A model of Stochastic Memoization & Name Generation in Probabilistic Programming

Younesse Kaddar, Sam Staton
University of Oxford

MFPS 2023

UNIVERSITY OF
OXFORD

Summary

1. Deterministic vs stochastic memoization
2. Stochastic memoization: Clustering example
3. Stochastic memoization equations
4. Dataflow property
5. Minimal probabilistic language
6. Operational semantics
7. Denotational semantics
8. Soundness & Haskell implementation

Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

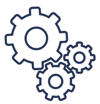
`x`	`is_prime x`
$2^{82589933} - 1$	???

Deterministic Memoization

<u>Idea</u>: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

`x`	`is_prime x`
$2^{82589933} - 1$	TRUE
9	???

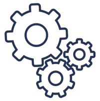


Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

x	is_prime x
$2^{82589933} - 1$	TRUE
9	FALSE
57	???




Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

`x`	`is_prime x`
$2^{82589933} - 1$	TRUE
9	FALSE
57	TRUE
$2^{82589933} - 1$	???

Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

x	is_prime x
$2^{82589933} - 1$	TRUE
9	FALSE
57	TRUE
$2^{82589933} - 1$	**TRUE**

Deterministic Memoization

Idea: **store** the result when a function is applied to an argument, **reuse it later** when the same call is made

Memo functions & machine learning
Donald Milchie, Nature 1968

`x`	`is_prime x`
$2^{82589933} - 1$	TRUE
9	FALSE
57	TRUE
$2^{82589933} - 1$	**TRUE**

- Dynamic programming (variation on laziness)
- Over pure functions: “just” a speed-up (no semantic change)

⚠️ Beware of recursion!

Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

```haskell
f :: Double → Prob Double
f _ = uniform
```

x	f x
0.1	0.3364

https://lazyppl-team.github.io/

Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

```
f :: Double → Prob Double
f _ = uniform
```

x	f x
0.1	0.8364
0.3	0.5468

https://lazyppl-team.github.io/

Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

```
f :: Double → Prob Double
f _ = uniform
```

x	f x
0.1	0.8364
0.3	0.5468
0.1	0.3484

https://lazyppl-team.github.io/

Stochastic Memoization

With probability: **changes** the semantics!

```
f :: Double → Prob Double
f _ = uniform

whiteNoise :: Prob (Double → Double)
whiteNoise = memoize f
```

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

```haskell
f :: Double → Prob Double
f _ = uniform

whiteNoise :: Prob (Double → Double)
whiteNoise = memoize f
```

x	f x
0.1	0.8364
0.3	0.5468
0.1	0.8364

https://lazyppl-team.github.io/

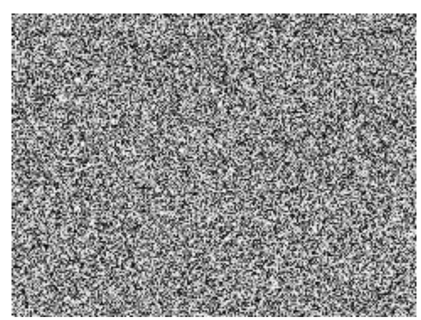
Stochastic Memoization

With probability: **changes** the semantics!

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

```
f :: Double → Prob Double
f _ = uniform

whiteNoise :: Prob (Double → Double)
whiteNoise = memoize f
```

white noise



Stochastic Memoization: Type

With probability: **changes** the semantics!

```haskell
memoize :: (a → Prob b) → Prob (a → b)
```

Stochastic Memoization: Example

Non-parametric clustering (Dirichlet Process)

- Objective: Cluster MFPS attendees based on their preferences for different areas of mathematics and computer science.
- Dataset: Conference attendees and their preferences for various areas of mathematics and computer science
  - `True` for liking an area, `False` for disliking
- Goal: Identify distinct groups of attendees with similar preferences.

Stochastic Memoization: Example

Non-parametric clustering (Dirichlet Process)

1. Randomly decide which proportion of individuals are in each cluster.

Stochastic Memoization: Example

Non-parametric clustering (Dirichlet Process)

1. Randomly decide which proportion of individuals are in each cluster.
2. Assign a unique identifier to each cluster, from some space $\mathbb{A}$ of identifiers.

   Dirichlet process with a **diffuse** base measure on $\mathbb{A}$

   (eg. normal distribution on $\mathbb{A} \stackrel{\text{def}}{=} \mathbb{R}$)

Stochastic Memoization: Example

Non-parametric clustering (Dirichlet Process)

1. Randomly decide which proportion of individuals are in each cluster.
2. Assign a unique identifier to each cluster, from some space $\mathbb{A}$ of identifiers.

   Dirichlet process with a **diffuse** base measure on $\mathbb{A}$

   (eg. normal distribution on $\mathbb{A} \stackrel{\text{def}}{=} \mathbb{R}$)
3. Assign attributes to the cluster identifiers.

   Attributes: Probability distribution over the fields (Algebra, Geometry, etc.)

   Assignment: sample from a random function in $\mathbb{A} \to$ *Attributes,* by

   memoizing a Markov kernel.

Stochastic Memoization: Example

Non-parametric clustering (Dirichlet Process)

1. Randomly decide which proportion of individuals are in each cluster.
2. Assign a unique identifier to each cluster, from some space $\mathbb{A}$ of identifiers.

   Dirichlet process with a **diffuse** base measure on $\mathbb{A}$

   (eg. normal distribution on $\mathbb{A} \stackrel{\text{def}}{=} \mathbb{R}$)
3. Assign attributes to the cluster identifiers.

   Attributes: Probability distribution over the fields (Algebra, Geometry, etc.)

   Assignment: sample from a random function in $\mathbb{A} \to$ *Attributes,* by

   memoizing a Markov kernel.
4. Bayesian clustering: start with steps (1)-(3) as a reasonable *prior*, and

   combine it with observed data to arrive at a *posterior*.

Stochastic Memoization: Example

```haskell
data MFPSAttendees = Alice | Bob | Charlie
  deriving (Eq, Ord, Show, Enum, Bounded)

data Areas = Algebra | Geometry | Topology | Algorithms | PLT
  deriving (Eq, Ord, Show, Enum, Bounded)

type 𝔸 = Double

dataset :: MFPSAttendees → Areas → Bool
dataset Alice Algebra = True
dataset Alice Geometry = True
dataset Alice _ = False
dataset Bob Algorithms = True
dataset Bob PLT = True
dataset Bob _ = False
dataset Charlie Topology = True
dataset Charlie _ = False
```

https://lazyppl-team.github.io/

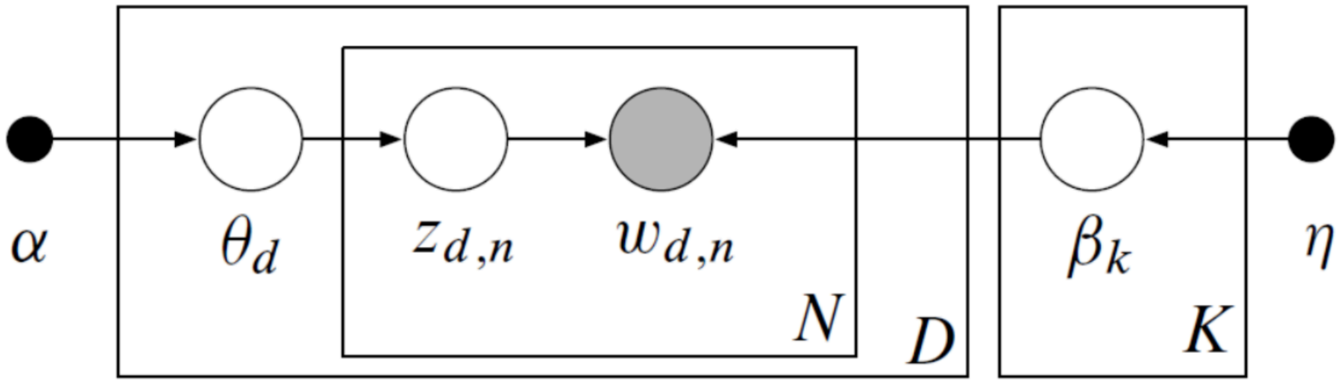
Stochastic Memoization: Example

```
clusterAttributes :: 𝔸 → Prob (Areas → Double)
clusterAttributes x = do
  (ps :: [Double]) ← dirichlet [0.1, 0.1, 0.1, 0.1, 0.1]
  return $ \a → ps !! fromEnum a

memoClusterAttributes :: Prob (𝔸 → Areas → Double)
memoClusterAttributes = memoize clusterAttributes

clusterModel :: Meas [(MFPSAttendees, 𝔸, Areas → Double)]
clusterModel = do
  (probClusters :: Prob 𝔸) ← sample (dp α uniform :: Prob (Prob 𝔸))
  (attributes :: 𝔸 → Areas → Double) ← sample memoClusterAttributes
  let mathematicians = [Alice .. Charlie]
  forM mathematicians $ \m → do
    cluster ← sample probClusters
    let (probAreas :: Areas → Double) = attributes cluster
    forM_ [Algebra .. PLT] $ \a → do
      score $ if dataset m a then probAreas a else 1 - probAreas a
    return (m, cluster, probAreas)
```

Stochastic Memoization: Example

Going further: Latent Dirichlet Allocation (LDA)

e.g. Blei et al. NeurIPS 2002.



- α and η are parameters of the prior distributions over θ and β
- θ_d is the distribution of topics for document d (real vector of length K)
- β_k is the distribution of words for topic k (real vector of length V)
- $z_{d,n}$ is the topic for the nth word in the dth document
- $w_{d,n}$ is the nth word of the dth document

courtesy: https://personal.utdallas.edu/~nrr150130/cs6347/2019sp/lects/Lecture_18_LDA.pdf

Stochastic Memoization

With probability: **changes** the semantics!

Key for

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

- infinite-dimensional structures in **non-parametrics**

  (iid sequences, Gaussian process, CRP, IBP, etc)

Stochastic Memoization

With probability: **changes** the semantics!

Key for

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

- infinite-dimensional structures in **non-parametrics**

  (iid sequences, Gaussian process, CRP, IBP, etc)

- **representation theorems**

  (à la de Finetti, Aldous-Hoover, etc)

Stochastic Memoization

With probability: **changes** the semantics!

Key for

e.g.
Roy, Mansinghka, Goodman, Tenenbaum, NPB 2008
Wood, Archambeau, Gasthaus, James, Teh. ICML 2009.

- infinite-dimensional structures in **non-parametrics**

  (iid sequences, Gaussian process, CRP, IBP, etc)

- **representation theorems**

  (à la de Finetti, Aldous-Hoover, etc)

- in practical **probabilistic programming**

  (Church, WebPPL, Hansei, BLOG, etc)

Stochastic Memoization

Finite Domain a

No problem!

```haskell
memoize :: (Bool → Prob b) → Prob (Bool → b)
memoize (f :: Bool → Prob b) = do
  fTrue ← f True
  fFalse ← f False
  return (\case
    True → fTrue
    False → fFalse)
```

$$(\mathrm{bool} \to \mathrm{Prob}\ \mathrm{b}) \cong (\mathrm{Prob}\ \mathrm{b}, \mathrm{Prob}\ \mathrm{b}) \xrightarrow{\text{double-strength}} \mathrm{Prob}\ (\mathrm{b}, \mathrm{b}).$$

https://lazyppl-team.github.io/

Stochastic Memoization

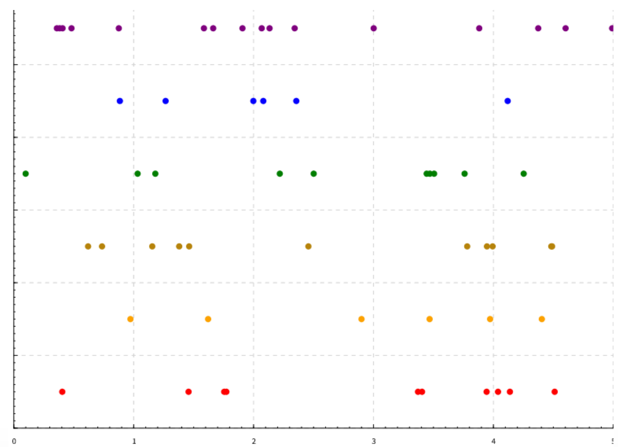
Enumerable Domain a

Laziness trick

Example: Poisson Point Process

Sample all the (exp distributed) interoccurrence times at once

⟶ *cf. Alex Simpson's Talk*



```haskell
poissonPPMemo :: Double → Double → Prob [Double]
poissonPPMemo lower rate = do
  intervals ← memoize $ \(_ :: Int) → exponential rate
  return $ scanl (+) lower $ map intervals [1..]
```

https://lazyppl-team.github.io/

Stochastic Memoization

Non-Enumerable Domain?

Open problem!

Stochastic Memoization

Non-Enumerable Domain?

Def:

Let `a` be an object with an equality predicate `(a,a)→bool`

A ***diffuse distribution*** is a term `p` such that:

```
do {x ← p ; y ← p ; return (x == y)}
```

is semantically equal to

```
return False
```

NB: *equations satisfied by a diffuse probability distribution = equations satisfied by name generation*

Memoization equations

<u>Notation</u>: *if f is of the form $\lambda x.\, u$, memoize (f) is written $\lambda_{מ} x.\, u$.*

$$\begin{array}{l} \mathsf{f} \leftarrow \lambda_{מ} x.\, u \\ \mathsf{f}\ \ \mathsf{n} \end{array} \quad \overset{\text{one sample}}{=} \quad u[\mathsf{n}/x]$$

$$\begin{array}{l} \mathsf{f} \leftarrow \lambda_{מ} x.\, u \\ \mathsf{v}_1 \leftarrow \mathsf{f}\ \ \mathsf{n} \\ \mathsf{w} \leftarrow \mathsf{f}\ \ \mathsf{m} \\ \mathsf{v}_2 \leftarrow \mathsf{f}\ \ \mathsf{n} \\ \mathbf{return}\ (\mathsf{v}_1,\ \mathsf{w},\ \mathsf{v}_2) \end{array} \quad \overset{\text{several samples}}{=} \quad \begin{array}{l} \mathsf{v} \leftarrow u[\mathsf{n}/x] \\ \mathsf{w} \leftarrow u[\mathsf{m}/x] \\ \mathbf{return}\ (\mathsf{v},\ \mathsf{w},\ \mathsf{v}) \end{array}$$

Memoization law

<u>Def</u>: A strong monad Prob is said to *support stochastic memoization of type* a→b
if it is equipped with a morphism
memoize :: (a → Prob b) → Prob (a → b)
satisfying:

```
memoize f = do
  y0 ← f x0
  fMem ← memoize f
  return (\x → if x==x0 then y0 else fMem x)
```

Memoization law

```
memoize f = do
  y₀ <- f x₀
  fMem <- memoize f
  return (\x -> if x==x₀ then y₀ else fMem x)
```

⛔ Problem:

The intuitive implementation uses **state**!

Memoization law

```
memoize f = do
  y0 ← f x0
  fMem ← memoize f
  return (\x → if x==x0 then y0 else fMem x)
```

⛔ Problem:

The intuitive implementation uses **state**!

Memoization law

```haskell
memoize f = do
  y0 <- f x0
  fMem <- memoize f
  return (\x -> if x==x0 then y0 else fMem x)
```

⛔ Problem:

The intuitive implementation uses **state**!

BUT

we want (non-negotiable):

the **dataflow property**

Memoization law

```
memoize f = do
  y0 ← f x0
  fMem ← memoize f
  return (\x → if x==x0 then y0 else fMem x)
```

⛔ <u>Problem</u>:

The intuitive implementation uses **state**!

BUT

we want (non-negotiable):

the **dataflow property**

Dataflow property

→ *cf. Paolo's/Dario's Talks*

```
do { x ← t; y ← u; return (x,y) }
                =
do { y ← u; x ← t; return (x,y) }
```

```
do { x ← t; return () }
           =
       return ()
```

*Program lines can be **reordered** and **discarded** if dataflow is preserved.*

Kock, Synthetic Measure Theory, 2011

$\Longleftrightarrow$

*The monad is **commutative** and **affine**.*

$\Longleftrightarrow$

*The Kleisli category is a **semi-cartesian monoidal** category.*

e.g. Cho, Jacobs MSCS 2019
Fritz, Adv Math 2021 (Markov categories)

$\Longleftrightarrow$

*In probability: **Fubini theorem** and **marginalisation/ normalisation** of probability measures*

Dataflow property

⛔ State **violates** the dataflow property (state monad not commutative)

Dataflow property

⛔ State **violates** the dataflow property (state monad not commutative)

BUT

Memoization **validates** dataflow

- Not *intrinsically* stateful
- *rather*: linked to pure probability theory

ℹ️ *Kolmogorov's extension theorem?*

Kolmogorov extension

In traditional probability theory:

Kolmogorov extension: TFAE:
a consistent family of finite dimensional probability distributions;
an infinite dimensional probability distribution.

$$
\begin{tikzcd}
\mathbb{R} & \mathbb{R} \times \mathbb{R} \arrow[l, "\text{first projection}"'] & \mathbb{R} \times \mathbb{R} \times \mathbb{R} \arrow[l] & \cdots \arrow[l] \\
& & & \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \ldots \arrow[lllu] \arrow[llu] \arrow[lu]
\end{tikzcd}
$$

⛔ **Problem:** only give a measure on the product space,
we want ***function spaces***

$\longrightarrow$ **Categorical probability**: Quasi-Borel spaces (cartesian closed)
<u>Unfortunately:</u> QBSes do not support memoization

Heunen, Kammar, Staton, Yang. LICS 2017

Objective

Challenge

Show that the following items are consistent:

- A probabilistic language with the **dataflow property**
- A type $\mathbb{A}$ with a **diffuse probability distribution**
- A type `bool` with **Bernoulli probability distributions**
- A type of functions $\mathbb{A} \to$ `bool` with **function application**
- **Stochastic memoization** of constant Bernoulli functions

Interface

```
-- Atoms (randomly generated fresh names)
new_atom :: 𝔸

-- Function labels:
-- type to be thought of as 𝔸 → Bool
new_function :: 𝔽

-- Application operator making every function memoized:
-- type of a bipartite graph
(@) :: (𝔽, 𝔸) → Bool
```

Language (fine-grained CBV)

Typing judgements

Typed values:

$$\frac{-}{\Gamma \vdash^{v} \mathsf{true} : \mathsf{bool}} \qquad \frac{-}{\Gamma \vdash^{v} \mathsf{false} : \mathsf{bool}} \qquad \frac{-}{\Gamma, x : A, \Gamma' \vdash^{v} x : A} \qquad \frac{\Gamma \vdash^{v} v : A \quad \Gamma \vdash^{v} w : B}{\Gamma \vdash^{v} (v, w) : A \times B}$$

Typed computations:

$$\frac{\Gamma \vdash^{v} v : A}{\Gamma \vdash^{c} \mathsf{return}(v) : A} \qquad \frac{\Gamma \vdash^{c} u : A \quad \Gamma, x : A \vdash^{c} t : B}{\Gamma \vdash^{c} \mathsf{let\ val}\ x \leftarrow u\ \mathsf{in}\ t : B}$$

$$\frac{\Gamma \vdash^{v} v : \mathsf{bool} \quad \Gamma \vdash^{c} u : A \quad \Gamma \vdash^{c} t : A}{\Gamma \vdash^{c} \mathsf{if}\ v\ \mathsf{then}\ u\ \mathsf{else}\ t : A} \qquad \frac{\Gamma \vdash^{v} v : A \times B \quad \Gamma, x : A, y : B \vdash^{c} t : C}{\Gamma \vdash^{c} \mathsf{match}\ v\ \mathsf{as}\ (x, y)\ \mathsf{in}\ t : C}$$

$$\frac{-}{\Gamma \vdash^{c} \mathsf{flip}(\theta) : \mathsf{bool}} \qquad \frac{-}{\Gamma \vdash^{c} \mathsf{fresh}() : \mathbb{A}} \qquad \frac{\Gamma \vdash^{v} v : \mathbb{A} \quad \Gamma \vdash^{v} w : \mathbb{A}}{\Gamma \vdash^{c} (v = w) : \mathsf{bool}}$$

$$\frac{\Gamma, x : \mathbb{A} \vdash^{c} u : \mathsf{bool}}{\Gamma \vdash^{c} \lambda_{\text{מ}} x.\, u : \mathbb{F}} \qquad \frac{\Gamma \vdash^{v} v : \mathbb{F} \quad \Gamma \vdash^{v} w : \mathbb{A}}{\Gamma \vdash^{c} (v @ w) : \mathsf{bool}}$$

Name generation, probabilistic effects, memoization

Operational Semantics

Extended expression typing judgements. Here, $(\mathfrak{f}, a) \notin \Delta \cup \Delta_1 \cup \Delta_2$.

$$\frac{\Gamma \vDash^{\mathsf{c}} u : A}{\Gamma \mid \emptyset \vDash^{\mathsf{c}} u : A} \qquad \frac{\Gamma \mid \Delta \vDash^{\mathsf{c}} u : A}{\Gamma \mid (\mathfrak{f}, a), \Delta \vDash^{\mathsf{c}} \{\!\!\{u\}\!\!\}^{\mathfrak{f},a}_{\gamma} : A} \qquad \frac{\Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma, x : A \mid \Delta_2 \vDash^{\mathsf{c}} t : B}{\Gamma \mid \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{let\ val}\ x \leftarrow u\ \mathsf{in}\ t : B}$$

$$\frac{\Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma, x : A \mid \Delta_2 \vDash^{\mathsf{c}} t : B}{\Gamma \mid (\mathfrak{f}, a), \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{let\ val}\ x \leftarrow \{\!\!\{u\}\!\!\}^{\mathfrak{f},a}_{\gamma}\ \mathsf{in}\ t : B} \qquad \frac{\Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma, x : A \mid \Delta_2 \vDash^{\mathsf{c}} t : B}{\Gamma \mid (\mathfrak{f}, a), \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{let\ val}\ x \leftarrow u\ \mathsf{in}\ \{\!\!\{t\}\!\!\}^{\mathfrak{f},a}_{\gamma} : B}$$

$$\frac{\Gamma \vDash^{\mathsf{v}} v : \mathsf{bool} \quad \Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma \mid \Delta_2 \vDash^{\mathsf{c}} t : A}{\Gamma \mid \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{if}\ v\ \mathsf{then}\ u\ \mathsf{else}\ t : A}$$

$$\frac{\Gamma \vDash^{\mathsf{v}} v : \mathsf{bool} \quad \Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma \mid \Delta_2 \vDash^{\mathsf{c}} t : A}{\Gamma \mid (\mathfrak{f}, a), \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{if}\ v\ \mathsf{then}\ \{\!\!\{u\}\!\!\}^{\mathfrak{f},a}_{\gamma}\ \mathsf{else}\ t : A} \qquad \frac{\Gamma \vDash^{\mathsf{v}} v : \mathsf{bool} \quad \Gamma \mid \Delta_1 \vDash^{\mathsf{c}} u : A \quad \Gamma \mid \Delta_2 \vDash^{\mathsf{c}} t : A}{\Gamma \mid (\mathfrak{f}, a), \Delta_1, \Delta_2 \vDash^{\mathsf{c}} \mathsf{if}\ v\ \mathsf{then}\ u\ \mathsf{else}\ \{\!\!\{t\}\!\!\}^{\mathfrak{f},a}_{\gamma} : A}$$

$$\frac{\Gamma \vDash^{\mathsf{v}} v : A \times B \quad \Gamma, x : A, y : B \mid \Delta \vDash^{\mathsf{c}} t : C}{\Gamma \mid \Delta \vDash^{\mathsf{c}} \mathsf{match}\ v\ \mathsf{as}\ (x, y)\ \mathsf{in}\ t : C} \qquad \frac{\Gamma \vDash^{\mathsf{v}} v : A \times B \quad \Gamma, x : A, y : B \mid \Delta \vDash^{\mathsf{c}} t : C}{\Gamma \mid (\mathfrak{f}, a), \Delta \vDash^{\mathsf{c}} \mathsf{match}\ v\ \mathsf{as}\ (x, y)\ \mathsf{in}\ \{\!\!\{t\}\!\!\}^{\mathfrak{f},a}_{\gamma} : C}$$

Memoization stack Δ and memoization contexts

Operational semantics

Configurations (intuition):

$\langle e,$

x	f0	f1	f2	f3	f4
0	0	1	0	0	1
1	1	1	0	1	0
2	0	0	0	1	0
3	0	1	1	1	1
4	0	1	1	1	0
5	0	1	0	1	1
6	0	1	0	0	0

$\rangle$

expression

memo table +
closure for each column

Sound for our denotational semantics, but works more generally.

Operational Semantics

Terminal computations r, Redexes ρ, and Reduction contexts $\mathcal{C}[-]$		
r	::=	$\mathsf{return}(v) \mid \lambda_{\mathfrak{p}} x.\, u \mid \mathsf{fresh}()$
ρ	::=	$\mathsf{let\ val}\ x \leftarrow r\ \mathsf{in}\ u \mid \{\!\{\mathsf{return}(v)\}\!\}^{\mathfrak{f},a}_{\gamma}$ where $\mathfrak{f} \in \mathfrak{g}_L,\ a \in \mathfrak{g}_R,\ \gamma \in \mathsf{Tree}(2 + \mathfrak{g}_L + \mathfrak{g}_R)^{\mathsf{Var}}$
		$\mid \mathsf{match}\ v\ \mathsf{as}\ (x, y)\ \mathsf{in}\ t \mid \mathsf{if}\ v\ \mathsf{then}\ t\ \mathsf{else}\ u \mid \mathsf{flip}(\theta) \mid (v = w) \mid (v @ w)$

Configurations $(\gamma, u, \mathfrak{g}, \lambda)$

$\gamma \in \mathsf{Tree}(2 + \mathfrak{g}_L + \mathfrak{g}_R)^{\mathsf{Var}}$ is a context value.

u is an extended expression $\Gamma \mid \Delta \Vdash^{\mathsf{c}} u : A$.

$\mathfrak{g} \stackrel{\text{def}}{=} (\mathfrak{g}_L, \mathfrak{g}_R, E)$ is a partial graph.

$\lambda\colon \mathfrak{g}_L \to \mathsf{Closures}$, where $\mathsf{Closures} \stackrel{\text{def}}{=} \left\{(\lambda_{\mathfrak{p}} x.\, u, \gamma) \mid \Gamma \Vdash^{\mathsf{c}} \lambda_{\mathfrak{p}} x.\, u : \mathbb{F} \text{ and } \gamma \in (\!|\Gamma|\!)\right\}$

Enables us to define a small-step operational semantics

Operational Semantics

$$
\begin{array}{lcl}
(\gamma, \mathsf{let\ val}\ x \leftarrow \mathsf{return}(v)\ \mathsf{in}\ u, \mathfrak{g}, \lambda) & \to & (\gamma \sqcup \{x \mapsto (\!|v|\!)_\gamma\}, u, \mathfrak{g}, \lambda) \\
(\gamma, \{\!\!\{\mathsf{return}(v)\}\!\!\}^{(\mathfrak{f},a)}_{\gamma'}, \mathfrak{g}, \lambda) & \to & \begin{cases} (\gamma', \mathsf{return}((\!|v|\!)_\gamma), (\mathfrak{g}_L, \mathfrak{g}_R, E \cup \{\mathfrak{f} \xrightarrow{(\!|v|\!)_\gamma} a\}), \lambda) \\ \qquad\qquad\qquad \text{if } (\!|v|\!)_\gamma \in \{\mathsf{true}, \mathsf{false}\} \\ \text{else, failure (cannot memoize a non-boolean function)} \end{cases} \\
(\gamma, \mathsf{let\ val}\ x \leftarrow \lambda_{\text{מ}} y.\ u\ \mathsf{in}\ t, \mathfrak{g}, \lambda) & \to & \big(\gamma \sqcup \{x \mapsto \mathfrak{f}\}, t, (\mathfrak{g}_L \sqcup \{\mathfrak{f}\}, \mathfrak{g}_R, E \sqcup \{f \xrightarrow{\perp} a\}_{a \in \mathfrak{g}_R}), \\
& & \qquad \lambda \sqcup \{\mathfrak{f} \mapsto (\lambda_{\text{מ}} y.\ u, \gamma)\}\big) \\
(\gamma, \mathsf{let\ val}\ x \leftarrow \mathsf{fresh}()\ \mathsf{in}\ t, \mathfrak{g}, \lambda) & \to & \big(\gamma \sqcup \{x \mapsto a\}, t, \\
& & \qquad (\mathfrak{g}_L, \mathfrak{g}_R \sqcup \{a\}, \mathfrak{g}_R, E \sqcup \{\mathfrak{f} \xrightarrow{\perp} a\}_{\mathfrak{f} \in \mathfrak{g}_L}), \lambda\big) \\
(\gamma, (v @ w), \mathfrak{g}, \lambda) & \to & \begin{cases} (\gamma, \mathsf{return}(\beta), \mathfrak{g}, \lambda) \quad \text{if } \beta \stackrel{\text{def}}{=} E((\!|v|\!)_\gamma, (\!|w|\!)_\gamma) \neq \perp \\ (\gamma_0 \sqcup \{y \mapsto (\!|w|\!)_\gamma\}, \{\!\!\{u\}\!\!\}^{\mathfrak{f},a}_{\gamma}, \mathfrak{g}, \lambda) \quad \text{else,} \\ \qquad\qquad \text{where } \lambda((\!|v|\!)_\gamma) \stackrel{\text{def}}{=} (\lambda_{\text{מ}} y.\ u, \gamma_0) \end{cases} \\
(\gamma, v = w, \mathfrak{g}, \lambda) & \to & (\gamma, \mathsf{return}(\beta), \mathfrak{g}, \lambda) \text{ where } \beta \stackrel{\text{def}}{=} ((\!|v|\!)_\gamma = (\!|w|\!)_\gamma) \\
(\gamma, \mathsf{flip}(\theta), \mathfrak{g}, \lambda) & \xrightarrow{\text{with proba. } \theta} & (\gamma, \mathsf{return}(\beta), \mathfrak{g}, \lambda) \quad \text{where } \beta \in \{\mathsf{true}, \mathsf{false}\} \\
(\gamma, \mathsf{match}\ v\ \mathsf{as}\ (x, y)\ \mathsf{in}\ t, \mathfrak{g}, \lambda) & \to & (\gamma \sqcup \{x \mapsto (\!|v|\!)_\gamma, y \mapsto (\!|w|\!)_\gamma\}, t, \mathfrak{g}, \lambda) \\
(\gamma, \mathsf{if}\ v\ \mathsf{then}\ t\ \mathsf{else}\ u) & \to & \begin{cases} (\gamma, t, \mathfrak{g}, \lambda) & \text{if } v = \mathsf{true} \\ (\gamma, u, \mathfrak{g}, \lambda) & \text{else, if } v = \mathsf{false} \end{cases}
\end{array}
$$

Operational Semantics

Example

$$\Big(\emptyset,\quad \begin{array}{l}\text{let val } x_0 \leftarrow \text{fresh}() \text{ in}\\ \text{let val } f_1 \leftarrow \lambda_{מ} x.\ (\text{let val } b \leftarrow (x = x_0) \text{ in}\\ \qquad\qquad \text{if } b \text{ then flip}(\tfrac{1}{2}) \text{ else false}) \text{ in}\\ \text{let val } f_2 \leftarrow \lambda_{מ} y.\ f_1@y \text{ in } f_2@x_0,\end{array}\quad (\emptyset,\emptyset,\emptyset),\ \emptyset\Big)$$

$$\rightarrow \Big(\{x_0 \mapsto a_0\},\quad \begin{array}{l}\text{let val } f_1 \leftarrow \lambda_{מ} x.\ (\text{let val } b \leftarrow (x = x_0) \text{ in}\\ \qquad\qquad \text{if } b \text{ then flip}(\tfrac{1}{2}) \text{ else false}) \text{ in}\\ \text{let val } f_2 \leftarrow \lambda_{מ} y.\ f_1@y \text{ in } f_2@x_0,\end{array}\quad (\emptyset,\{a_0\},\emptyset),\ \emptyset\Big)$$

$$\rightarrow^2 \Big(\overbrace{\{x_0 \mapsto a_0,\ f_1 \mapsto \mathfrak{f}_1,\ f_2 \mapsto \mathfrak{f}_2\}}^{\stackrel{\text{def}}{=}\gamma_0},\quad f_2@x_0,\quad (\{\mathfrak{f}_1,\mathfrak{f}_2\},\ \{a_0\},\{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\ \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\quad \begin{array}{l}\{\mathfrak{f}_1 \mapsto (\lambda_{מ} x.\ \text{let val } b \leftarrow (x = x_0) \text{ in}\\ \qquad\qquad \text{if } b \text{ then flip}(\tfrac{1}{2}) \text{ else false}), \{x_0 \mapsto a_0\}),\\ \mathfrak{f}_2 \mapsto (\lambda_{מ} y.\ f_1@y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\end{array}\Big)$$

$$\rightarrow \Big(\overbrace{\{x_0 \mapsto a_0,\ f_1 \mapsto \mathfrak{f}_1,\ y \mapsto a_0\}}^{\stackrel{\text{def}}{=}\gamma_1},\quad \{\!\{f_1@y\}\!\}^{\mathfrak{f}_2,a_0}_{\gamma_0},\quad (\{\mathfrak{f}_1,\mathfrak{f}_2\},\ \{a_0\},\{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\ \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\quad \begin{array}{l}\{\mathfrak{f}_1 \mapsto (\lambda_{מ} x.\ \text{let val } b \leftarrow (x = x_0) \text{ in}\\ \qquad\qquad \text{if } b \text{ then flip}(\tfrac{1}{2}) \text{ else false}), \{x_0 \mapsto a_0\}),\\ \mathfrak{f}_2 \mapsto (\lambda_{מ} y.\ f_1@y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\end{array}\Big)$$

Operational Semantics

Example

$$
\begin{aligned}
\to\Big(&\{x_0 \mapsto a_0,\, x \mapsto a_0\},\\
&\Big\{\!\!\Big\{ \{\!\!\{ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in}\\
&\qquad\quad \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false} \}\!\!\}^{\mathfrak{f}_1, a_0}_{\gamma_1} \Big\}\!\!\Big\}^{\mathfrak{f}_2, a_0}_{\gamma_0},\\
&(\{\mathfrak{f}_1, \mathfrak{f}_2\},\ \{a_0\}, \{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\, \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\\
&\{\mathfrak{f}_1 \mapsto (\lambda_{\text{מ}} x.\ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in}\\
&\qquad\quad \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false}), \{x_0 \mapsto a_0\}),\\
&\mathfrak{f}_2 \mapsto (\lambda_{\text{מ}} y.\ f_1 @ y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\to\Big(&\{x_0 \mapsto a_0,\, x \mapsto a_0,\, b \mapsto \mathsf{true}\},\\
&\Big\{\!\!\Big\{ \{\!\!\{ \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false} \}\!\!\}^{\mathfrak{f}_1, a_0}_{\gamma_1} \Big\}\!\!\Big\}^{\mathfrak{f}_2, a_0}_{\gamma_0},\\
&(\{\mathfrak{f}_1, \mathfrak{f}_2\},\ \{a_0\}, \{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\, \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\\
&\{\mathfrak{f}_1 \mapsto (\lambda_{\text{מ}} x.\ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in}\\
&\qquad\quad \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false}), \{x_0 \mapsto a_0\}),\\
&\mathfrak{f}_2 \mapsto (\lambda_{\text{מ}} y.\ f_1 @ y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\to\Big(&\{x_0 \mapsto a_0,\, x \mapsto a_0,\, b \mapsto \mathsf{true}\},\\
&\Big\{\!\!\Big\{ \{\!\!\{ \mathsf{flip}(\tfrac{1}{2}) \}\!\!\}^{\mathfrak{f}_1, a_0}_{\gamma_1} \Big\}\!\!\Big\}^{\mathfrak{f}_2, a_0}_{\gamma_0},\\
&(\{\mathfrak{f}_1, \mathfrak{f}_2\},\ \{a_0\}, \{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\, \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\\
&\{\mathfrak{f}_1 \mapsto (\lambda_{\text{מ}} x.\ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in}\\
&\qquad\quad \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false}), \{x_0 \mapsto a_0\}),\\
&\mathfrak{f}_2 \mapsto (\lambda_{\text{מ}} y.\ f_1 @ y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\overset{\text{proba. } \frac{1}{2}}{\to}\Big(&\{x_0 \mapsto a_0,\, x \mapsto a_0,\, b \mapsto \mathsf{true}\},\\
&\Big\{\!\!\Big\{ \{\!\!\{ \mathsf{return}(\beta) \}\!\!\}^{\mathfrak{f}_1, a_0}_{\gamma_1} \Big\}\!\!\Big\}^{\mathfrak{f}_2, a_0}_{\gamma_0},\\
&(\{\mathfrak{f}_1, \mathfrak{f}_2\},\ \{a_0\}, \{\mathfrak{f}_1 \xrightarrow{\perp} a_0,\, \mathfrak{f}_2 \xrightarrow{\perp} a_0\}),\\
&\{\mathfrak{f}_1 \mapsto (\lambda_{\text{מ}} x.\ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in}\\
&\qquad\quad \mathsf{if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false}), \{x_0 \mapsto a_0\}),\\
&\mathfrak{f}_2 \mapsto (\lambda_{\text{מ}} y.\ f_1 @ y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\to^2\Big(&\overbrace{\{x_0 \mapsto a_0,\ f_1 \mapsto \mathfrak{f}_1,\ f_2 \mapsto \mathfrak{f}_2\}}^{\overset{\text{def}}{=} \gamma_0},\quad \mathsf{return}(\beta),\quad (\{\mathfrak{f}_1, \mathfrak{f}_2\},\ \{a_0\}, \{\mathfrak{f}_1 \xrightarrow{\beta} a_0,\, \mathfrak{f}_2 \xrightarrow{\beta} a_0\}),\\
&\{\mathfrak{f}_1 \mapsto (\lambda_{\text{מ}} x.\ \mathsf{let\ val}\ b \leftarrow (x = x_0)\ \mathsf{in\ if}\ b\ \mathsf{then\ flip}(\tfrac{1}{2})\ \mathsf{else\ false}), \{x_0 \mapsto a_0\}),\\
&\mathfrak{f}_2 \mapsto (\lambda_{\text{מ}} y.\ f_1 @ y, \{x_0 \mapsto a_0, f_1 \mapsto \mathfrak{f}_1\})\}\Big)
\end{aligned}
$$

Operational Semantics

Proposition:

$$
\begin{array}{l}
\mathsf{f} \leftarrow \lambda_{\text{מ}} x.\, u \\
\mathsf{v}_1 \leftarrow \mathsf{f}\ \ \mathsf{n} \\
\mathsf{w} \leftarrow \mathsf{f}\ \ \mathsf{m} \\
\mathsf{v}_2 \leftarrow \mathsf{f}\ \ \mathsf{n} \\
\mathbf{return}\ (\mathsf{v}_1,\ \mathsf{w},\ \mathsf{v}_2)
\end{array}
\quad \overset{\text{several samples}}{=} \quad
\begin{array}{l}
\mathsf{v} \leftarrow u[\mathsf{n}/x] \\
\mathsf{w} \leftarrow u[\mathsf{m}/x] \\
\mathbf{return}\ (\mathsf{v},\ \mathsf{w},\ \mathsf{v})
\end{array}
$$

have indeed the same (big-step) operational semantics

Probabilistic local state monad

c.f. Plotkin Power FOSSACS 2002.
Kaddar, Staton, MFPS 2023.

Theorem.

$$T(X)(g) \stackrel{\text{def}}{=} \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} \left(X(h) \times [0,1]^{(h-g)_L}\right)\right)^{[0,1]^{g_L}}$$

defines a commutative affine monad.

For all covariant presheaf *X*: *BiGrph*$_{emb}$ → *Set*
and bipartite graph (bigraph) *g*

$$T(X)(g \stackrel{\iota}{\hookrightarrow} g') = \begin{cases} \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L}\right)^{[0,1]^{g_L}} \longrightarrow \left(P_{\mathrm{f}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L}\right)^{[0,1]^{g'_L}} \\ \vartheta \longmapsto \lambda' \mapsto \text{let } \vartheta(\lambda' \iota_L) = \langle \overrightarrow{p} \mid [x_h, \lambda^h]_g \rangle_h \text{ in } \left\langle \overrightarrow{p} \,\middle|\, [X(h \hookrightarrow h \coprod_g g')(x_h), \lambda^h]_{g'} \right\rangle_h \end{cases}$$

Gives a denotational semantics to our language

Probabilistic local state monad

Garbage collection of the coend:

$$\begin{tikzcd}
& X(h) \times [0,1]^{(h'-g)_L} \arrow[dl, "X(h\hookrightarrow h')\times 1"'] \arrow[dr, "1\times(-\circ(h-g)_L\hookrightarrow(h'-g)_L)"] & \\
X(h') \times [0,1]^{(h'-g)_L} \arrow[dr] & & X(h) \times [0,1]^{(h-g)_L} \arrow[dl] \\
& \int^{g\hookrightarrow h} \left(X(h) \times [0,1]^{(h-g)_L}\right) &
\end{tikzcd}$$

Denotational semantics

$$[\![\mathbb{F}]\!] \stackrel{\text{def}}{=} \mathbf{BiGrph}_{emb}(\circ, -) \quad [\![\mathbb{A}]\!] \stackrel{\text{def}}{=} \mathbf{BiGrph}_{emb}(\bullet, -) \quad \text{so that } [\![\mathbb{F}]\!](g) \cong g_L,\ [\![\mathbb{A}]\!](g) \cong g_R.$$

A computation of type:

- $\mathbb{A}$ returns the label of an already existing atom
  or a fresh one with its connections to the already existing functions:

$$T([\![\mathbb{A}]\!])(g) \;\cong\; P_{\mathrm{f}}(g_R + 2^{g_L})^{[0,1]^{g_L}}$$

- $\mathbb{F}$ returns the label of an already existing function
  or creates a new function with its connections to already existing atoms
  and a fixed probabilistic bias:

$$T([\![\mathbb{F}]\!])(g) \;\cong\; P_{\mathrm{f}}(g_L + 2^{g_R} \times [0,1])^{[0,1]^{g_L}}$$

Denotational semantics

$$[\![v@w]\!]_g : [\![\Gamma, v : \mathbb{F}, w : \mathbb{A}]\!](g) \to [0,1]^{[0,1]^{g_L}}$$

$$[\![v@w]\!] \stackrel{\text{def}}{=} 1 \times ([\![\Gamma]\!] \times [\![\mathbb{F}]\!] \times [\![\mathbb{A}]\!]) \xrightarrow{\eta \times (! \times \mathcal{E})} T([\![\mathsf{bool}]\!])^{[\![\mathsf{bool}]\!]} \times [\![\mathsf{bool}]\!] \xrightarrow{\mathrm{ev}} T([\![\mathsf{bool}]\!])$$

$$[\![v = w]\!]_g : [\![\Gamma, v : \mathbb{A}, w : \mathbb{A}]\!](g) \to [0,1]^{[0,1]^{g_L}}$$

$$[\![v = w]\!] \stackrel{\text{def}}{=} [\![\Gamma]\!] \times [\![\mathbb{A}]\!]^2 \cong 1 \times [\![\Gamma]\!] \times \Big(よ(\bullet) + よ(\bullet + \bullet) \Big) \xrightarrow{\eta \times ! \times \left[! \,;\, \iota_{\mathsf{true}},\ ! \,;\, \iota_{\mathsf{false}} \right]} T([\![\mathsf{bool}]\!])^{[\![\mathsf{bool}]\!]} \times [\![\mathsf{bool}]\!] \xrightarrow{\mathrm{ev}} T([\![\mathsf{bool}]\!])$$

$$[\![\mathsf{fresh}()]\!]_g : [\![\Gamma]\!](g) \to T([\![\mathbb{A}]\!])(g)$$

$$[\![\mathsf{fresh}()]\!]_g : \begin{cases} 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \to P_{\mathrm{f}}(g_R + 2^{g_L})^{[0,1]^{g_L}} \\ \_,\ \_,\ \_ \mapsto \lambda \mapsto \left\langle \dfrac{1}{Z} \displaystyle\prod_{\mathfrak{f} \in g_L} \lambda(\mathfrak{f})^{E^h(\mathfrak{f}, a_h(\bullet))} (1 - \lambda(\mathfrak{f}))^{1 - E^h(\mathfrak{f}, a_h(\bullet))} \middle|\, [\underbrace{\bullet \xhookrightarrow{a_h} h}_{\cong (h-g)_R},\ !]_g \right\rangle_{h \in R_g} \end{cases}$$

Denotational semantics

Def: A function $\lambda_{\text{מ}} x.\, u$ is *freshness-invariant* if, for every $g, b^k \in 2^k, \kappa_i \colon \circ \hookrightarrow g$, $\tau_j \colon \bullet \hookrightarrow g$ and $\lambda \in [0,1]^{g_L}$, we have (where $\iota_1,\ \iota_2$ are the coprojections):

$$\forall e \in 2^{g_L},\ [\![u]\!]_g\big(b^k,\ (\circ \xhookrightarrow{\kappa_i} g \xhookrightarrow{\iota_1} g +_e \bullet)_i,\ (\bullet \xhookrightarrow{\tau_j} g \xhookrightarrow{\iota_1} g +_e \bullet)_j,\ \bullet \xhookrightarrow{\iota_2} g +_e \bullet,\ \lambda\big) \text{ is a constant } \tilde{p}_u$$

$$[\![\lambda_{\text{מ}} x.\, u]\!]_g \colon \begin{cases} 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \to P_{\mathrm{f}}(g_L + 2^{g_R} \times [0,1])^{[0,1]^{g_L}} \\[2ex] b^k, \begin{matrix} (\circ \xhookrightarrow{\kappa_i} g)_i, \\ (\bullet \xhookrightarrow{\tau_j} g)_j \end{matrix} \mapsto \lambda \mapsto \left\langle \frac{1}{Z} \prod_{a \in g_R} p_a^{E^h(\mathfrak{f}_h(\circ), a)} (1 - p_a)^{1 - E^h(\mathfrak{f}_h(\circ), a)} \,\Big|_{\cong\, (h-g)_L} \big[\underbrace{\circ}_{} \xhookrightarrow{\mathfrak{f}_h} h,\ \_ \mapsto \tilde{p}_u\big]_g \right\rangle_{h \in L_g} \end{cases}$$

where $p_a \stackrel{\text{def}}{=} [\![u]\!]_g\big(b^k,\ (\circ \xhookrightarrow{\kappa_i} g)_i,\ (\bullet \xhookrightarrow{\tau_j} g)_j,\ \bullet \xhookrightarrow{a} g,\ \lambda\big)$ for every $a \in g_R$, and $\tilde{p}_u$ is as in Def. 5.8. As a result,

Theorem.
The probabilistic local state monad T supports stochastic memoization for freshness-invariant functions (so, in particular, constant Bernoulli functions).

Soundness

Configurations of the form $(\gamma, e, \mathfrak{g}, \lambda)$, where e is of type A, can be denotationally interpreted as

$$[\![(\gamma, e, \mathfrak{g}, \lambda)]\!] \stackrel{\text{def}}{=} \sum_{\tilde{e} \in 2^{U_{\mathfrak{g}}}} \prod_{(\mathfrak{f}, a) \in U_{\mathfrak{g}}} \lambda(\mathfrak{f})^{\tilde{e}(\mathfrak{f},a)} \big(1 - \lambda(\mathfrak{f})\big)^{1 - \tilde{e}(\mathfrak{f},a)} [\![u]\!]_{\mathfrak{g}_{\tilde{e}}}(\gamma, \lambda) \in T(A)_{\mathfrak{g}_{\tilde{e}}}(\gamma)(\lambda)$$

where $U_{\mathfrak{g}} \stackrel{\text{def}}{=} \big\{(\mathfrak{f}, a) \mid E(\mathfrak{f}, a) = \bot\big\} \subseteq \mathfrak{g}_L \times \mathfrak{g}_R$ and $\mathfrak{g}_{\tilde{e}}$ extends $\mathfrak{g}$ according to $\tilde{e}$: $E(\mathfrak{f}, a) = \tilde{e}(\mathfrak{f}, a)$ for all $(\mathfrak{f}, a) \in U_{\mathfrak{g}}$.

Theorem.

The denotational semantics is sound

with respect to

the operational semantics.

$$[\![(\gamma, e, \mathfrak{g}, \lambda)]\!] \cong \sum_{\substack{(\gamma, e, \mathfrak{g}, \lambda) \to (\gamma', e', \mathfrak{g}', \lambda') \\ \textit{with proba. } p}} p \cdot [\![(\gamma', e', \mathfrak{g}', \lambda')]\!]$$

Haskell Toy Implementation

```haskell
smallStep :: Expr a -> EnvVal -> T (ExprOrValue a)

bigStep :: Expr a -> EnvVal -> T (Value a)

den :: Expr a -> EnvVal -> T (Value a)
```

https://github.com/youqad/stochastic-memoization-implementation

Conclusion

Challenge ✅

Show that the following items are consistent:

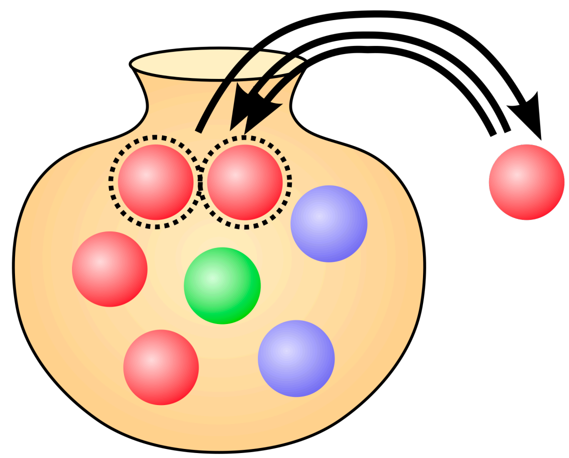
- A probabilistic language with the **dataflow property** ✅
- A type $\mathbb{A}$ with a **diffuse probability distribution** ✅
- A type `bool` with **Bernoulli probability distributions** ✅
- A type of functions $\mathbb{A} \to$ `bool` with **function application** ✅
- **Stochastic memoization** of constant Bernoulli functions ✅

Summary

1. Deterministic vs stochastic memoization
2. Stochastic memoization: Clustering example
3. Stochastic memoization equations
4. Dataflow property
5. Minimal probabilistic language
6. Operational semantics
7. Denotational semantics
8. Soundness & Haskell implementation

Pólya's Urn and de Finetti

Staton, Stein, Yang, Ackermann, Freer, Roy, ICALP 2018



```
class BinProcess hyperparam process where
  new :: hyperparam → Prob process
  get :: process → Prob Bool
```

courtesy: https://commons.wikimedia.org/wiki/File:Urn_problem_qtl5.svg

Pólya's Urn and de Finetti

Staton, Stein, Yang, Ackermann, Freer, Roy, ICALP 2018

```haskell
class BinProcess hyperparam process where
  new :: hyperparam → Prob process
  get :: process → Prob Bool
```

```haskell
newtype Polya = Polya (IORef (Int, Int))

instance BinProcess (Int, Int) Polya where
  new (i, j) = return
    $ Polya $ unsafePerformIO
    $ newIORef (i, j)
  get (Polya ref) = do
    let (i, j) = unsafePerformIO
          $ readIORef ref
    b ← bernoulli
      (fromIntegral i / fromIntegral (i + j))
    if b then return
      $ unsafePerformIO
      $ writeIORef ref (i + 1, j)
      >> return True
    else return
      $ unsafePerformIO
      $ writeIORef ref (i, j + 1)
      >> return False
```

=

```haskell
newtype BetaBern = BetaBern Double

instance BinProcess (Int, Int) BetaBern where
  new (i, j) = do
    θ ← beta (fromIntegral i)
             (fromIntegral j)
    return $ BetaBern θ
  get (BetaBern θ) = bernoulli θ
```

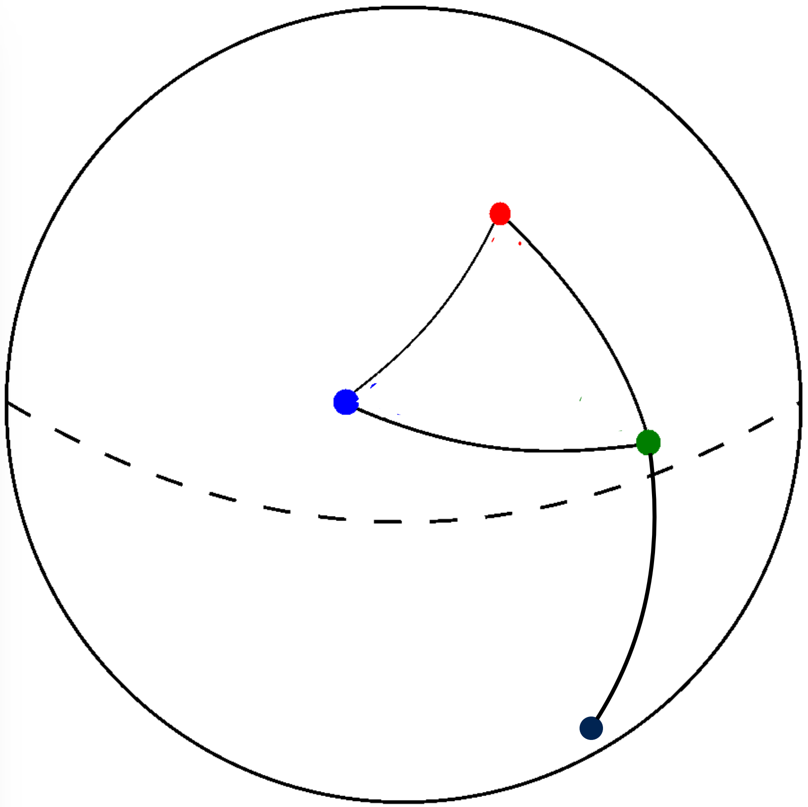
Random graphs: memoization for representation theorems

Aldous-Hoover (2-dimensional de Finetti)
for exchangeable simple random graphs

```haskell
class RandomGraph g where
  type Graph g
  data Vertex g
  newGraph :: g -> Prob (Graph g)
  newVertex :: Graph g -> Prob (Vertex g)
  isEdge :: Graph g -> Vertex g -> Vertex g -> Bool
```

Random graphs: geometric graphs

```haskell
data GeomGraph = GeomGraph Int Double

instance RandomGraph GeomGraph where
  type Graph GeomGraph = GeomGraph
  data Vertex GeomGraph = GGV [Double]
  newGraph g = return g
  newVertex (GeomGraph dim _) =
    GGV <$> replicateM
      dim uniform
  isEdge (GeomGraph _ neighRadius) x y =
    distance x y < neighRadius
```



building on
Bubeck, Ding, Eldan, Racz, 2015
Devroye, György, Lugosi, Udina, 2011

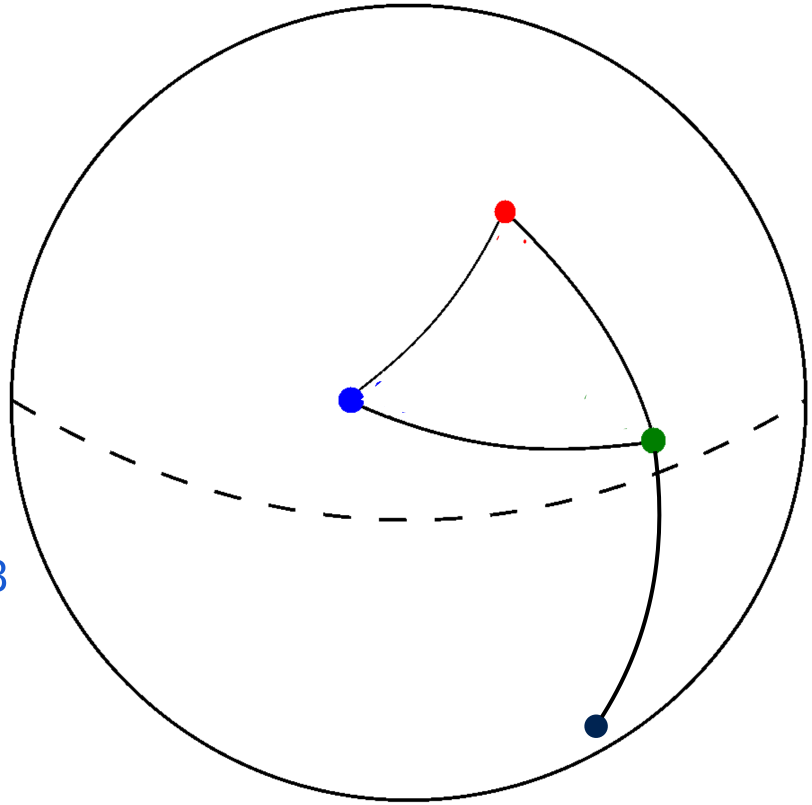
Random graphs: geometric graphs

```
newVertex = uniform Sₙ

isEdge(p,q) = if d(p,q) < π/2 then True else False
```

```
p <- newVertex
q <- newVertex        =  bernoulli 1/2
isEdge(p,q)
```

```
p <- newVertex
q <- newVertex
r <- newVertex
is-edge(p,q)
&& is-edge(q,r)
&& is-edge(p,r)
```

$\xrightarrow[n \to \infty]{}$ bernoulli 1/8



building on
Bubeck, Ding, Eldan, Racz, 2015
Devroye, György, Lugosi, Udina, 2011

Random graphs: Rado/Erdős-Rényi graph

Limiting case, when n → ∞

Can be implemented with `memoize`

```haskell
newtype Graphon = G ((Double, Double) -> Double)

instance RandomGraph Graphon where
  type Graph Graphon = (Double, Double) -> Bool
  data Vertex Graphon = V Double
  newGraph (G graphon) = memoize $ bernoulli . graphon
  newVertex _ = V <$> uniform
  isEdge g (V x) (V y) = g (x, y)
```

Staton: if exchangeable, up to contextual equivalence, it is the *only* implementation

Bipartite random graph topos

```
-- Atoms (randomly generated fresh names)
new_atom :: 𝔸

-- Function labels:
-- type to be thought of as 𝔸 → Bool
new_function :: 𝔽

-- Application operator making every function memoized:
-- type of a bipartite graph
(@) :: (𝔽, 𝔸) → Bool
```

Analogously to the Rado topos setting:
denotational semantics where we look for a topos where a *random countable bigraph* plays the role of the Rado graph in the Rado topos.

Bipartite random graph topos

```haskell
-- Atoms (randomly generated fresh names)
new_atom :: 𝔸

-- Function labels:
-- type to be thought of as 𝔸 → Bool
new_function :: 𝔽

-- Application operator making every function memoized:
-- type of a bipartite graph
(@) :: (𝔽, 𝔸) → Bool
```

Analogously to the Rado topos setting:
denotational semantics where we look for a topos where a *random countable bigraph* plays the role of the Rado graph in the Rado topos.

$\Longrightarrow$ We work in the category of covariant ***(pre)sheaves on*** the category of ***finite bigraphs and embeddings***

$$[\![\mathbb{F}]\!] = \mathbf{BiGrph}_{emb}(\circ, -) \qquad [\![\mathbb{A}]\!] = \mathbf{BiGrph}_{emb}(\bullet, -)$$

Memo-nominal sets

Similarly to *nominal sets (toposic Galois for empty theory)*:

Gabbay, Pitts. e.g. FACS 2002
Pitts' book, CUP 2013.
Stark 1994

```
mem-bernoulli p :: Prob (Atoms -> Bool)
fresh :: Prob Atoms
```

Two sorts of atoms: $\mathbb{F}$, $\mathbb{A}$.

Infinite memo table that embeds every finite memo table and allows extension (ultrahomogeneous)

Now reformulate nominal sets using automorphisms of this memo table.

e.g. Bojanczyk, Klin, Lasota, LMCS 2014

$\mathbb{F}$ labelling functions

$\mathbb{A}$ labelling arguments

x	f0 x	f1 x	f2 x	f3 x
0	0	1	0	0
1	1	1	0	1
2	0	0	0	1
3	0	1	1	1
4	0	1	1	1
5	0	1	0	1

$\mathbb{F} \times \mathbb{A} \to 2$ memo table

$\mathbb{F} \subseteq [\mathbb{A} \to 2]$, currying

Theorem. $\mathbf{MemoNom}$ is a sheaf subcategory of $[\mathbf{FinBiGrph}, \mathbf{Set}]$.

cf Caramello 2008,
Caramello & Lafforgue, JGL 2019

Rado topos

Staton: Erdős-Rényi graphons $[0,1]^2 \to [0,1]$ correspond to *internal* probability measures $2^V \to \mathbb{R}_{\geq 0}$ for which the Fubini theorem holds.

$$\mathrm{Sh}(\mathrm{FinGrph}^{\mathrm{op}}_{emb}, \mathrm{J}_{at}) \simeq \mathrm{Cont}(\mathrm{Aut}(R))$$

Toposic Galois approach

Rado graph = Fraïssé limit of the amalgamation class of finite graphs and embeddings $\longrightarrow$ *Ultrahomogeneous structure*

Fraïssé, 1950s.

For more general module interfaces:

1. Start with the signature of a countable first-order language L

2. Consider the category $\mathbb{C}$ of finitely generated L-structures and embeddings.

3. If $\mathcal{C} \overset{\text{def}}{=} \mathrm{ob}(\mathbb{C})$ is a suitable amalgamation class with Fraïssé limit M, when do we have

$$\mathrm{Sh}(\mathbb{C}^{\mathrm{op}}, \mathrm{J}_{at}) \simeq \mathrm{Cont}(\mathrm{Aut}(M)) \quad ?$$

cf Caramello 2008,
Caramello & Lafforgue, JGL 2019