Typed Abstractions for Causal Probabilistic Programming

Théo Wang* [1] Dario Stein* [2] Eli Bingham [3] Jack Feser [3] Ohad Kammar [4] Michael Lee [5] Jeremy Yallop [5]

[1]University of Oxford [2]Radboud University Nijmegen [3]Basis [4]University of Edinburgh [5]University of Cambridge

TL;DR

Causal Inference [2]: answers questions like "Does X cause Y?" (interventional) or "Given that Y is observed, had X been true, would Y still be true?" (counterfactual).

- Useful in sciences, e.g. clinical trials, climate modelling & Machine Learning.
- Standard treatment based on SCMs: not very expressive!

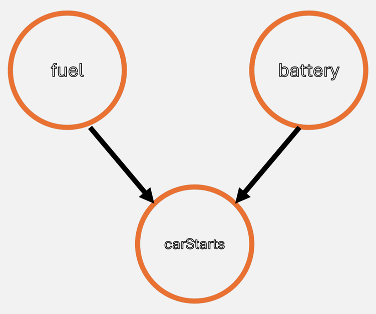
Causal Probabilistic Programming = first class causal primitives on PPLs. (e.g. Multiverse [3], OmegaC [4] or ChiRho [1])

- Useful & more general than SCMs! But practitioner-oriented and semantics often entangled with implementation concerns.

Contribution: Typed version of ChiRho [1], disentangling its abstractions from implementation and thus clearer semantics.
Caveat: Fully Bayesian, so sidestep questions of identifiability!

A Simple Example

Model: D. has a car. It starts if it has both fuel and battery.




```
model :: (MonadDistribution m) => m Bool
model =
    fuel <- bernoulli(0.8)
    battery <- bernoulli(0.9)
    let carStarts = fuel && battery
    return carStarts
```

Query:

Factual World | Counterfactual World

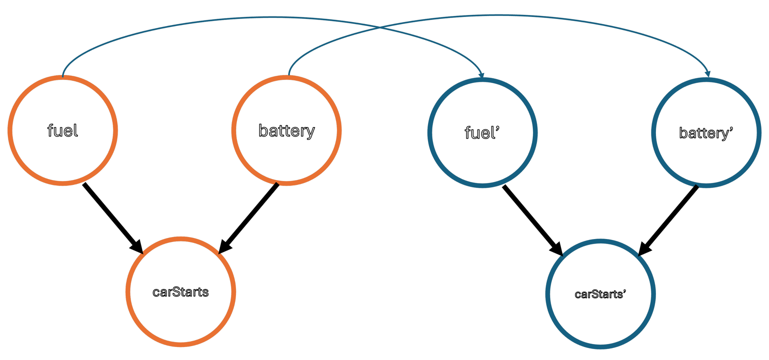
D.'s car didn't start today. Had he refuelled, would the car have started?

Observing the Factual World | **Intervening** on the Counterfactual World | Result of the query

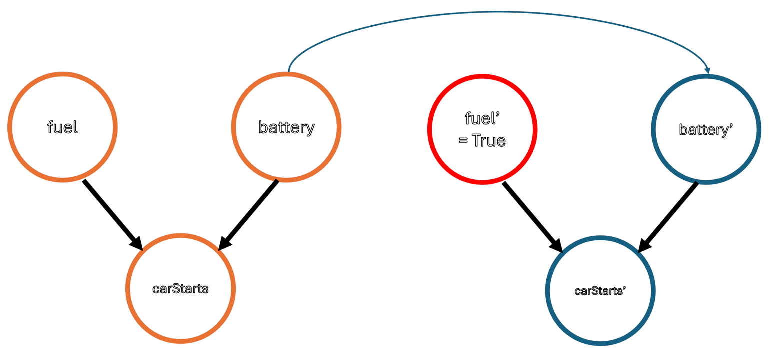
Systematic Causal Reasoning à la Pearl

Pearl: Reduce Causal query to Probabilistic query via program transformations.

1. Twinning [2]: to split the model into two correlated copies/worlds (factual & counterfactual). Note that here both `fuel` and `battery` are treated as exogenous: they are shared between the two worlds. Result:



2. Intervention on the counterfactual world do(fuel′ = True): forcefully set the value of `fuel` to `True`. Equivalent to the intuition of *going back in time, forcing D. to refuel his car while keeping everything else the same*. Result:



3. Conditioning on the factual world using the PPL primitive. Result:

```
fuel <- 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'
```

And indeed, running inference on this program gives us the right answer: around 64%. Success!

Problem statement

As shown above, Pearl's *intervention* and *twinning* can be presented as program transformations. But program transformations can be subtle to deal with.

How can we build a language where causal models are first class and where these transformations can be performed **automatically, compositionally and safely**?

Implementation & more

Our implementation can be found here: github.com/causal-ppl/chirho-haskell. We have many more examples there, incl the ChiRho tutorial, mediation analysis, etc. to showcase the expressivity of our language.

Our solution: Typed & Automatic Causal Reasoning à la ChiRho

We distill the core functionality of ChiRho [1] into a Haskell library. We can then solve the example as follows:

Model code (in red: results of running the model under the intervention at line 4 of inference code. Read this column and come back to this.)

```
1 model :: MonadDistribution m =>
2     InterventionPoint m Bool -> Caus m (MVal Bool)
3 model fuelPt = do
4   fuel <- sample (pure (bernoulli 0.8));      -- ([], (): f), f ~ bernoulli 0.8
5   fuelInt <- new_ fuelPt fuel; -- Intervention point  -- (["fuelTrue"], (0): f, (1): True)
6   battery <- sample (pure (bernoulli 0.9));  -- ([], (): b), b ~ bernoulli 0.9
7   let carStarts = ((&&) <$> fuelInt) <*> battery;
8   return carStarts     -- (["fuelTrue"], (0): f && b, (1): True && b)
```

Inference code:

```
1 fuelPt :: InterventionPoint Bool <- createKey -- (impl detail)
2 let intervenedConditionedModel :: Caus m Bool = do
3   -- attach intervention instructions for `fuelPt`
4   carStarts <- do_ fuelPt (Value True) "fuelTrue" (model fuelPt);
5   -- condition on the factual value of carStarts
6   condition (getFactual carStarts == False);
7   -- return its counterfactual value
8   return (getCounterfactual carStarts);
9 -- Run inference
10 avg <- infer (run intervenedConditionedModel)
11 print avg  -- approx 0.64
```

The Caus type: Interventions as a reader monad

Slogan: Instead of transforming the program, transform the program's environment.

- **Key idea: intervention point**: typed identifier of program locations where we can insert interventions (see line 5 of model code).
- Causal model = suspended computation which runs when provided with interventions for each intervention point.

$$\mathrm{Caus}(X) \triangleq \mathsf{IntvEnv} \to \mathsf{Prob}(X)$$

  where $\mathsf{IntvEnv}$ is a map associating each intervention point $\mathsf{InterventionPoint}(X)$ to the intervention instructions inserted at that point $\mathsf{List}(\mathsf{IntvInstr}(X))$.
- **Fact:** do-intervention = *transformation of environments* that inserts the intervention instruction into the entry in the environment corresponding to the intervention point.

$$\mathsf{do} \;:\; \mathsf{InterventionPoint}(X) \to \mathsf{IntvInstr}(X) \to (\mathsf{IntvEnv} \to \mathsf{IntvEnv})$$

  And we can apply the intervention to a model by precomposition.

$$\mathsf{IntvEnv} \xrightarrow{\text{do fuelPt instr}} \mathsf{IntvEnv} \xrightarrow{\text{model fuelPt}} \mathsf{Prob}(X)$$

The MVal type: Automatic Twinning using applicatives

Strategy: eagerly compute all worlds of interest at the same time.

- **Which worlds?** We generate them from interventions.
  - 1 intervention = 2 worlds: factual world (0), counterfactual world (1).
  - 2 interventions = 4 worlds: fully factual (00), one of them applied (10)/(01), both applied (11).
  - In general, n interventions = 2^n worlds.

  Thus, each intervention (instruction) will act as a binary branching point. We give a branching point name to each instruction $\mathsf{IntvInstr}(X) = \mathsf{Intv}(X) \times \mathsf{Names}$.
- **How to compute at the same time?** Multivalues.

$$\mathsf{MVal}(X) \triangleq \underbrace{\prod_{N \subseteq_{\mathsf{fin}} \mathsf{Names}}}_{\text{Branching point names}} (2^N \to X)$$

  For each generated world (choice of factual (0) or counterfactual (1) per branching point), an X value.
- Examples of multivalues:
  - $M_1 = ([], \{() : x\})$: an unbranched multivalue with value x.
  - $M_2 = ([b], \{(0) : y_0, (1) : y_1\})$: a multivalue depending on an upstream intervention b: if b isn't applied (branch 0), then it takes value y_0; if b is applied (branch 1), then it takes value y_1.
- **How to share variables between worlds?** Applicative structure of **MVal** (≈ broadcasting).

$$f : X \to Y \to Z \vdash \mathbf{pure}(f) \,\langle\star\rangle\, M_1 \,\langle\star\rangle\, M_2 = ([b], \{(0) : f(x, y_0), (1) : f(x, y_1)\})$$

Acknowledgement

Théo's research was generously funded by the Clarendon Scholarship and Oxford-St Peter's College Foundation Scholarship. We thank Sam Staton for useful discussions on this work.

References

[1] *ChiRho*. `https://basisresearch.github.io/chirho`. Accessed: 2025-10-30.
[2] Judea Pearl. *Causality*. Cambridge University Press, 2009.
[3] Yura Perov et al. "Multiverse: causal reasoning using importance sampling in probabilistic programming". In: *Symposium on advances in approximate bayesian inference*. PMLR. 2020, pp. 1–36.
[4] Zenna Tavares et al. "A language for counterfactual generative models". In: *International conference on machine learning*. PMLR. 2021, pp. 10173–10182.

 theo.wang@cs.ox.ac.uk