A model of Stochastic Memoization & Name Generation

Younesse Kaddar, Sam Staton
University of Oxford

UNIVERSITY OF OXFORD

HOPE Workshop 2022

Summary

1. Deterministic vs stochastic memoization
2. Memoization equations
3. Dataflow property & Higher-order functions
4. Representation theorems, random graphs
5. Rado topos and toposic Galois theory
6. Our categorical model
    1. Bigraph topos
    2. Probabilistic local state monad

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

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	???

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**

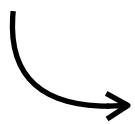
Deterministic Memoization

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!

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

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

x	f x
0.1	0.3364

https://lazyppl.bitbucket.io

Stochastic Memoization

With probability: **changes** the semantics!

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

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

x	f x
0.1	0.8364
0.3	0.5468

https://lazyppl.bitbucket.io

Stochastic Memoization

With probability: **changes** the semantics!

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

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

x	f x
0.1	0.8364
0.3	0.5468
0.1	0.3484

https://lazyppl.bitbucket.io

Stochastic Memoization

With probability: **changes** the semantics!

```haskell
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.

https://lazyppl.bitbucket.io

Stochastic Memoization

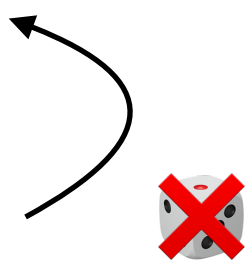
With probability: **changes** the semantics!

```haskell
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.

x	f x
0.1	0.8364
0.3	0.5468
0.1	0.8364



https://lazyppl.bitbucket.io

Stochastic Memoization

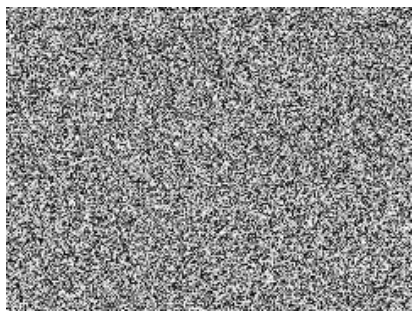
With probability: **changes** the semantics!

```haskell
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.

white noise



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, etc)

Stochastic Memoization

With probability: **changes** the semantics!

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

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)
```

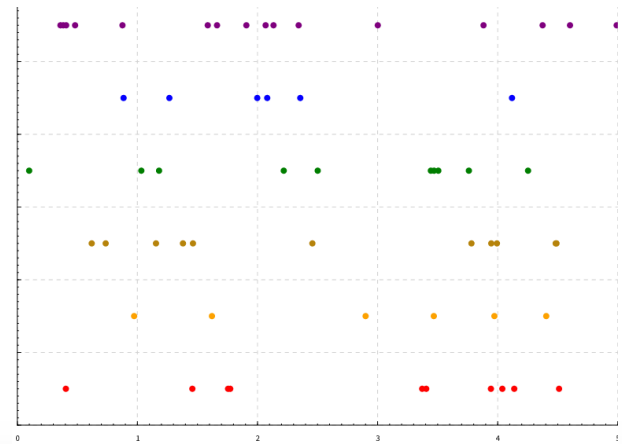
https://lazyppl.bitbucket.io

Stochastic Memoization

Countable Domain a

Laziness trick

Example: Poisson Point Process



Sample all the (exp distributed) interoccurrence times at once

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

https://lazyppl.bitbucket.io

Stochastic Memoization

Uncountable Domain?

Open problem!

Memoization equations

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

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

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

Memoization law

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

Memoization law

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

⛔ Problem:

The naive 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 naive implementation uses **state**!

And we want (non-negotiable):

1. The **dataflow property**
2. **Higher-order functions**

Memoization law

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

⛔ Problem:

The naive implementation uses **state**!

And we want (non-negotiable): ❌

1. The **dataflow property**
2. **Higher-order functions**

Dataflow property

```
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)

Dataflow property

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

BUT

Memoization **validates** dataflow

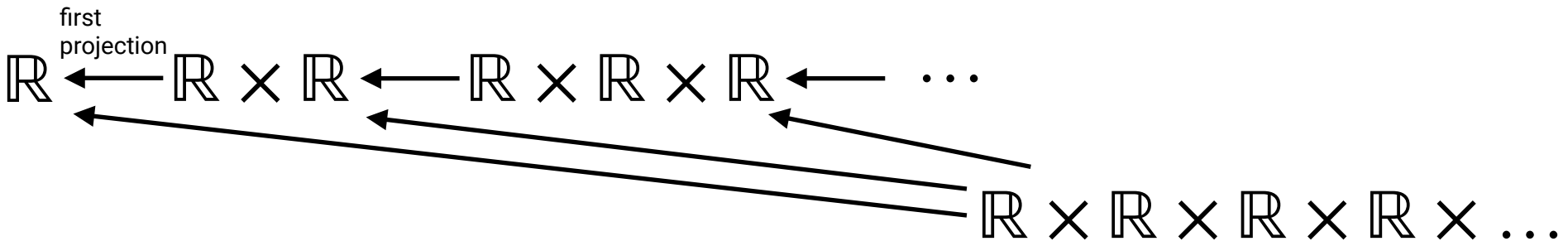
- Not *intrinsically* stateful
- *rather*: linked to pure probability theory

↳ ℹ️ *cf. Kolmogorov's extension theorem*
Pólya's Urn

Kolmogorov extension

In traditional probability theory:

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



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

⟶ Categorical probability: Quasi-Borel spaces (cartesian closed)
Unfortunately: QBSes do not support memoization

Heunen, Kammar, Staton, Yang. LICS 2017

Objective

⟶ Categorical probability: Quasi-Borel spaces (cartesian closed)

Unfortunately: QBSes do not support memoization

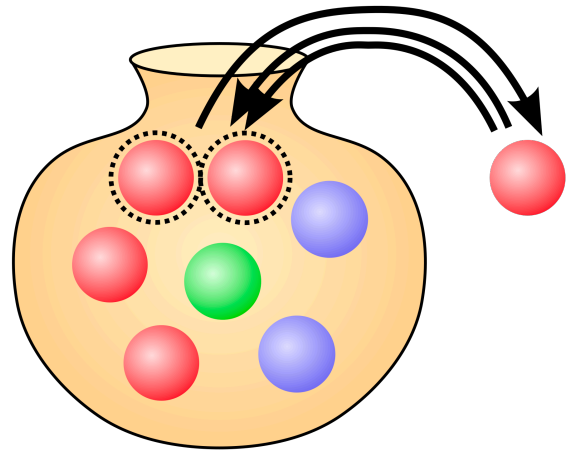
Heunen, Kammar, Staton, Yang. LICS 2017

Open problem

Find a model of probability with
- stochastic memoization
- dataflow property, and
- function spaces.

Pólya's Urn and de Finetti

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



```
1  class BinProcess hyperparam process where
2    new :: hyperparam → Prob process
3    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

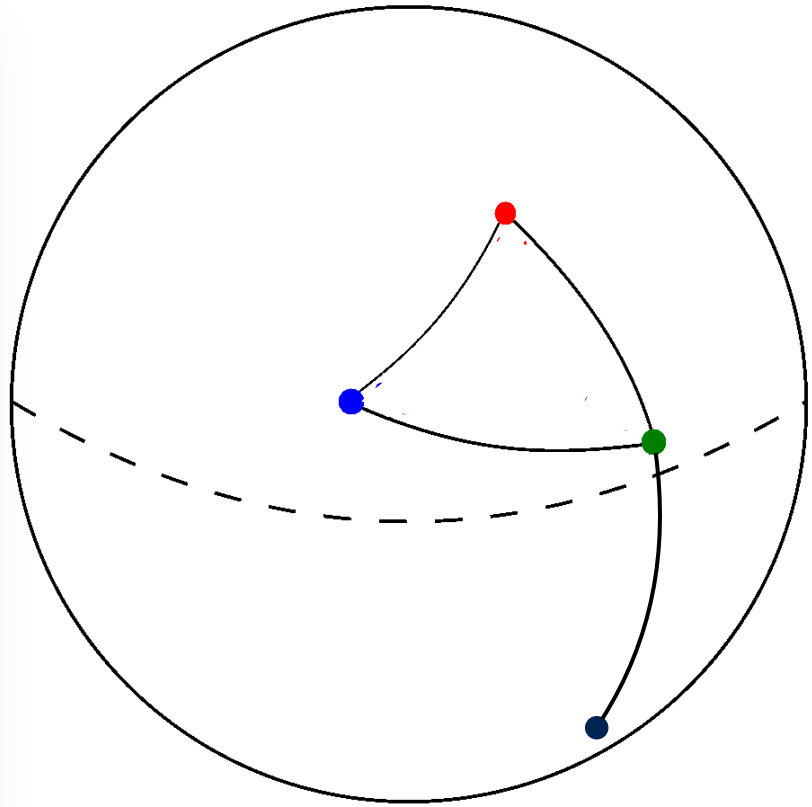
Aldous-Hoover (2-dimensional de Finetti)
for exchangeable simple random graphs

```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

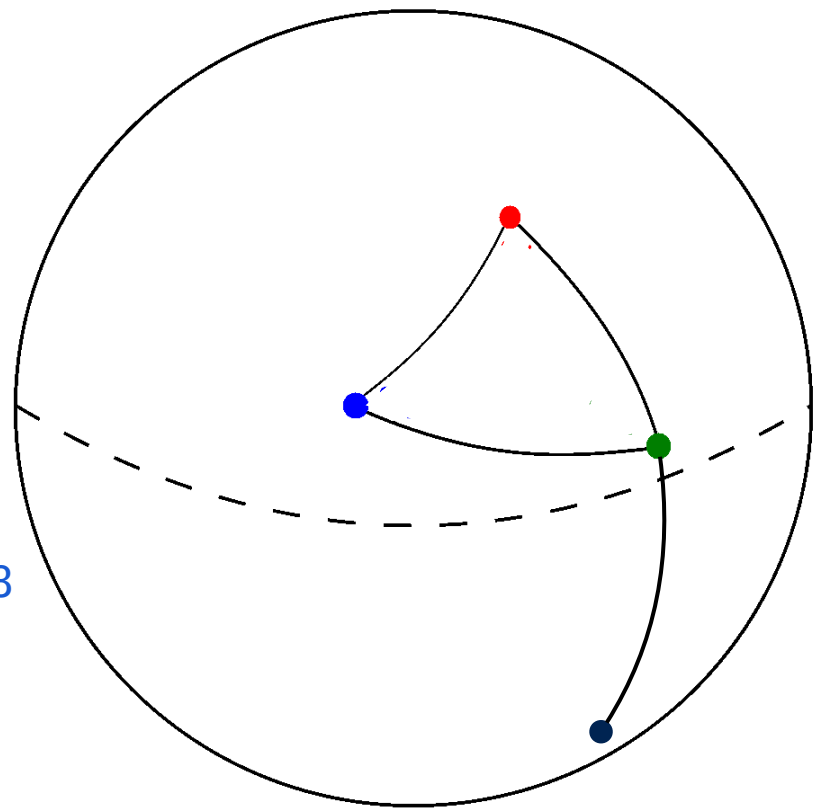
Random graphs: geometric graphs

```
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  ⟶ (n → ∞)  bernoulli 1/8
is-edge(p,q)
&& is-edge(q,r)
&& is-edge(p,r)
```



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

Rado topos

Staton: Graphons $[0,1]^2 \to [0,1]$ are in one-to-one correspondence with *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 $\mathscr{C} \stackrel{\text{def}}{=} \mathrm{ob}(\mathbb{C})$ 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

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
```

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

```
-- 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. **MemoNom** is a sheaf subcategory of **[FinBiGrph, Set]**.

cf Caramello 2008,
Caramello & Lafforgue, JGL 2019

Simplified language

Values:

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

Computations:

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

Matching:

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

Language-specific commands:

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

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

Name generation, probabilistic effects, memoization

Probabilistic local state monad

Theorem.

c.f. Plotkin Power FOSSACS 2002.
Kaddar, Staton, ongoing.

$$\mathrm{Prob}(X)(g) \;:=\; \left(\mathrm{FinProb} \int^{g \hookrightarrow h} \left(X(h) \times [0,1]^{(h-g)_{\mathbb{F}}}\right)\right)^{[0,1]^{g_{\mathbb{F}}}}$$

defines a commutative affine monad.

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

Gives a denotational semantics to our language

Denotational semantics

Denotation of $\Gamma \vdash \mathsf{fresh}() : \mathbb{A}$

The map $[\![\mathsf{fresh}]\!]_g : [\![\Gamma]\!](g) \to T([\![\mathbb{A}]\!])(g)$ randomly chooses connections to each left node according to the state of biases, and makes a fresh right node with those connections.

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

where Z is a normalization constant.

Denotational semantics

Denotation of $\Gamma \vdash \lambda_{\text{ᴍ}} x.\, u : \mathbb{F}$

As $\lambda_{\text{ᴍ}}$-abstractions are formed based on computation judgements of the form $\Gamma, x : \mathbb{A} \vdash u : \mathsf{bool}$, we first note that

$$T(\llbracket \mathsf{bool} \rrbracket) g \;\cong\; P_{\mathrm{f}}(2)^{[0,1]^{g_L}} \;\cong\; [0,1]^{[0,1]^{g_L}}$$

$$\llbracket \lambda_{\text{ᴍ}} x.\, u \rrbracket_g : \begin{cases} 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \longrightarrow P_{\mathrm{f}}(g_L + 2^{g_R} \times [0,1])^{[0,1]^{g_L}} \\ b^k,\ (\circ \overset{\kappa_i}{\hookrightarrow} g)_i,\ (\bullet \overset{\tau_j}{\hookrightarrow} g)_j \mapsto \lambda \mapsto \left\langle \dfrac{1}{Z} \displaystyle\prod_{a \in g_R} p_a^{E^h(\mathfrak{f}_h(\circ), a)} (1 - p_a)^{1 - E^h(\mathfrak{f}_h(\circ), a)} \;\middle|\; \underbrace{[\circ}_{\cong (h-g)_L} \overset{\mathfrak{f}_h}{\hookrightarrow} h,\ \_ \mapsto \tilde{p}]_g \right\rangle_{h \in L_g} \end{cases}$$

where Z is a normalization constant, and

- for every $a \in g_R$, $\quad p_a \;:=\; \llbracket u \rrbracket_g \big(b^k,\ (\circ \overset{\kappa_i}{\hookrightarrow} g)_i,\ (\bullet \overset{\tau_j}{\hookrightarrow} g)_j,\ \bullet \overset{a}{\hookrightarrow} g,\ \lambda\big)$
- $\tilde{p} \;:=\; \llbracket u \rrbracket_g \big(b^k,\ (\circ \overset{\kappa_i}{\hookrightarrow} g \overset{\iota_1}{\hookrightarrow} g + \bullet)_i,\ (\bullet \overset{\tau_j}{\hookrightarrow} g \overset{\iota_1}{\hookrightarrow} g + \bullet)_j,\ \bullet \overset{\iota_2}{\hookrightarrow} g + \bullet,\ \lambda\big)$ where $\iota_1,\ \iota_2$ are the coprojections.

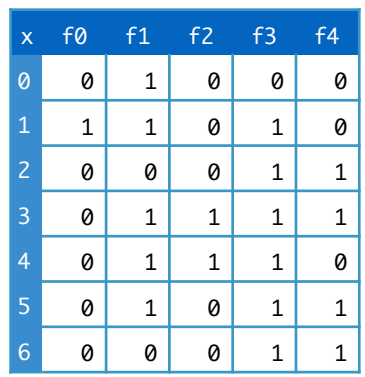
Summary

1. Deterministic vs stochastic memoization
2. Memoization equations
3. Dataflow property & Higher-order functions
4. Representation theorems, random graphs
5. Rado topos and toposic Galois theory
6. Our categorical model
    1. Bigraph topos
    2. Probabilistic local state monad

Other approaches

Operational semantics

Configurations:

$\langle e,$  $\rangle$

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

expression

memo table +
closure for each column

Sound for our denotational semantics, but works more generally.

Operational techniques to show dataflow property?