caus m (MVal Bool)
4 carModel fuelPt = do
5   fuel <- sample (pure (bernoulli 0.8));
6   fuelInt <- new_ fuelPt fuel;
7   battery <- sample (pure (bernoulli 0.9));
8   let carStarts = ((&&) <$> fuelInt) <*> battery;
9   return carStarts
```

l1-l3 the type signature shows that this is a causal model exposing one Boolean intervention point (for fuel) and returning a Boolean multivalue.

l5 we initialize `fuel :: MVal Bool` as an unbranched multivalue, containing a random Boolean.

l6 `new_` reads the intervention instructions associated with intervention point `fuelPt`; the multivalue `fuelInt` is a possibly branched version of `fuel`, depending on what interventions we encounter.

l7 `battery` is an unbranched multivalue, containing a random Boolean.

l8 we use the applicative structure of `MVal` to compute `carStarts` across all branches. Note that as in eq. (1), `battery` will be shared across both branches.

The following workflow evaluates our counterfactual query:

```
uelPt <- createKey -- (implementation detail)
let intervenedConditionedModel = do
  -- attach intervention instructions for `fuelPt`
  carStarts <- do_ fuelPt (Value True)
                   "fuelTrue" (carModel fuelPt);
  -- condition on the factual value of carStarts
  condition (not (getFactual carStarts));
  -- return its counterfactual value
  return (getCounterfactual carStarts);
-- Run inference
avg <- infer (run intervenedConditionedModel)
print avg  -- approx 0.64
```

We use the `do_` function to append a specific intervention instruction to `carModel`: at intervention point `fuelPt`, create a counterfactual branch with value `True`. This controls the behavior of `new_` in line 5 of `carModel`. The remaining program uses `getCounterfactual` and `getFactual` to read off the corresponding values from a multivalue, and then applies ordinary Bayesian conditioning and inference. The interested reader can find nontrivial examples of our implementation in the project repository [3], including a full translation of the ChiRho tutorial code [1].

Grading. Semantically, the monad `Caus` is graded by the available intervention points [8, 15]. As Haskell's effect system doesn't allow tracking of grades, we pass values of type `InterventionPoint` around manually, thereby tracking intervention points in the type signature. Grading has been studied in probabilistic programming for example in [9, 12, 13].

3 Context and Methodology

Our language follows a Bayesian approach to causality which is consistent with Bayesian probabilistic programming. To *infer* parameters or causal structures, we consider a prior over them and use the inference capabilities of the underlying PPL [25, 26]. This workflow and its limitations are detailed in [1]. In particular, questions of identifiability (do-calculus) or partial identifiability need to be addressed separately.

References

[1] [n. d.]. Causal probabilistic programming without tears - ChiRho documentation. https://basisresearch.github.io/chirho/tutorial_i.html. Accessed: 2025-10-30.

[2] [n. d.]. ChiRho. https://basisresearch.github.io/chirho. Accessed: 2025-10-30.

[3] [n. d.]. ChiRho Haskell. https://anonymous.4open.science/r/chirho-haskell-FBCB/. Accessed: 2025-10-30.

[4] [n. d.]. PyWhy, An Open Source Ecosystem for Causal Machine Learning. https://www.pywhy.org/. Accessed: 2025-10-30.

[5] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2018. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* (2018).

[6] Ivi Chatzi, Nina Corvelo Benz, Eleni Straitouri, Stratis Tsirtsis, and Manuel Gomez-Rodriguez. 2024. Counterfactual token generation in large language models. *arXiv preprint arXiv:2409.17027* (2024).

[7] Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. 2023. Affine Monads and Lazy Structures for Bayesian Programming. *Proc. ACM Program. Lang.* 7, POPL, Article 46 (Jan. 2023), 31 pages. https://doi.org/10.1145/3571239

[8] Soichiro Fujii, Shin-ya Katsumata, and Paul-André Melliès. 2016. Towards a formal theory of graded monads. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 513–530.

[9] Bruno Gavranović. 2024. *Fundamental Components of Deep Learning: A category-theoretic approach.* Ph. D. Dissertation. University of Strathclyde.

[10] Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. 2019. Causal inference by string diagram surgery. In *Foundations of Software Science and Computation Structures: 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings 22.* Springer, 313–329.

[11] Jean Kaddour, Aengus Lynch, Qi Liu, Matt J Kusner, and Ricardo Silva. 2022. Causal machine learning: A survey and open problems. *arXiv preprint arXiv:2206.15475* (2022).

[12] Alexander K Lew, Marco F Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K Mansinghka. 2019. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–32.

[13] Jack Liell-Cock and Sam Staton. 2025. Compositional imprecise probability: A solution from graded monads and markov categories. *Proceedings of the ACM on Programming Languages* 9, POPL (2025), 1596–1626.

[14] Conor McBride and Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18, 1 (2008), 1–13. https://doi.org/10.1017/S0956796807006326

[15] Dominic Orchard, Philip Wadler, and Harley Eades III. 2020. Unifying graded and parameterised monads. *Proc. MSFP 2020* (2020).

[16] Judea Pearl. 2009. *Causality.* Cambridge University Press.

[17] Yura Perov, Logan Graham, Kostis Gourgoulias, Jonathan Richens, Ciaran Lee, Adam Baker, and Saurabh Johri. 2020. Multiverse: causal reasoning using importance sampling in probabilistic programming. In *Symposium on advances in approximate bayesian inference.* PMLR, 1–36.

[18] Du Phan, Neeraj Pradhan, and Martin Jankowiak. 2019. Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro. *arXiv preprint arXiv:1912.11554* (2019).

[19] Edoardo Pona, Milad Kazemi, Yali Du, David Watson, and Nicola Paoletti. 2025. Abstract Counterfactuals for Language Model Agents. *arXiv preprint arXiv:2506.02946* (2025).

[20] Bernhard Schölkopf. 2022. Causality for machine learning. In *Probabilistic and causal inference: The works of Judea Pearl.* 765–804.

[21] Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. 2015. Practical probabilistic programming with monads. In *Proceedings of the 2015 ACM SIGPLAN Symposium on Haskell.* 165–176.

[22] Adam Ścibior, Ohad Kammar, and Zoubin Ghahramani. 2018. Functional programming for modular Bayesian inference. *Proceedings of the ACM on Programming Languages* 2, ICFP (2018), 1–29.

[23] Adam Michał Ścibior. 2019. *Formally justified and modular Bayesian inference for probabilistic programs.* Ph. D. Dissertation.

[24] 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.

[25] Sam Witty, David Jensen, and Vikash Mansinghka. 2021. SBI: A Simulation-Based Test of Identifiability for Bayesian Causal Inference. *arXiv preprint arXiv:2102.11761* (2021).

[26] Sam A Witty. 2023. *Bayesian Structural Causal Inference with Probabilistic Programming.* Ph. D. Dissertation. University of Massachusetts Amherst.