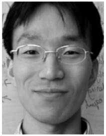
Semantics for Probabilistic Programming

Chris Heunen

THE UNIVERSITY of EDINBURGH informatics

Bayes' law

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

Bayes' law

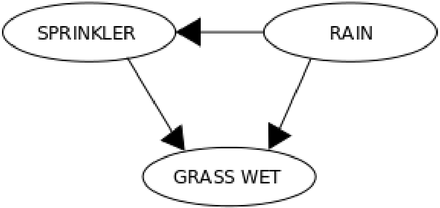
$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

Bayesian reasoning:

- predict future, based on model and prior evidence
- infer causes, based on model and posterior evidence
- learn better model, based on prior model and evidence

Bayesian networks

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Bayesian inference

Stan implements gradient-based Markov chain Monte Carlo (MCMC) algorithms for Bayesian inference, stochastic, gradient-based variational Bayesian methods for approximate Bayesian inference, and gradient-based optimization for penalized maximum likelihood estimation.

About TensorFlow

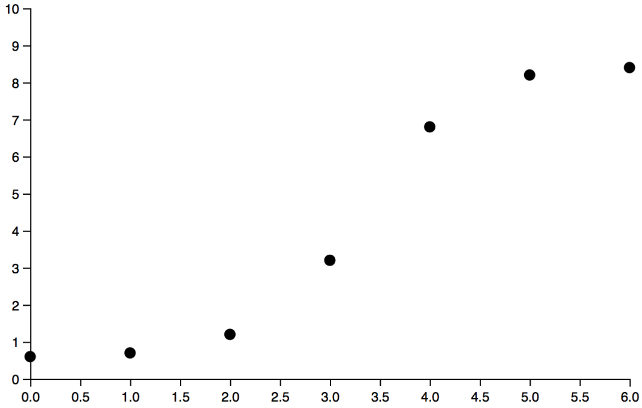
TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the

Infer.NET

Infer.NET is a framework for running Bayesian inference in graphical models.

Bayesian data modelling

1. Develop probabilistic (generative) model
2. Design inference algorithm for model
3. Use algorithm to fit model to data



Example: find effect of drug on patient, given data

Linear regression

Generative model

$$
\begin{aligned}
s &\sim \text{normal}(0, 2) \\
b &\sim \text{normal}(0, 6) \\
f(x) &= s \cdot x + b \\
y_i &= \text{normal}(f(i), 0.5) \\
&\qquad \text{for } i = 0 \ldots 6
\end{aligned}
$$

Conditioning

$$y_0 = 0.6, y_1 = 0.7, y_2 = 1.2, y_3 = 3.2, y_4 = 6.8, y_5 = 8.2, y_6 = 8.4$$

Predict f

Linear regression

```
# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but TensorFlow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.global_variables_initializer()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))
```

Probabilistic programming

1. ~~Develop probabilistic (generative) model~~ Write a program
~~2. Design inference algorithm for model~~
2. Use built-in algorithm to fit model to data

Probabilistic programming

1. ~~Develop probabilistic (generative) model~~ Write a program
~~2. Design inference algorithm for model~~
2. Use built-in algorithm to fit model to data

$$P(A \mid B) \propto P(B \mid A) \times P(A)$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

functional programming + **observe** + **sample**

Probabilistic programming

1. ~~Develop probabilistic (generative) model~~ Write a program
~~2. Design inference algorithm for model~~
2. Use built-in algorithm to fit model to data

$$\mathrm{P}(A \mid B) \propto \mathrm{P}(B \mid A) \times \mathrm{P}(A)$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

$$\text{functional programming} + \textbf{observe} + \textbf{sample}$$

Church is a universal probabilistic programming language, extending Scheme with probabilistic semantics, and is well suited for describing infinite-dimensional stochastic processes and other recursively-defined generative processes

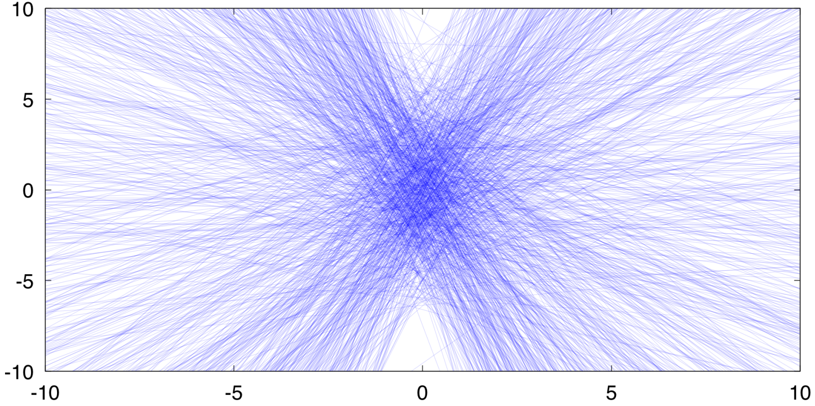
Venture is an interactive, Turing-complete, higher-order probabilistic programming platform that aims to be sufficiently expressive, extensible and efficient for general-purpose use. Its virtual machine supports multiple scalable, reprogrammable inference strategies, plus two front-end languages: VenChurch and VentureScript.

Anglican is a portable Turing-complete research probabilistic programming language that includes particle MCMC inference.

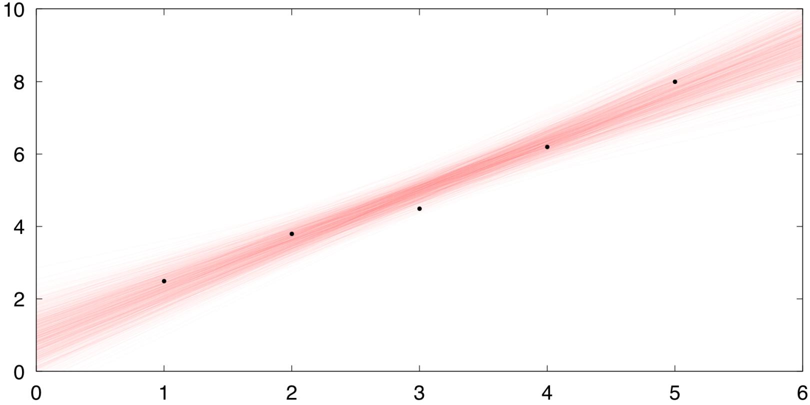
Linear regression

```
(defquery Bayesian-linear-regression

 (let [f (let [s (sample (normal 0.0 3.0))
               b (sample (normal 0.0 3.0))]
           (fn [x] (+ (* s x) b)))]

   (observe (normal (f 1.0) 0.5) 2.5)
   (observe (normal (f 2.0) 0.5) 3.8)
   (observe (normal (f 3.0) 0.5) 4.5)
   (observe (normal (f 4.0) 0.5) 6.2)
   (observe (normal (f 5.0) 0.5) 8.0)

   (predict :f f)))
```

Linear regression

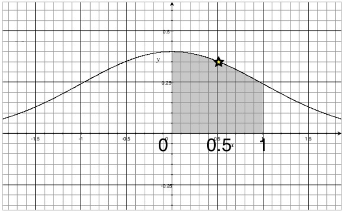


Linear regression



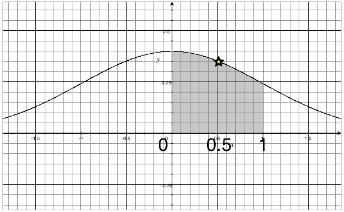
Measure theory

Impossible to sample 0.5 from standard normal distribution
But sample in interval $(0, 1)$ with probability around 0.34



Measure theory

Impossible to sample 0.5 from standard normal distribution
But sample in interval $(0, 1)$ with probability around 0.34

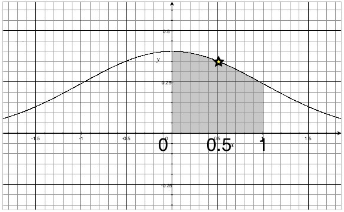


A measurable space is a set X with a family Σ_X of subsets
that is closed under countable unions and complements

A (probability) measure on X is a function $p \colon \Sigma_X \to [0, \infty]$
that satisfies $p(\sum U_n) = \sum p(U_n)$ (and has $p(X) = 1$)

Measure theory

Impossible to sample 0.5 from standard normal distribution
But sample in interval $(0, 1)$ with probability around 0.34

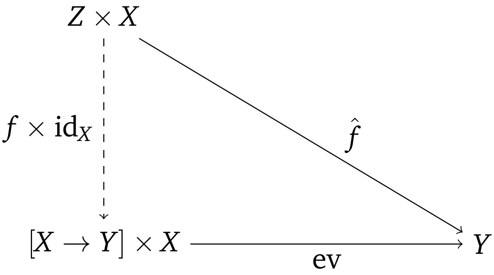


A measurable space is a set X with a family Σ_X of subsets
that is closed under countable unions and complements

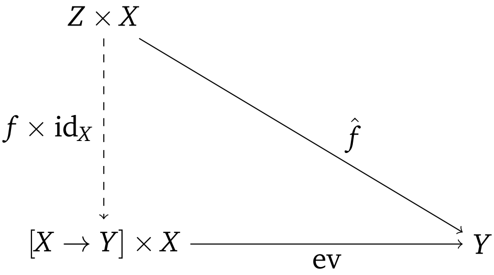
A (probability) measure on X is a function $p \colon \Sigma_X \to [0, \infty]$
that satisfies $p(\sum U_n) = \sum p(U_n)$ (and has $p(X) = 1$)

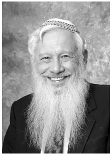
A function $f \colon X \to Y$ is measurable if $f^{-1}(U) \in \Sigma_X$ for $U \in \Sigma_Y$
A random variable is a measurable function $\mathbb{R} \to X$

Function types



Function types





$[\mathbb{R} \to \mathbb{R}]$ cannot be a measurable space!

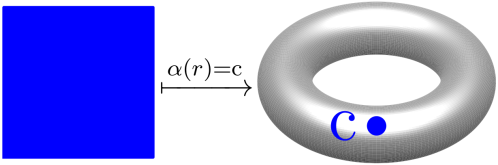
Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

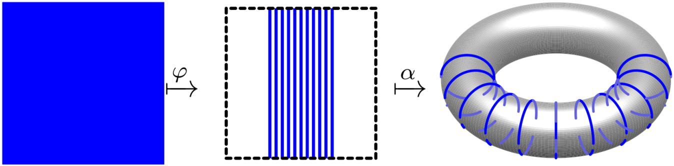
- $\alpha \in M_X$ if $\alpha \colon \mathbb{R} \to X$ is constant



Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

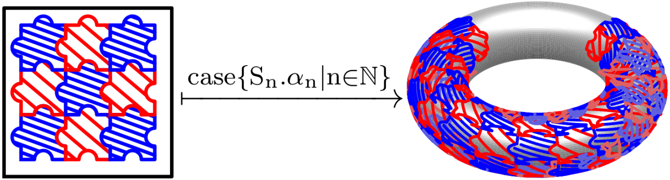
- $\alpha \in M_X$ if $\alpha\colon \mathbb{R} \to X$ is constant
- $\alpha \circ \varphi \in M_X$ if $\alpha \in M_X$ and $\varphi\colon \mathbb{R} \to \mathbb{R}$ is measurable



Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

- $\alpha \in M_X$ if $\alpha\colon \mathbb{R} \to X$ is constant
- $\alpha \circ \varphi \in M_X$ if $\alpha \in M_X$ and $\varphi\colon \mathbb{R} \to \mathbb{R}$ is measurable
- if $\mathbb{R} = \biguplus_{n \in \mathbb{N}} S_n$, with each set S_n Borel, and $\alpha_1, \alpha_2, \ldots \in M_X$, then β is in M_X, where $\beta(r) = \alpha_n(r)$ for $r \in S_n$



Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

- $\alpha \in M_X$ if $\alpha \colon \mathbb{R} \to X$ is constant
- $\alpha \circ \varphi \in M_X$ if $\alpha \in M_X$ and $\varphi \colon \mathbb{R} \to \mathbb{R}$ is measurable
- if $\mathbb{R} = \biguplus_{n \in \mathbb{N}} S_n$, with each set S_n Borel, and $\alpha_1, \alpha_2, \ldots \in M_X$, then β is in M_X, where $\beta(r) = \alpha_n(r)$ for $r \in S_n$

A morphism is a function $f \colon X \to Y$ with $f \circ \alpha \in M_Y$ if $\alpha \in M_X$

Quasi-Borel spaces

A quasi-Borel space is a set X together with $M_X \subseteq [\mathbb{R} \to X]$ satisfying:

- $\alpha \in M_X$ if $\alpha \colon \mathbb{R} \to X$ is constant
- $\alpha \circ \varphi \in M_X$ if $\alpha \in M_X$ and $\varphi \colon \mathbb{R} \to \mathbb{R}$ is measurable
- if $\mathbb{R} = \biguplus_{n \in \mathbb{N}} S_n$, with each set S_n Borel, and $\alpha_1, \alpha_2, \ldots \in M_X$, then β is in M_X, where $\beta(r) = \alpha_n(r)$ for $r \in S_n$

A morphism is a function $f \colon X \to Y$ with $f \circ \alpha \in M_Y$ if $\alpha \in M_X$

- has product types
- has sum types
- has function types!

$$M_{[X \to Y]} = \{\alpha \colon \mathbb{R} \to [X \to Y] \mid \hat{\alpha} \colon \mathbb{R} \times X \to Y \text{ morphism}\}$$

Example quasi-Borel spaces

$$\mathbf{Set} \underset{\longleftarrow}{\overset{\longrightarrow}{\perp}} \mathbf{Qbs}$$

$$X \longmapsto (X, \{\texttt{case}\, S_n.x_n \mid S_n \subseteq X \text{ partition}, x_n \in \mathbb{R}\})$$

$$X \longleftarrow\!\!\dashv (X, M_X)$$

Example quasi-Borel spaces

$$\mathbf{Set} \underset{}{\overset{}{\rightleftarrows}} \mathbf{Qbs} \quad (\bot)$$

$$X \longmapsto (X, \{\texttt{case}\, S_n.x_n \mid S_n \subseteq X \text{ partition}, x_n \in \mathbb{R}\})$$

$$X \longleftarrow\!\shortmid (X, M_X)$$

$$\mathbf{Set} \underset{}{\overset{}{\rightleftarrows}} \mathbf{Qbs} \quad (\top)$$

$$X \longmapsto (X, \{\alpha\colon \mathbb{R} \to X\})$$

$$X \longleftarrow\!\shortmid (X, M_X)$$

Example quasi-Borel spaces

$$\mathbf{Set} \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathbf{Qbs}$$

$$X \longmapsto (X, \{\texttt{case}\, S_n.x_n \mid S_n \subseteq X \text{ partition}, x_n \in \mathbb{R}\})$$

$$X \longleftarrow (X, M_X)$$

$$\mathbf{Set} \underset{\longleftarrow}{\overset{\longrightarrow}{\top}} \mathbf{Qbs}$$

$$X \longmapsto (X, \{\alpha \colon \mathbb{R} \to X\})$$

$$X \longleftarrow (X, M_X)$$

$$\mathbf{Meas} \underset{\longleftarrow}{\overset{\longrightarrow}{\top}} \mathbf{Qbs}$$

$$(X, \Sigma_X) \longmapsto (X, \{\alpha \colon \mathbb{R} \to X \text{ measurable}\})$$

$$(X, \{U \mid \forall \alpha \in M_X \colon \alpha^{-1}(U) \text{ measurable}\}) \longleftarrow (X, M_X)$$

Distribution types

A measure on a quasi-Borel space (X, M_X) consists of

- $\alpha \in M_X$ and
- a probability measure μ on $\mathbb{R}$

Two measures are identified when they induce the same $\mu(\alpha^{-1}(-))$

Distribution types

A measure on a quasi-Borel space (X, M_X) consists of

- $\alpha \in M_X$ and
- a probability measure μ on $\mathbb{R}$

Two measures are identified when they induce the same $\mu(\alpha^{-1}(-))$

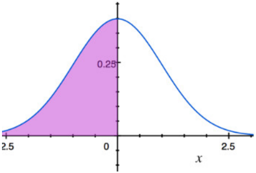
Gives monad

- $P(X, M_X) = \{(\alpha, \mu) \text{ measure on } (X, M_X\}/\sim$
- return $x = [\lambda r.x, \mu]_\sim$ for arbitrary μ
- bind uses integral $\int f \mathrm{d}(\alpha, \mu) := \int (f \circ \alpha) \mathrm{d}\mu$ if $f \colon (X, M_X) \to \mathbb{R}$

for distribution types

Example: facts about distributions

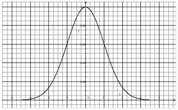
$$\left[\!\left[\begin{array}{l}\texttt{let x = sample(gauss(0.0,1.0))}\\ \texttt{in return (x<0)}\end{array}\right]\!\right] = [\![\texttt{sample(bern(0.5))}]\!]$$



Example: importance sampling

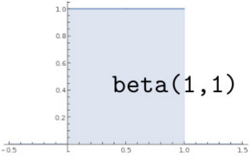
```
[[ sample(exp(2)) ]]
```

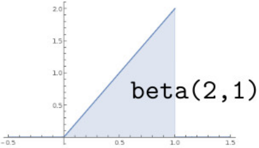
```
= [[ let x = sample(gauss(0,1)))
     observe(exp-pdf(2,x)/gauss-pdf(0,1,x));
     return x ]]
```



Example: conjugate priors

$$\left[\!\left[\begin{array}{l}\texttt{let x = sample(beta(1,1))}\\ \texttt{in observe(bern(x), true);}\\ \texttt{return x}\end{array}\right]\!\right] = \left[\!\left[\begin{array}{l}\texttt{observe(bern(0.5), true);}\\ \texttt{let x = sample(beta(2,1))}\\ \texttt{in return x}\end{array}\right]\!\right]$$





Linear regression

```
(defquery Bayesian-linear-regression
```

Prior:

```
(let [f (let [s (sample (normal 0.0 3.0))
              b (sample (normal 0.0 3.0))]
          (fn [x] (+ (* s x) b)))]
```

Likelihood:

```
  (observe (normal (f 1.0) 0.5) 2.5)
  (observe (normal (f 2.0) 0.5) 3.8)
  (observe (normal (f 3.0) 0.5) 4.5)
  (observe (normal (f 4.0) 0.5) 6.2)
  (observe (normal (f 5.0) 0.5) 8.0)
```

Posterior:

```
  (predict :f f)))
```

Linear regression: prior

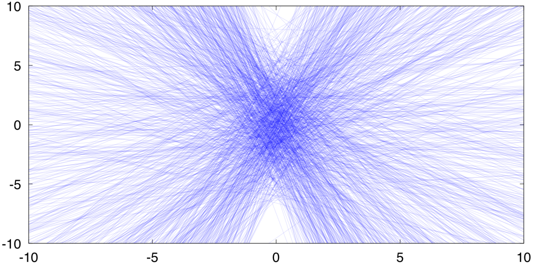
Define a prior measure on $[\mathbb{R} \to \mathbb{R}]$

$$\left[\!\left[\texttt{(let [f (let [s (sample (normal 0.0 3.0))} \quad \texttt{b (sample (normal 0.0 3.0))]} \quad \texttt{(fn [x] (+ (* s x) b)))]} \right]\!\right]$$

```
(let [f (let [s (sample (normal 0.0 3.0))
              b (sample (normal 0.0 3.0))]
          (fn [x] (+ (* s x) b)))]
```

$$= \quad [\alpha, \nu \otimes \nu]_{\sim} \in P([\mathbb{R} \to \mathbb{R}])$$

where ν is normal distribution, mean 0 and standard deviation 3, and $\alpha : \mathbb{R} \times \mathbb{R} \to [\mathbb{R} \to \mathbb{R}]$ is $(s, b) \mapsto \lambda r.sr + b$



Linear regression: likelihood

Define likelihood of observations (with some noise)

$$\left[\!\left[\begin{array}{l}\texttt{(observe (normal (f 1.0) 0.5) 2.5)}\\ \texttt{(observe (normal (f 2.0) 0.5) 3.8)}\\ \texttt{(observe (normal (f 3.0) 0.5) 4.5)}\\ \texttt{(observe (normal (f 4.0) 0.5) 6.2)}\\ \texttt{(observe (normal (f 5.0) 0.5) 8.0)}\end{array}\right]\!\right]$$

$$= \quad d(f(1), 2.5) \cdot d(f(2), 3.8) \cdot d(f(3), 4.5) \cdot d(f(4), 6.2) \cdot d(f(5), 8.0)$$

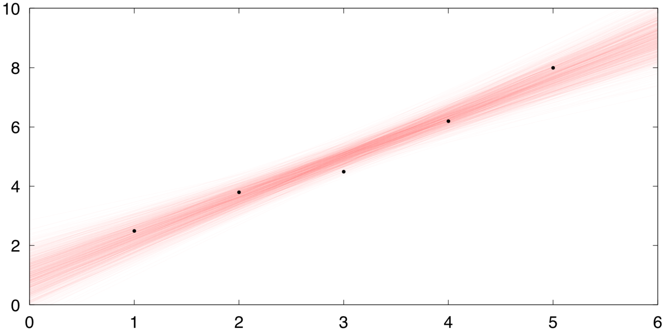
where f free variable of type $[\mathbb{R} \to \mathbb{R}]$, and $d \colon \mathbb{R}^2 \to [0, \infty)$ is density of normal distribution with standard deviation 0.5

$$d(\mu, x) = \sqrt{2/\pi} \exp(-2(x-\mu)^2)$$

Linear regression: Posterior

Normalise combined prior and likelihood

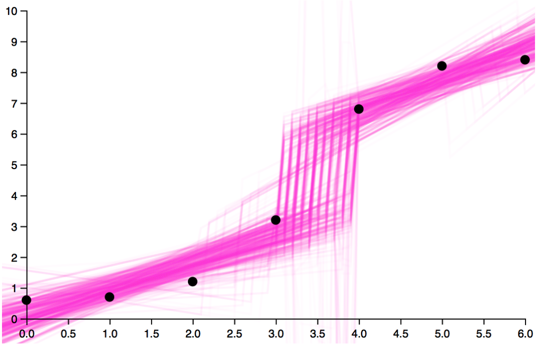
$$[\![(\texttt{predict :f f})))]\!] \in P([\mathbb{R} \to \mathbb{R}])$$



Piecewise linear regression: Posterior

Normalise combined prior and likelihood

$$[\![(\texttt{predict :f f})))]\!] \in P([\mathbb{R} \to \mathbb{R}])$$



Modular inference algorithms

An inference representation is monad $(T, \texttt{return}, \gg\!=)$ with $T\,X \to P\,X$, sample: $1 \to T\,[0,1]$, score: $[0,\infty) \to T\,1$.

- Discrete weighted sampler (e.g. coin flip)
- Continuous sampler

Modular inference algorithms

An inference representation is monad $(T, \texttt{return}, \gg=)$ with $T\,X \to P\,X$, sample: $1 \to T\,[0,1]$, score: $[0,\infty) \to T\,1$.

- Discrete weighted sampler (e.g. coin flip)
- Continuous sampler

An inference transformer respects meaning, sample, and score.

- List: $T(-) \mapsto T\big(\texttt{List}(-)\big)$
- Continuous weighting: $T(-) \mapsto T\big([0,\infty) * (-)\big)$
- Population: $T(-) \mapsto T\Big(\texttt{List}\big([0,\infty) * (-)\big)\Big)$

Modular inference algorithms library

Sequential Monte Carlo: approximate distribution by population of weighted samples (particles/suspended computations), repeatedly applying fixed random process (particle filter)

$$
\begin{aligned}
&\textbf{instance } \textit{Sampling Monad}\,(\mathrm{Sam}) \textbf{ where}\\
&\quad \textbf{return}\, x = \mathrm{Return}\, x\\
&\quad a \gg\!= f \quad = \textbf{match } a \textbf{ with } \{\\
&\qquad\qquad \mathrm{Return}\, x \to f(x)\\
&\qquad\qquad \mathrm{Sample}\, k \to\\
&\qquad\qquad\quad \mathrm{Sample}\,(\lambda r.\, k(r) \gg\!= f)\}\\
&\quad \mathrm{sample} \quad = \mathrm{Sample}\, \lambda r.\,(\mathrm{Return}\, r)\\
&\quad m\, a \quad = \textbf{match } a \textbf{ with } \{\\
&\qquad\qquad \mathrm{Return}\, x \to \underline{\delta}_x\\
&\qquad\qquad \mathrm{Sample}\, k \to \oint_{\mathbb{I}} k(x)\mathbf{U}(\mathrm{d}x)\}
\end{aligned}
$$

(a) Continuous sampler representation

$$
\begin{aligned}
&\textbf{instance } \textit{Cond Trans}\,(\mathrm{Pop}) \textbf{ where}\\
&\quad \textbf{return}_{\mathrm{Pop}\,\underline{T}} = \textbf{return}_{(\mathrm{W}\circ\mathrm{ListT})\underline{T}}\\
&\quad \gg\!=_{\mathrm{Pop}\,\underline{T}} \quad = \gg\!=_{(\mathrm{W}\circ\mathrm{ListT})\underline{T}}\\
&\quad \textbf{lift}_{\mathrm{Pop}\,\underline{T}} \quad = \textbf{lift}_{\mathrm{W}(\mathrm{ListT}\,\underline{T})} \circ \textbf{lift}_{\mathrm{ListT}\,\underline{T}}\\
&\quad \textbf{tmap}_{\mathrm{Pop}\,\underline{T}} \quad = \textbf{tmap}_{\mathrm{W}(\mathrm{ListT}\,\underline{T})} \circ \textbf{tmap}_{\mathrm{ListT}\,\underline{T}}\\
&\quad m_{\mathrm{Pop}\,\underline{T}} \quad = m_{(\mathrm{W}\circ\mathrm{ListT})\underline{T}}\\
&\qquad\qquad = \lambda a.\, \oint_{\mathrm{List}(\mathbb{R}_+\times X)} m^T(a)(\mathrm{d}x_s) \textstyle\sum_{(r,x)\in x_s} r \odot \underline{\delta}_x\\
&\quad \textbf{score}_{\mathrm{Pop}\,\underline{T}} \quad = \textbf{score}_{(\mathrm{W}\circ\mathrm{ListT})\underline{T}}
\end{aligned}
$$

(b) The population transformer

$$
\begin{aligned}
&\textbf{instance } \textit{Cond} \implies \textit{Cond Trans}\,(\mathrm{Sus}) \textbf{ where}\\
&\quad \textbf{return}_{\mathrm{Sus}\,\underline{T}}\, x \quad = \mathrm{return}_{\underline{T}}(\mathrm{Return}\, x)\\
&\quad a \gg\!=_{\mathrm{Sus}\,\underline{T}} f \quad = \textbf{fold}\,(\lambda b.\, \underline{T}.\textbf{do}\, \{\\
&\qquad\qquad t \leftarrow b;\\
&\qquad\qquad \textbf{match } t \textbf{ with } \{\\
&\qquad\qquad\quad \mathrm{Return}\, x \to f(x)\\
&\qquad\qquad\quad \mid \mathrm{Yield}\, t \quad \to \mathrm{Yield}\, t\}\})\\
&\qquad\qquad a\\
&\quad \textbf{lift}_{\mathrm{Sus}\,\underline{T}}\, a \quad = \underline{T}.\textbf{do}\, \{x \leftarrow a; \mathrm{return}_{\mathrm{Sus}\,\underline{T}}\, x\}\\
&\quad (\textbf{tmap}_{\mathrm{Sus}\,\underline{T}}\, \underline{t})_X = \mathrm{Sus}\,\underline{T}X.\textbf{fold}\,(\lambda b.\ m_{\underline{S}}(b))\\
&\quad m_{\mathrm{Sus}\,\underline{T}}\, a \quad = m_{\underline{T}}(\mathrm{finish}_{\mathrm{Sus}\,\underline{T}}(a))\\
&\quad \textbf{score}\, r \quad = \mathrm{return}_{\underline{T}}(\mathrm{Yield}\ \mathrm{lift}_{\mathrm{Sus}\,\underline{T}}(\textbf{score}\, r))
\end{aligned}
$$

(a) The suspension transformer

Want more?

- “Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints”
  LiCS 2016
- “A convenient category for higher-order probability theory”
  LiCS 2017
- “Denotational validation of higher-order Bayesian inference”
  POPL 2018

De Finetti's theorem

Every exchangeable sequence of random observations on $\mathbb{R}$ can be generated by:

- choose a single probability distribution on $\mathbb{R}$
- sample that one independently repeatedly

De Finetti's theorem

Every exchangeable sequence of random observations on a quasi-Borel space X can be generated by:

- choose a single probability distribution on X
- sample that one independently repeatedly

Trace Markov Chain Monte Carlo

Repeatedly use kernel to propose new value,
decide whether to accept (Metropolis-Hastings update).
Random walk in target space: program traces.

- Metropolis-Hastings-Green: update preserves distribution.
- Program traces form inference representation.
- Trace MCMC is inference transformation
  (parametrised by proposal kernel)