A convenient category for higher-order probability theory

Chris Heunen, Ohad Kammar, Sam Staton,
and Hongseok Yang

https://arxiv.org/abs/1701.02547

Thirty-Second Annual ACM/IEEE Symposium on
Logic in Computer Science (LICS)
20–23 June 2017, Reykjavik

What is higher-order probabilistic programming?

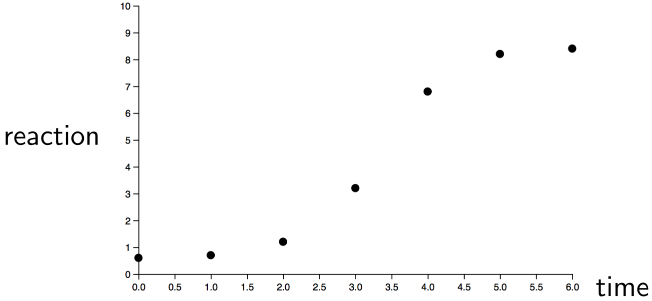
Bayesian data modelling

1. Develop a probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algorithm, fit the model to the data.

What is higher-order probabilistic programming?

Example

Effect of a drug on a patient, given data:



What is higher-order probabilistic programming?

Generative model

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

What is higher-order probabilistic programming?

Generative model

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

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?

What is higher-order probabilistic programming?

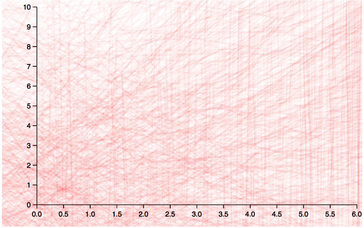
Bayesian inference

$$P(s, b|y_0, \ldots, y_6) = \frac{P(y_0, \ldots, y_6|s, b) \cdot P(s, b)}{P(y_0, \ldots, y_6)}$$

What is higher-order probabilistic programming?

Bayesian inference

$$P(s, b | y_0, \ldots, y_6) = \frac{P(y_0, \ldots, y_6 | s, b) \cdot P(s, b)}{P(y_0, \ldots, y_6)}$$

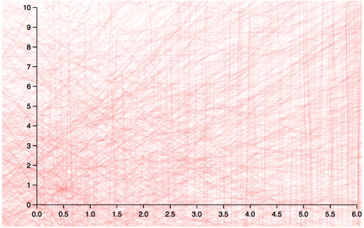


Prior

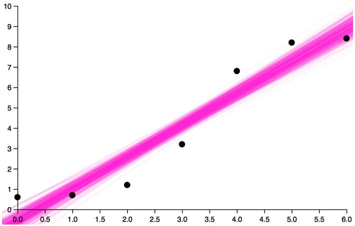
What is higher-order probabilistic programming?

Bayesian inference

$$P(s, b | y_0, \ldots, y_6) = \frac{P(y_0, \ldots, y_6 | s, b) \cdot P(s, b)}{P(y_0, \ldots, y_6)}$$



Prior



Posterior

What is higher-order probabilistic programming?

Probabilistic programming models

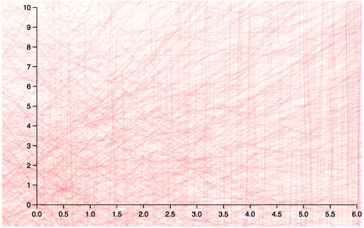
1. Develop a probabilistic (generative) model.
   Write a program.
2. ~~Design an inference algorithm for the model.~~
3. Using the built-in algorithm, fit the model to the data.

What is higher-order probabilistic programming?

In Anglican [Wood et al.'14]

```
(let [s (sample (normal 0.0 2.0))
      b (sample (normal 0.0 6.0))
      f (fn [x] (+ (* s x) b))]
```

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



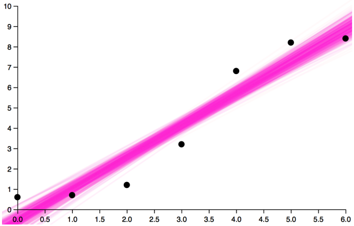
What is higher-order probabilistic programming?

In Anglican [Wood et al.'14]

```
(let [s (sample (normal 0.0 2.0))
      b (sample (normal 0.0 6.0))
      f (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))
```

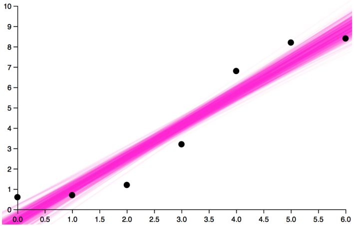


What is higher-order probabilistic programming?

In Anglican [Wood et al.'14]

```
(let [F (fn [] (let [s (sample (normal 0.0 2.0))
                     b (sample (normal 0.0 6.0))]
            (fn [x] (+ (* s x) b))))
      f (F)]
  (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))
```

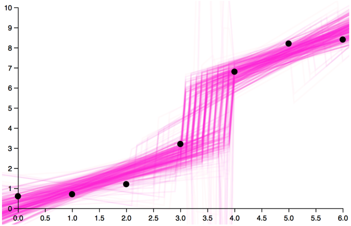


What is higher-order probabilistic programming?

In Anglican [Wood et al.'14]

```
(let [F (fn [] (let [s (sample (normal 0.0 2.0))
                      b (sample (normal 0.0 6.0))]
          (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6) ]
  (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))
```

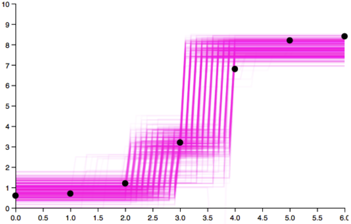


What is higher-order probabilistic programming?

In Anglican [Wood et al.'14]

```
(let [F (fn [] (let [
                     b (sample (normal 0.0 6.0))]
          (fn [x]    b )))
      f (add-change-points F 0 6) ]
  (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))
```



Semantics for probabilistic languages

Specification

- Continuous distributions.

Semantics

Category of measurable spaces

Semantics for probabilistic languages

Specification

- Continuous distributions.
- Distributions over higher-order data.
- Higher-order "glue".

Semantic challenge

Category of measurable spaces is **not** Cartesian closed!

Semantics for probabilistic languages

Specification

- Continuous distributions.
- Distributions over higher-order data.
- Higher-order "glue".

Semantic challenge

Category of measurable spaces is **not** Cartesian closed!

Theorem (Aumann'61)

The set $\mathbf{Meas}(\mathbb{R}, \mathbb{R})$ *cannot be made into a measurable space with*

$$eval : \mathbf{Meas}(\mathbb{R}, \mathbb{R}) \times \mathbb{R} \to \mathbb{R}$$

measurable.

Contribution

Quasi-Borel spaces

A Cartesian closed category for higher-order probability theory:

- Sets-with-structure ('random elements') and structure preserving functions.
- Well-behaved and concrete categorical structure.
- Well-behaved probabilistic structure: randomization and de Finetti's Theorem

Brief measure theory

Measures subsets of $\mathbb{R}$

Borel subsets $\mathcal{B}(\mathbb{R})$ as closure under:

- Intervals $[a, b]$.
- Countable unions.
- Complements.

$\varphi : \mathbb{R} \to \mathbb{R}$ is **measurable** when:

$$B \in \mathcal{B}(\mathbb{R}) \qquad \Longrightarrow \qquad \varphi^{-1}[B] \in \mathcal{B}(\mathbb{R})$$

Measure theory generalises to σ-algebras over arbitrary sets.

Source of randomness

Key idea

Propagating randomness from discrete and continuous sampling:

$$\alpha : \mathbb{R} \to X$$

along "random elements":

- for measurable spaces: **derived** through measurable functions;
- for quasi-Borel spaces: **axiomised** through structure.

The category $\mathbf{Qbs}$

Objects

A **quasi-Borel space** $X = \left\langle |X|, X^{\mathbb{R}} \right\rangle$ consists of:

- a **carrier set** X;
- a set of **random elements** $X^{\mathbb{R}} \subseteq |X|^{\mathbb{R}}$

such that the random elements are closed under:

The category **Qbs**

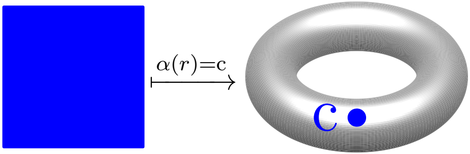
Objects

A **quasi-Borel space** $X = \left\langle |X|, X^{\mathbb{R}} \right\rangle$ consists of:

- a **carrier set** X;
- a set of **random elements** $X^{\mathbb{R}} \subseteq |X|^{\mathbb{R}}$

such that the random elements are closed under:

- constant functions $\underline{c}$;



The category $\mathbf{Qbs}$

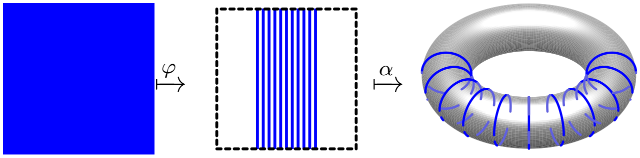
Objects

A **quasi-Borel space** $X = \left\langle |X|, X^{\mathbb{R}} \right\rangle$ consists of:

- a **carrier set** X;
- a set of **random elements** $X^{\mathbb{R}} \subseteq |X|^{\mathbb{R}}$

such that the random elements are closed under:

- constant functions $\underline{c}$;
- precomposition with a measurable $\varphi : \mathbb{R} \to \mathbb{R}$



The category Qbs

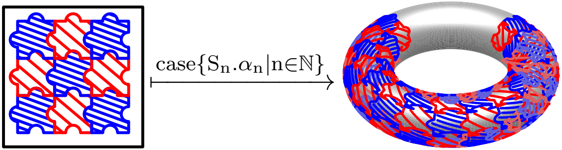
Objects

A **quasi-Borel space** $X = \left\langle |X|, X^{\mathbb{R}} \right\rangle$ consists of:

- a **carrier set** X;
- a set of **random elements** $X^{\mathbb{R}} \subseteq |X|^{\mathbb{R}}$

such that the random elements are closed under:

- constant functions $\underline{c}$;
- precomposition with a measurable $\varphi : \mathbb{R} \to \mathbb{R}$
- countable measurable case split.



The category $\mathbf{Qbs}$

Morphisms $f : X \to Y$

Functions $f : |X| \to |Y|$ such that:

$$\alpha \in X^{\mathbb{R}} \qquad \Longrightarrow \qquad f \circ \alpha \in Y^{\mathbb{R}}$$

Categorical structure

Free and cofree spaces

Equip a set $A \in \mathbf{Set}$ with:

$$(\mathsf{Free}\quad A)^{\mathbb{R}} := \left\{\mathsf{case}\,\{S_n.\underline{a}_n | n \in \mathbb{N}\} \middle| (S_n) \text{ a measurable partition}\right\}$$
$$(\mathsf{Cofree}A)^{\mathbb{R}} := A^{\mathbb{R}}$$

$$\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "\mathsf{Free}"] \arrow[r, bend right, "\mathsf{Cofree}"'] & \mathbf{Qbs} \arrow[l] \end{tikzcd}$$

Categorical structure

Measurable spaces

Adjunction with measurable spaces ($M \in \mathbf{Meas}$, $X \in \mathbf{Qbs}$):

$$(\mathbf{Qbs}M)^{\mathbb{R}} := \mathbf{Meas}(\mathbb{R}, M)$$
$$\Sigma_{(\mathbf{Meas}X)} := \left\{ B \subseteq X \middle| \forall \alpha \in X^{\mathbb{R}}, \alpha^{-1}[X] \in \mathcal{B}(\mathbb{R}) \right\}$$

$$\begin{tikzcd}
\mathbf{Qbs} \arrow[rr, bend left, "\mathbf{Meas}"] \arrow[rr, phantom, "\perp"] \arrow[dr] & \arrow[d, phantom, "="] & \mathbf{Meas} \arrow[ll, bend left, "\mathbf{Qbs}"] \arrow[dl] \\
& \mathbf{Set} &
\end{tikzcd}$$

NB: $\mathbf{Meas} \circ \mathbf{Qbs}X = X$ for standard Borel spaces X.

Categorical structure

Products

Correlated random elements:

$$(X \times Y)^{\mathbb{R}} := \left\{ r \mapsto \left\langle \alpha(r), \beta(r) \right\rangle \middle| \alpha \in X^{\mathbb{R}}, \beta \in Y^{\mathbb{R}} \right\}$$

Categorical structure

Function spaces

$$\left|Y^X\right| := \mathbf{Qbs}(X, Y)$$
$$(Y^X)^{\mathbb{R}} := \left\{ f : \mathbb{R} \to \left|Y^X\right| \,\middle|\, \text{uncurry}\, f \in \mathbf{Qbs}(\mathbb{R} \times X, Y) \right\}$$

NB: The exponential $X^{\mathbb{R}}$ is the space of random elements.

Categorical structure

Function spaces

$$\left|Y^X\right| := \mathbf{Qbs}(X, Y)$$
$$(Y^X)^{\mathbb{R}} := \left\{ f : \mathbb{R} \to \left|Y^X\right| \,\middle|\, \text{uncurry } f \in \mathbf{Qbs}(\mathbb{R} \times X, Y) \right\}$$

NB: The exponential $X^{\mathbb{R}}$ is the space of random elements.

More structure

Coproducts, limits, colimits, ...

Probabilistic structure

Probability distributions on X

Pairs $\langle \alpha, \mu \rangle$:

- random element $\alpha \in X^{\mathbb{R}}$
- probability distribution μ on $\mathbb{R}$

Too intensional, e.g.:

$$(x \mapsto \min(0, \max(1, \quad x)), \mathbf{U_{[0,1]}})$$

vs.

$$(x \mapsto \min(0, \max(1, 1-x)), \mathbf{U_{[0,1]}})$$

Probabilistic structure

Quotienting distributions

$\langle \alpha, \mu \rangle \sim \langle \beta, \nu \rangle$ when $\alpha_* \mu = \beta_* \nu$ in $\mathbf{Meas} X$ and set

- $|PX|$: $\sim$-equivalence classes of probability distributions.
- $(PX)^{\mathbb{R}} :=$
  $\left\{ r \mapsto [\alpha, k(r)] \middle| \; k : \mathbb{R} \times \mathcal{B}(\mathbb{R}) \to [0,1] \text{ is a Markov kernel } \right\}$

NB: PX coincides with Giry on standard Borel spaces X

Contribution

Quasi-Borel spaces

A Cartesian closed category for higher-order probability theory:

- Sets-with-structure ('random elements') and structure preserving functions.
- Well-behaved and concrete categorical structure.
- Well-behaved probabilistic structure: randomization and de Finetti's Theorem

Contribution

Quasi-Borel spaces

A Cartesian closed category for higher-order probability theory:

- Sets-with-structure (‘random elements’) and structure preserving functions.
- Well-behaved and concrete categorical structure.
- Well-behaved probabilistic structure: randomization and de Finetti’s Theorem

Thank you!

FAQ

Difference with previous category [LICS'16]

$$\begin{tikzcd}
\mathbf{SMeas} \arrow[rr, hook, "\mathbf{Qbs}"] \arrow[dr, hook, "\mathbf{y}X=\mathbf{SMeas}(-,X)"'] & {=} & \mathbf{Qbs} \arrow[dl, hook', "\mathbf{Meas}(-,\mathbf{Meas}X)"] \\
& {[\mathbf{SMeas}^{\mathrm{op}}, \mathbf{Set}]_{\prod_\omega}} &
\end{tikzcd}$$

Every $Q \in [\mathbf{SMeas}^{\mathrm{op}}, \mathbf{Set}]_{\prod_\omega}$ has the data:

- $Q\mathbb{1}$: elements.
- $Q\mathbb{R}$: random elements.
- $eval : Q\mathbb{R} \xrightarrow{(Q\underline{r})_{r\in\mathbb{R}}} \prod_{r\in\mathbb{R}} Q\mathbb{1}$.

A quasi-Borel space is fully determined by this data.

Thank you!

FAQ

Why call them "quasi-Borel spaces"?

$$\begin{tikzcd}
\mathbf{SMeas} \arrow[rr, hook, "\mathbf{Qbs}"] \arrow[dr, hook, "\mathbf{y}X = \mathbf{SMeas}(-,X)"'] & \arrow[d, phantom, "="] & \mathbf{Qbs} \arrow[dl, hook', "\mathbf{Meas}(-,\mathbf{Meas}X)"] \\
& {[\mathbf{SMeas}^{\mathrm{op}}, \mathbf{Set}]_{\prod_\omega}} &
\end{tikzcd}$$

Analogous to Spanier's quasi-topological spaces that arise similarly:

- Sheaf (in this case): countable product preservation
- Separatedness: $eval : Q\mathbb{R} \xrightarrow{(Q\underline{r})_{r\in\mathbb{R}}} \prod_{r\in\mathbb{R}} Q\mathbb{1}$ injective

Thank you!

FAQ

What about recursion?

- Sub-probability distributions can deal with first-order iteration
- We are currently working on type recursion.