Typed Abstractions for Causal Probabilistic Programming

Théo Wang*, Dario Stein*, Eli Bingham, Jack Feser, Ohad Kammar, Michael Lee, Jeremy Yallop

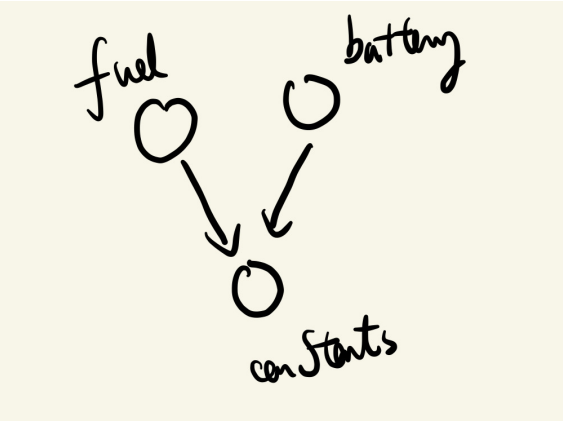
Our work in context

- **Causal Inference (Pearl)**
  - “Does X cause Y”
  - “Given that we saw Y, had X not been the case, would Y still happen?”
  - Wide applications in empirical sciences (e.g. climate modelling, clinical trials etc.) + Machine learning
- **Causal Probabilistic Programming**
  - Fully-fledged PPLs + First class causal primitives
  - E.g. MultiVerse (Perov et al 2020), OmegaC (Tavares 2021), **ChiRho** (Basis)
- **Contribution**: ChiRho as a Haskell library
  - Disentangled its dependence on effect handlers and vectorisation
  - Typed abstractions
  - Clarified its semantics

Overview

- Crash course in causal inference à la Pearl with an example.
- Using typed abstractions à la ChiRho to solve this example.

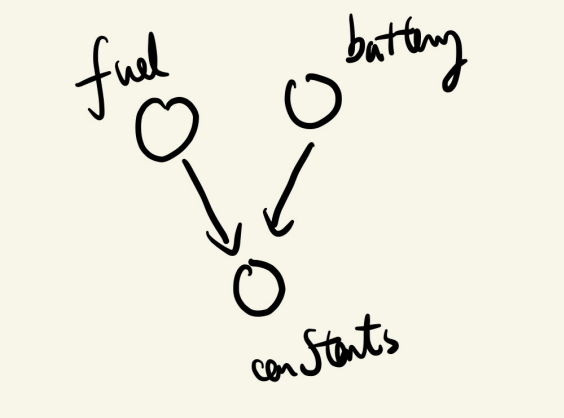
A simple example: Dario’s Car




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

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

A simple example: Dario's Car




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

Factual World

Counterfactual World

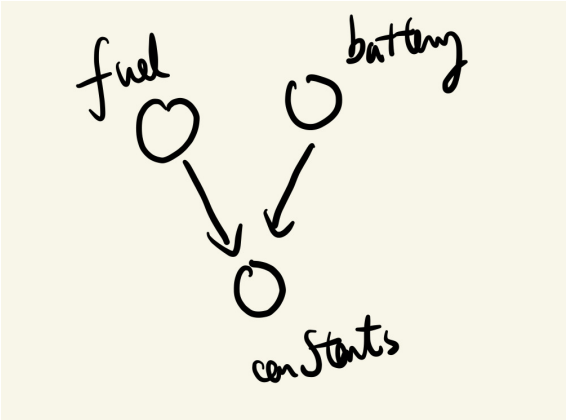
Dario's car didn't start today. Had he refuelled, would the car have started?

Observing the Factual World

Intervening on the Counterfactual World

Result of the query

Solving the problem *à la Pearl*



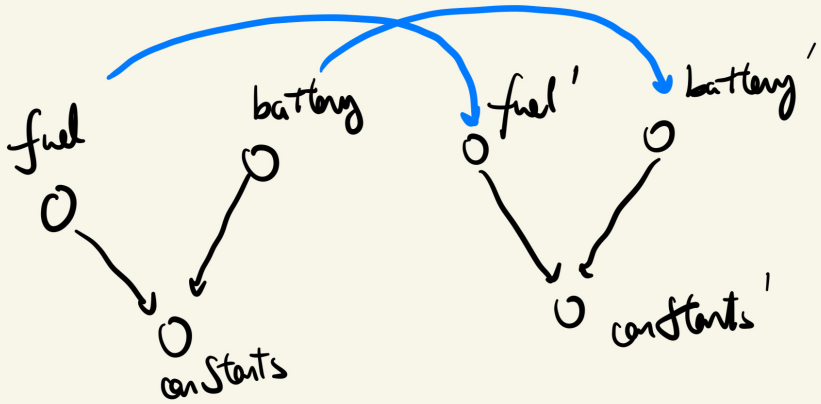

```
model :: (MonadDistribution m) => m Bool
model =
    fuel <- bernoulli(0.8)
    battery <- bernoulli(0.9)
    let carStarts = fuel && battery
    return carStarts
```

Factual World

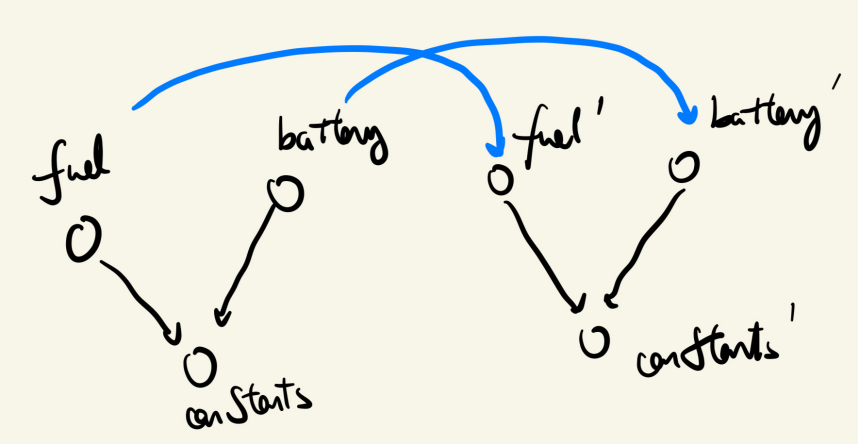
Counterfactual World

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

(1) Twinning (e.g. Jacobs ’19, Pearl ’09) to split the model into two correlated copies/**worlds** (factual & counterfactual)




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




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

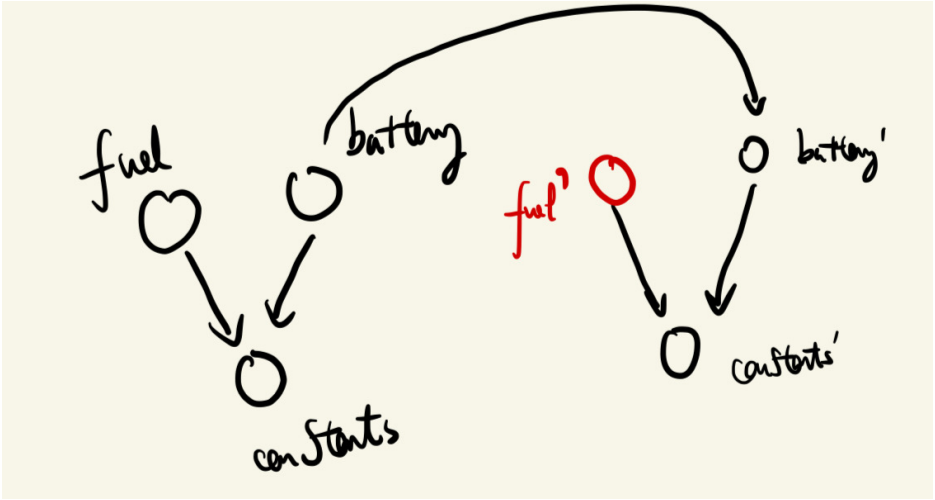
Dario's car didn't start today. Had he refuelled, would the car have started?

Observing the Factual World

Intervening on the Counterfactual World

Result of the query

(2) **Intervening** with **do(fuel'=True)** and **observing** carStarts

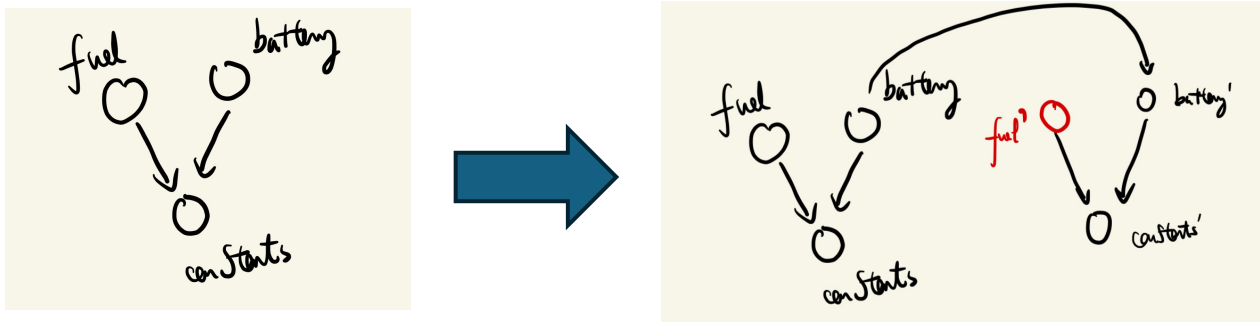



```
fuel <- bernoulli(0.8)
battery <- bernoulli(0.9)
let fuel' = True -- do-intervention
let battery' = battery
let carStarts = fuel && battery
let carStarts' = fuel' && battery'
condition(carStarts == False) -- conditioning
return carStarts'
```

→ About 64%

So far...

Causal operations as program transformations



First class causal primitives

+ Types

Typed and automatic causal reasoning à la ChiRho

Implementing Interventions: the Caus monad

- **Intervention points**: typed identifiers of program locations where interventions are allowed

$$\text{Caus}(X) = \text{InterventionEnv} \to \text{Prob}(X)$$

- **A model** is a suspended computation executed under **intervention instructions** which specify intervention point behaviour.

```
model :: (MonadDistribution m) =>
    InterventionPoint m Bool -> Caus m (...)
model fuelPt =
    fuel <- bernoulli(0.8)
    fuelInt <- new_ fuelPt fuel  -- New intervention point for fuel
    ...
```

Implementing Interventions ct'd

- **Do** intervention as an environment transformation; apply on model by precomposition

$$\mathsf{do} : \mathsf{InterventionPoint}(X) \to \mathsf{InterventionInstr}(X) \to (\mathsf{InterventionEnv} \to \mathsf{InterventionEnv})$$

$$\mathsf{model} : \mathsf{Caus}(X) = \mathsf{InterventionEnv} \to \mathsf{Prob}(X)$$

$$\mathsf{InterventionEnv} \xrightarrow{\mathsf{do\ ipt\ instr}} \mathsf{InterventionEnv} \xrightarrow{\mathsf{model}} \mathsf{Prob}(X)$$

Implementing Twinning (briefly)

- **Strategy:** eagerly compute all worlds of interest at the same time.
- **Which worlds?**: determined by interventions
- **How?** Multivalue applicative
- Check out my poster!

Putting it together

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

```haskell
model :: MonadDistribution m =>
    InterventionPoint m Bool -> Caus m (MVal Bool)
model fuelPt = do
  fuel <- sample (pure (bernoulli 0.8));
  fuelInt <- new_ fuelPt fuel;
  battery <- sample (pure (bernoulli 0.9));
  let carStarts = ((&&) <$> fuelInt) <*> battery;
  return carStarts
```

Model code

Inference code

```haskell
fuelPt :: InterventionPoint Bool <- createKey -- (impl detail)
let intervenedConditionedModel :: Caus m Bool = do
  -- attach intervention instructions for ‘fuelPt‘
  carStarts <- do_ fuelPt (Value True)
                   "fuelTrue" (model fuelPt);
  -- condition on the factual value of carStarts
  condition (getFactual carStarts == False);
  -- return its counterfactual value
  return (getCounterfactual carStarts);
-- Run inference
avg <- infer (run intervenedConditionedModel)
print avg  -- approx 0.64
```

Outlook

- Naively, Pearl's intervention and twinning are program transformations.
- But we can do them automatically and safely using Typed ChiRho (https://github.com/causal-ppl/chirho-haskell)
  - Interventions = reader monad
  - Counterfactuals à la ChiRho = applicative
  - More detail on the poster 🥺
- Highly expressive (implementation has more examples!)
- Towards semantics for Causal PPLs?

Backup slides

Haskell ChiRho vs Python ChiRho

Haskell ChiRho	Python ChiRho
Intervention points	Pyro's string names of sample statements
Do interventions	The do handler in ChiRho (with some difference in the order in which intervention instructions are run)
MVal type	PyTorch tensors
Intervention names	Named tensor dimensions generated and managed by ChiRho and its handlers, e.g. MultiWorldCounterfactual
Applicative structure	Tensor broadcasting
sample: MVal Caus a -> Caus MVal a	Vectorised sampling

Implementing Twinning (Counterfactuals)

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

$$\mathsf{MVal}(X) \triangleq \coprod_{N \subseteq_{\mathrm{fin}} \mathsf{Names}} (2^N \to X)$$

Branching points

For each generated world (choice of factual (0) or counterfactual (1) per branching point), an X value.

E.g. $M_1 = ([], \{() \mapsto a\})$ $M_2 = ([b], \{(0) \mapsto a_0, (1) \mapsto a_1\})$

Which Worlds are of interest?

2 Interventions

1. **do(A=1)**
2. **do(B=2)**

4 Worlds of interest

- Fully Factual
- 1 is applied
- 2 is applied
- Both are applied

Sharing information across worlds via broadcasting

$$\mathbf{pure} : X \to \mathsf{MVal}(X)$$

$$\langle * \rangle : \mathsf{MVal}(X \to Y) \to \mathsf{MVal}(X) \to \mathsf{MVal}(Y)$$

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

Graded monad

$$\mathsf{Caus}\ [A_1 ... A_n]\ X \triangleq \prod_i \mathsf{List}(\mathsf{InterventionInstr}(\mathsf{A_i})) \to P(X)$$