Programming language foundations for statistics

Sam Staton, Oxford

partly based on joint work with Ackerman, Dash, Freer, Jacobs, Kaddar, Moss, Paquet, Perrone, Roy, Sabok, Stein, Wolman, Yang, and others.
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics
   example; discussion; Monte Carlo

2. Function spaces ...

3. ... and understanding them.

4. Symmetries
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?

Clue: it’s not 51%! 

Simon Walker / HM Treasury & Simon Dawson / No10 Downing Street
Open Government Licence v3.0
High level view: poll example

A very simple model deducing chance of win from poll.

```haskell
model :: Prob ([Bool] , Bool)
model = do
  voteShare <- uniform 0 1
  votes <- repeat (bernoulli voteShare)
  return (take 100 votes , (voteShare > 0.5))
```
High level view: poll example

A very simple model deducing chance of win from poll.

```haskell
model :: Prob ([Bool] , Bool)
model = do
  voteShare <- uniform 0 1
  votes <- repeat (bernoulli voteShare)
  return (take 100 votes , (voteShare > 0.5))
```

Crude rejection sampling Monte Carlo:

- Run 1000000s of times, each time getting (poll result, win?)
- Reject the runs that mis-predict poll
- What proportion of the remainder are winners?

![Bar chart showing # non-rejected runs with Win and Lose categories]
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?

Answer: 0.579.

Crude rejection sampling Monte Carlo:
- Run 1000000s of times, each time getting (poll result, win?)
- Reject the runs that mis-predict poll
- What proportion of the remainder are winners?
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?

Answer: 0.579.

(See Andrew Gelman and coauthors for a proper discussion of using PPL for election modelling.)
Probabilistic programming in practice

Applications to social science, biology, physical sciences, machine learning

Church, Anglican, Hakaru, MonadBayes, Gen...
LazyPPL
https://lazyppl.bitbucket.io

Dash, Kaddar, Paquet, Staton, POPL 2023
Abstraction in traditional programming

High level  e.g. higher-order functions
           abstract types

Low level  e.g. machine code,
           Boolean circuits
Abstraction in traditional programming

High level e.g. higher-order functions, abstract types

Low level e.g. machine code, Boolean circuits

<table>
<thead>
<tr>
<th></th>
<th>Engineering</th>
<th>Foundational</th>
</tr>
</thead>
<tbody>
<tr>
<td>High level</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Low level</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Abstraction in *probabilistic* programming

<table>
<thead>
<tr>
<th>High level</th>
<th>e.g. infinite dimensional systems</th>
<th>higher-order functions</th>
<th>abstract types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low level</td>
<td>e.g. bets, frequencies, decisions</td>
<td>Monte Carlo simulation</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ML / stats apps</th>
<th>Foundational</th>
</tr>
</thead>
<tbody>
<tr>
<td>High level</td>
<td>✔</td>
</tr>
<tr>
<td>Low level</td>
<td>✔</td>
</tr>
</tbody>
</table>
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics example; discussion; Monte Carlo

2. Function spaces ...

3. ... and understanding them.

4. Symmetries
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?

Answer: 0.579.

(See Andrew Gelman and coauthors for a proper discussion of using PPL for election modelling.)
High level view: poll example

A very simple model deducing chance of win from poll.

Question:
A quick poll gives 51:49 votes. What is the chance of winning?

Answer: 0.579.

Crude rejection sampling Monte Carlo:
- Run 1000000s of times, each time getting (poll result, win?)
- Reject the runs that mis-predict poll
- What proportion of the remainder are winners?
Towards weighted sampling

A very simple model deducing chance of win from poll.

```haskell
model :: Prob ([Bool] , Bool)
model = do
  voteShare <- uniform 0 1
  votes <- repeat (beroulli voteShare)
  return (take 100 votes , (voteShare > 0.5))
```

Crude rejection sampling Monte Carlo:

- Run 1000000s of times, each time getting (poll result, win?)
- Reject the runs that mis-predict poll
- What proportion of the remainder are winners?
Towards weighted sampling

A very simple model deducing chance of win from poll.

```plaintext
model :: Prob ([Bool], Bool)
model = do
    voteShare <- uniform 0 1
    votes <- repeat (bernoulli voteShare)
    return (take 100 votes, (voteShare > 0.5))
```
Weighted sampling

\[
\text{model :: Prob ([Bool], Bool)}
\]
\[
\text{model = do}
\]
\[
\text{voteShare <- uniform 0 1}
\]
\[
\text{forM poll (actualVote->}
\]
\[
\text{score (bernoulliPdf voteShare actualVote))}
\]
\[
\text{return (voteShare > 0.5)}
\]

\[
\text{likelihood}(v) = v^{51}(1 - v)^{49}
\]
Weighted sampling

model :: Prob ([Bool], Bool)
model = do
    voteShare <- uniform 0 1
    forM poll (\actualVote->
        score (bernoulliPdf voteShare actualVote))
    return (voteShare > 0.5)

**Weighted** Monte Carlo:

- Run 1000000s of times, each time getting *(win?)*
- Each time pick a `voteShare`, and weight by the *likelihood*.
- Find weighted proportion of winners.
Weighted sampling

\text{likelihood}(v) = v^{51}(1 - v)^{49}

Area under curve $= \frac{49! \cdot 51!}{101!} 

\text{Green proportion (win)} \approx 0.579
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics

2. Function spaces ...
   Examples; higher-order functions

3. ... and understanding them.

4. Symmetries
Abstraction in *probabilistic* programming

<table>
<thead>
<tr>
<th></th>
<th>High level</th>
<th>Low level</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. infinite dimensional systems</td>
<td>☑</td>
<td></td>
</tr>
<tr>
<td>higher-order functions</td>
<td>☑</td>
<td>e.g. bets, frequencies, decisions</td>
</tr>
<tr>
<td>abstract types</td>
<td></td>
<td>Monte Carlo simulation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ML / stats apps</th>
<th>Foundational</th>
</tr>
</thead>
<tbody>
<tr>
<td>High level</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Low level</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>
Random linear functions

\[
\text{randlinear :: Prob} \ (\text{RealNum} , \text{RealNum})
\]
\[
\text{randlinear } =
\]
\[
\text{do } a \leftarrow \text{normal} \ 0 \ 3
\]
\[
b \leftarrow \text{normal} \ 0 \ 3
\]
\[
\text{return} \ (a,b)
\]

\[
\text{type RealNum = Double}
\]
Random linear functions

```haskell
randlinear :: Prob (RealNum, RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     return (a, b)
```

We will use this for a regression problem:

which function probably generated these points?
Random linear functions

\[
\text{randlinear} :: \text{Prob (RealNum, RealNum)}
\]

\[
\text{randlinear} = \\
\quad \text{do } a \leftarrow \text{normal 0 3} \\
\quad b \leftarrow \text{normal 0 3} \\
\quad \text{return } (a,b)
\]
Random linear functions

```haskell
calendar :: Prob (RealNum -> RealNum)
calendar =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f
```

10000 samples
Random linear functions

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
    do a <- normal 0 3
       b <- normal 0 3
       let f x = a*x + b
       return f
```

100 samples
Bayesian regression

```
randlinear :: Prob (RealNum -> RealNum)

regress :: RealNum -> Prob (a -> RealNum) -> [(a,RealNum)] -> Meas (a -> RealNum)
regress sigma prior dataset =
  do f <- sample prior
     forM dataset (\(x,y) -> score $ normalPdf (f x) sigma y)
     return f
```

lazyppl includes a type

```
Meas a
```
of unnormalized

measures and

```
mh
```
a Metropolis-Hastings

inference method.
Random linear functions

randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f

100 samples
Types as spaces of distributions

randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f

There’s a type constructor Prob (a monad), and...

- Prob RealNum contains probability distributions (e.g. normal 0 3, uniform 0 1)
- Prob Bool contains probability distributions like bernoulli 0.5
Types as spaces of distributions

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f
```

There’s a type constructor **Prob** (a monad), and...

- **Prob RealNum** contains probability distributions (e.g. `normal 0 3`, `uniform 0 1`)
Types as spaces of distributions

\[
\text{randlinear} :: \text{Prob} \ (\text{RealNum} \to \text{RealNum}) \\
\text{randlinear} = \\
\quad \text{do} \ a \leftarrow \text{normal} \ 0 \ 3 \\
\quad \quad \ b \leftarrow \text{normal} \ 0 \ 3 \\
\quad \quad \ \text{let} \ f \ x = a \times x + b \\
\quad \return \ f
\]

There’s a type constructor \textbf{Prob} (a monad), and...

- \textbf{Prob} \ \textbf{RealNum} contains probability distributions (e.g. \text{normal} \ 0 \ 3, \text{uniform} \ 0 \ 1)
- \textbf{normal} :: \textbf{RealNum} \to \textbf{RealNum} \to \text{Prob} \ \textbf{RealNum}
  is a parameterized distribution
- \textbf{bernoulli} :: \textbf{RealNum} \to \text{Prob} \ \text{Bool}
  is a parameterized distribution too
Types as spaces of distributions

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a * x + b
     return f

There's a type constructor \texttt{Prob} (a monad), and...

\begin{itemize}
  \item \texttt{Prob RealNum} contains probability distributions (e.g. \texttt{normal 0 3}, \texttt{uniform 0 1})
  \item \texttt{RealNum -> Prob RealNum} contains parameterized distributions (e.g. \texttt{normal 0})
\end{itemize}
```
Types as spaces of distributions

\[
\text{randlinear} :: \text{Prob} \ (\text{RealNum} \to \text{RealNum})
\]

\[
\text{randlinear} =
\]
\[
\text{do} \ a \leftarrow \text{normal} \ 0 \ 3
\]
\[
\phantom{\text{do} \ a} b \leftarrow \text{normal} \ 0 \ 3
\]
\[
\phantom{\text{do} \ a \ b} \text{let} \ f \ x = a \ast x + b
\]
\[
\phantom{\text{do} \ a \ b \ f} \text{return} \ f
\]

There’s a type constructor \text{Prob} (a monad), and...

- \text{Prob} \ \text{RealNum} \ contains \ probability \ distributions \ (e.g. \ \text{normal} \ 0 \ 3, \ \text{uniform} \ 0 \ 1)
- \text{RealNum} \to \text{Prob} \ \text{RealNum} \ contains \ parameterized \ distributions \ (e.g. \ \text{normal} \ 0)
- \text{Prob} \ (\text{RealNum} \to \text{RealNum}) \ contains \ random \ functions \ (e.g. \ \text{randlinear})
Types as spaces of distributions

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f
     :: RealNum -> RealNum
```

There’s a type constructor `Prob` (a monad), and...

- `Prob RealNum` contains probability distributions (e.g. `normal 0 3`, `uniform 0 1`)
- `RealNum -> Prob RealNum` contains parameterized distributions (e.g. `normal 0`)
- `Prob (RealNum -> RealNum)` contains random functions (e.g. `randlinear`)
Types as spaces of distributions

```haskell
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 3
     b <- normal 0 3
     let f x = a*x + b
     return f
:: Prob (RealNum -> RealNum)
```

There’s a type constructor `Prob` (a monad), and...

- **Prob RealNum** contains probability distributions (e.g. `normal 0 3`, `uniform 0 1`)
- **RealNum -> Prob RealNum** contains parameterized distributions (e.g. `normal 0`)
- **Prob (RealNum -> RealNum)** contains random functions (e.g. `randlinear`)
Types as spaces of distributions

```haskell
randlinear :: Prob (RealNum -> RealNum)
randlinear =
d o a <- normal 0 3
    b <- normal 0 3
    let f x = a*x + b
    return f
```

There’s a type constructor `Prob` (a monad), and...

- `Prob RealNum` contains probability distributions (e.g. `normal 0 3`, `uniform 0 1`)
- `RealNum -> Prob RealNum` contains parameterized distributions (e.g. `normal 0`)
- `Prob (RealNum -> RealNum)` contains random functions (e.g. `randlinear`)

lazyppl.bitbucket.io
Types as spaces of distributions

\[
\text{randlinear} :: \text{Prob} \ (\text{RealNum} \rightarrow \text{RealNum}) \\
\text{randlinear} = \\
\quad \text{do } a \leftarrow \text{normal} \ 0 \ 3 \\
\quad \quad b \leftarrow \text{normal} \ 0 \ 3 \\
\quad \quad \text{let } f \ x = a \times x + b \\
\quad \text{return } f
\]

There’s a type constructor \text{Prob} (a monad), and...

- \text{Prob RealNum} contains probability distributions (e.g. \text{normal} \ 0 \ 3, \text{uniform} \ 0 \ 1)
- \text{RealNum} \rightarrow \text{Prob RealNum} contains parameterized distributions (e.g. \text{normal} \ 0)
- \text{Prob} (\text{RealNum} \rightarrow \text{RealNum}) contains random functions (e.g. \text{randlinear})
- \text{Prob} (\text{Prob Bool}) contains random distributions, etc..
Types as spaces of distributions

\[ \text{randlinear} :: \text{Prob} \ (\text{RealNum} \rightarrow \text{RealNum}) \]

\[
\begin{align*}
\text{randlinear} = \\
& \quad \text{do} \ a \leftarrow \text{normal} \ 0 \ 3 \\
& \quad \ b \leftarrow \text{normal} \ 0 \ 3 \\
& \quad \text{let} \ f \ x = a \times x + b \\
& \quad \text{return} \ f
\end{align*}
\]

There’s a type constructor \( \text{Prob} \) (a monad), and...

- \( \text{Prob RealNum} \) contains probability distributions (e.g. \( \text{normal 0 3}, \text{uniform 0 1} \))
- \( \text{RealNum} \rightarrow \text{Prob RealNum} \) contains parameterized distributions (e.g. \( \text{normal 0} \))
- \( \text{Prob (RealNum} \rightarrow \text{RealNum))} \) contains random functions (e.g. \( \text{randlinear} \))
- \( \text{Prob (Prob Bool)} \) contains random distributions, etc..

**Challenge:**

Aumann (1961) showed that measure-theoretic probability does not support function spaces properly!
Random functions & program synthesis

data Expr = Var | Constt RealNum | Add Expr Expr | Mult Expr Expr | IfLess RealNum Expr Expr

eval :: Expr -> (RealNum -> RealNum)
eval Var x = x
eval (Constt r) _ = r
eval (Add e1 e2) x = (eval e1 x) + (eval e2 x)
eval (Mult e1 e2) x = (eval e1 x) * (eval e2 x)
eval (IfLess r e1 e2) x = if x < r then eval e1 x else eval e2 x
Random functions & program synthesis

data Expr = Var | Constt RealNum | Add Expr Expr | Mult Expr Expr | IfLess RealNum Expr Expr

eval :: Expr -> (RealNum -> RealNum)
eval Var x = x
eval (Constt r) _ = r
eval (Add e1 e2) x = (eval e1 x) + (eval e2 x)
eval (Mult e1 e2) x = (eval e1 x) * (eval e2 x)
eval (IfLess r e1 e2) x = if x < r then eval e1 x else eval e2 x

randexpr :: Prob Expr

randprog :: Prob (RealNum -> RealNum)
randprog = do e <- randexpr
doo e <- randexpr
    return (eval e)
Random functions & program synthesis

data Expr = …

eval :: Expr -> (RealNum -> RealNum)

randexpr :: Prob Expr

randprog :: Prob (RealNum -> RealNum)
randprog = do e <- randexpr
           e <- randexpr
           return (eval e)
data Expr = ...  

eval :: Expr -> (RealNum -> RealNum)  

randexpr :: Prob Expr  

randprog :: Prob (RealNum -> RealNum)  

randprog = do e <- randexpr  
               return (eval e)  

mh 0.1 (regress 0.25 randprog dataset)
Gaussian processes as random functions

\[
\text{wiener} :: \text{Prob} \ (\text{RealNum} \rightarrow \text{RealNum})
\]

mh (regress 0.3 wiener dataset)
Gaussian processes as random functions

gprbf :: Prob (RealNum -> RealNum)

mh (regress 0.3 gprbf dataset)
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics

2. Function spaces ...
   Examples; higher-order functions

3. ... and understanding them.

4. Symmetries
**Piecewise constant regression**

**Defn.** A *point process* on $a$ is an inhabitant of $\text{Prob} \ [a]$ (or $\text{Prob} (\text{Bag} \ a)$).

**Idea:** Fit a piecewise constant function where the change-points come from a point process.
Defn. A point process on $a$ is an inhabitant of $\text{Prob} \ [a]$ (or $\text{Prob} \ (\text{Bag} \ a)$).

Idea: Fit a piecewise linear function where the change-points come from a point process.
**Piecewise linear regression**

**Defn.** A *point process* on $a$ is an inhabitant of $\text{Prob} \ [a]$ (or $\text{Prob} \ (\text{Bag } a)$).

**Idea:** Fit a piecewise linear function where the change-points come from a point process.

**What is "piecewise"?**
Defn. A point process on \( a \) is an inhabitant of \( \text{Prob} \ [a] \) (or \( \text{Prob} \ \text{(Bag} \ a) \)).

\[
\begin{align*}
\text{randconst} & \quad : \quad \text{Prob} \ (\text{RealNum} \rightarrow \text{RealNum}) \\
\text{randconst} & = \\
& \quad \text{do} \ a \leftarrow \text{normal} \ 0 \ 5 \\
& \quad \text{let} \ f \ x = a \\
& \quad \text{return} \ f
\end{align*}
\]
Defn. A *point process* on $a$ is an inhabitant of $\text{Prob } [a]$ (or $\text{Prob } \text{(Bag } a\text{)})$.

e.g. $\text{poissonPP} :: \text{RealNum} \rightarrow \text{RealNum} \rightarrow \text{Prob } [\text{RealNum}]

$\text{randconst} :: \text{Prob } (\text{RealNum} \rightarrow \text{RealNum})$

$\text{randconst} =$

$\text{do } a \leftarrow \text{normal } 0 \text{ 5} $

$\text{let } f \ x = a$

$\text{return } f$

$\text{splice} :: \text{Prob } [\text{RealNum}] \rightarrow$

$\text{Prob } (\text{RealNum} \rightarrow \text{RealNum}) \rightarrow$

$\text{Prob } (\text{RealNum} \rightarrow \text{RealNum})$

$\text{mh (regress } 0.1 \text{ (splice (poissonPP 0 0.1) randconst) dataset)}$
**Piecewise constant regression**

**Defn.** A *point process* on a is an inhabitant of \( \text{Prob} \ [a] \) (or \( \text{Prob} \ (\text{Bag} \ a) \)).

*E.g.* \( \text{poissonPP} :: \text{RealNum} \to \text{RealNum} \to \text{Prob} \ [\text{RealNum}] \)

\( \text{randlinear} :: \text{Prob} \ (\text{RealNum} \to \text{RealNum}) \)

\( \text{randlinear} = \)
\[
\begin{aligned}
\text{do} \ &a \leftarrow \text{normal} \ 0 \ 3 \\
\ &b \leftarrow \text{normal} \ 0 \ 3 \\
\ &\text{let} \ f \ x = a \times x + b \\
\ &\text{return} \ f
\end{aligned}
\]

\( \text{splice} :: \text{Prob} \ [\text{RealNum}] \to\)
\[
\begin{aligned}
\ &\text{Prob} \ (\text{RealNum} \to \text{RealNum}) \\
\ &\text{Prob} \ (\text{RealNum} \to \text{RealNum})
\end{aligned}
\]
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics

2. Function spaces ...

3. ... and understanding them. 
   *models in the abstract* ; quasi-Borel spaces

4. Symmetries
# Curry-Howard correspondence

<table>
<thead>
<tr>
<th>Programming</th>
<th>Maths</th>
<th>Category theory</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Spaces</td>
<td>Objects</td>
<td>Propositions</td>
</tr>
<tr>
<td>Programs</td>
<td>Continuous functions</td>
<td>Morphisms</td>
<td>Proofs</td>
</tr>
</tbody>
</table>
## Curry-Howard correspondence

<table>
<thead>
<tr>
<th>Programming</th>
<th>Maths</th>
<th>Category theory</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Spaces</td>
<td>Objects</td>
<td>Propositions</td>
</tr>
<tr>
<td>Programs</td>
<td>Continuous functions</td>
<td>Morphisms</td>
<td>Proofs</td>
</tr>
<tr>
<td>Probabilistic programs</td>
<td>Measures</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Desiderata for a theory of Prob

Dataflow property:

*Program lines can be reordered and discarded if dataflow is preserved.*
Desiderata for a theory of Prob

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 2
     b <- normal 0 3
     let f x = a*x + b
     return f
```

Dataflow property:

*Program lines can be reordered and discarded if dataflow is preserved.*
Desiderata for a theory of Prob

```haskell
randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 2
     b <- normal 0 3
     do b <- normal 0 3
        a <- normal 0 2
     let f x = a*x + b
     return f
  let f x = a*x + b
  return f
```

Dataflow property:

*Program lines can be reordered and discarded if dataflow is preserved.*
Desiderata for a theory of Prob

randlinear :: Prob (RealNum -> RealNum)
randlinear =
  do a <- normal 0 2
     b <- normal 0 3
     let f x = a*x + b
     return f
  do b <- normal 0 3
     a <- normal 0 2
     let f x = a*x + b
     return f

Dataflow property:
Program lines can be reordered and discarded if dataflow is preserved.
Desiderata for a theory of Prob

randlinear :: Prob (RealNum -> RealNum)
randlinear =
    do a <- normal 0 2
       b <- normal 0 3
       let f x = a*x + b
       return f

Dataflow property:

Program lines can be reordered and discarded if dataflow is preserved.
Desiderata for a theory of Prob

\[
\text{randlinear} :: \text{Prob} \ (\text{RealNum} -> \text{RealNum})
\]

\[
\text{randlinear} =
\begin{align*}
do \ a &\leftarrow \text{normal} \ 0 \ 2 \\
b &\leftarrow \text{normal} \ 0 \ 3 \\
let \ f \ x = a*x + b \\
\end{align*}
\]

\[
\begin{align*}
\text{return} \ f \\
\end{align*}
\]

Dataflow property:

Program lines can be reordered and discarded if dataflow is preserved.
Desiderata for a theory of Prob

\[
\int \int k(\lambda x. ax + b) \, db \, da \quad \int \int k(\lambda x. ax + b) \, da \, db
\]

Dataflow property:

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

Related to Fubini’s theorem.
Desiderata for a theory of Prob

\[
\int \int k(\lambda x. ax + b) \, db \, da
\]

\[
\int \int k(\lambda x. ax + b) \, da \, db
\]

\[
\int \int \int k(\lambda x. ax + b) \, da \, db \, dc
\]

Dataflow property:

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

Related to Fubini’s theorem.

Also related to
Kock TAC 2012
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics

2. Function spaces ...

3. ... and understanding them.
   models in the abstract; quasi-Borel spaces

4. Symmetries
A semantic model:

Quasi-Borel spaces

Heunen, Kammar, Staton, Yang, LICS 2017

There’s a type constructor \( \text{Prob} \) (a monad), and...

- \( \text{Prob} \text{ RealNum} \) contains probability distributions (e.g. \( \text{normal} \ 0 \ 3 \), \( \text{uniform} \ 0 \ 1 \))
- \( \text{RealNum} \rightarrow \text{Prob} \text{ RealNum} \) contains parameterized distributions (e.g. \( \text{normal} \ 0 \))
- \( \text{Prob} \ (\text{RealNum} \rightarrow \text{RealNum}) \) contains random functions (e.g. \( \text{randlinear} \))
- The dataflow property holds.
Other options:

- Domain-theoretic models; 
- Linear-logic based models; 
- Topological-domain-based models...

For now: quasi-Borel spaces

Inspired by:

- Logical relations
- Quasi-topological spaces, diffeological spaces, sequential spaces...

see also Matache, Moss, Staton, LICS 2022
Quasi-Borel spaces

**Defn.** A *quasi-Borel space* is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \to X]$ such that...

**Lemma.** One uniform distribution is sufficient to generate all probability measures*.

```plaintext
do { r <- uniform ; return (α r) }
```
Quasi-Borel spaces

Defn. A quasi-Borel space is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \rightarrow X]$ such that...

Types: quasi-Borel spaces.

Programs: morphisms, i.e. functions $f : X \rightarrow Y$ such that

$$f \circ M_X \subseteq M_Y$$

Lemma. One uniform distribution is sufficient to generate all probability measures$^*$. 

```latex
do { r <- uniform ; return (α r) }
```
Quasi-Borel spaces

**Defn.** A quasi-Borel space is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \to X]$ such that...

**Defn.** A probability measure on a qBs $(X, M_X)$ is a function in $M_X$ modulo $\sim$.

**Types :** quasi-Borel spaces.

**Programs :** morphisms, i.e. functions $f : X \to Y$ such that

$$f \circ M_X \subseteq M_Y$$

**Lemma.** One uniform distribution is sufficient to generate all probability measures*.

```plaintext
do { r <- uniform ; return (α r) }
```
**Quasi-Borel spaces**

**Defn.** A quasi-Borel space is a set \( X \) equipped with a set of random elements, \( M \subseteq \mathbb{R} \rightarrow X \) such that...

**Types :** quasi-Borel spaces.

**Programs :** morphisms, i.e. functions \( f : X \rightarrow Y \) such that 
\[
f \circ M_X \subseteq M_Y
\]

**Lemma.** One uniform distribution is sufficient to generate all probability measures*.

\[
\text{do } \{ \ r \leftarrow \text{uniform} ; \ \text{return} \ (\alpha \ r) \ \} 
\]

**Defn.** A probability measure on a qBs \((X, M_X)\) is a function in \( M_X \) modulo \( \sim \).
Quasi-Borel spaces

**Defn.** A quasi-Borel space is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \to X]$ such that...

**Types:** quasi-Borel spaces.

**Programs:** morphisms, i.e. functions $f : X \to Y$ such that

$$f \circ M_X \subseteq M_Y$$

**Lemma.** One uniform distribution is sufficient to generate all probability measures*.

\[
\text{do } \{ \ r \leftarrow \text{uniform} ; \ \text{return} \ (\alpha \ r) \ \}\]

**Defn.** A probability measure on a qBs $(X, M_X)$ is a function in $M_X$ modulo $\sim$.
**Quasi-Borel spaces**

**Defn.** A *quasi-Borel space* is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \to X]$ such that...

**Types:** quasi-Borel spaces.

**Programs:** morphisms, i.e. functions $f : X \to Y$ such that

$$f \circ M_X \subseteq M_Y$$

**Lemma.** One uniform distribution is sufficient to generate all probability measures*.

$$\text{do } \{ r \leftarrow \text{uniform} ; \text{return } (\alpha \ r) \}$$

**Defn.** A *probability measure on a qBs* $(X, M_X)$ is a function in $M_X$ modulo $\sim$.

The qBs of reals $(\mathbb{R}, M_{\mathbb{R}})$ has $M_{\mathbb{R}} \subseteq [\mathbb{R} \to \mathbb{R}]$ as the Borel functions.
Defn. A quasi-Borel space is a set $X$ equipped with a set of random elements, $M \subseteq [\mathbb{R} \to X]$ such that...

Types: quasi-Borel spaces.

$\text{RealNum} \equiv \mathbb{R}$

$\text{Prob a} \equiv Pr([\text{a}])$

Programs: morphisms, i.e. functions $f : X \to Y$ such that

$$f \circ M_X \subseteq M_Y$$

Defn. A probability measure on a qBs $(X, M_X)$ is a function in $M_X$ modulo $\sim$.

The qBs of reals $(\mathbb{R}, M_{\mathbb{R}})$ has $M_{\mathbb{R}} \subseteq [\mathbb{R} \to \mathbb{R}]$ as the Borel functions.
Desiderata for a theory of Prob

randlinear :: Prob (RealNum -> RealNum)

randlinear =
do a <- normal 0 3
  b <- normal 0 3
  let f x = a*x + b
  return f

Theorem. The quasi-Borel space model satisfies the dataflow property.

\[
\int \int k(\lambda x. ax + b) \, db \, da
\]

\[
\int \int \int k(\lambda x. ax + b) \, da \, db \, dc
\]

Dataflow property:

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

Related to Fubini’s theorem.
Desiderata for a theory of Prob

Theorem. The quasi-Borel space model satisfies the dataflow property.

- The probability monad is commutative and affine. \(^{\text{cf Kock TAC 2012}}\)
- The parameterized distributions form a monoidal category \(^{\text{cf Fritz Adv Math 2020, Cho & Jacobs MSCS 2019, Stein & Staton LICS 2021}}\)

Dataflow property:

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

Related to Fubini’s theorem.
repeat in quasi-Borel spaces
A very simple model deducing chance of win from poll.

```haskell
model :: Prob ([Bool], Bool)
model = do
    voteShare <- uniform 0 1
    votes <- repeat (bernoulli voteShare)
    return (take 100 votes, (voteShare > 0.5))
```

Simon Walker / HM Treasury & Simon Dawson / No10 Downing Street
Open Government Licence v3.0
repeat in quasi-Borel spaces

model :: Prob ([Bool], Bool)
model = do
  voteShare <- uniform 0 1
  votes <- repeat (bernoulli voteShare)
  return (take 100 votes , (voteShare > 0.5))

repeat :: Prob a -> Prob [a]

Repeatedly draws from a distribution, forever.

Observation.
In measure theoretic probability, repeat is defined by Kolmogorov extension.
**repeat in quasi-Borel spaces**

\[
\text{model} :: \text{Prob} ([\text{Bool}], \text{Bool}) \\
\text{model} = \text{do} \\
\quad \text{voteShare} \leftarrow \text{uniform} \ 0 \ 1 \\
\quad \text{votes} \leftarrow \text{repeat} \ (\text{bernoulli} \ \text{vo} \\
\quad \text{return} \ (\text{take} \ 100 \ \text{votes}, (\text{vote} \\
\text{repeat} :: \text{Prob} \ a \rightarrow \text{Prob} \ [a] \\
\text{Repeatedly draws from a distribution, forever.} \\

\text{Theorem (summer 2022).} \text{ repeat can be defined for any quasi-Borel space a.}

\text{Observation.} \\
\text{In measure theoretic probability, repeat is defined by Kolmogorov extension.}
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics

2. Function spaces ...

3. ... and understanding them.

4. Symmetries and names
Dataflow symmetries

```
randlinear :: Prob (RealNum -> RealNum)
randlinear =
    do a <- normal 0 2          do b <- normal 0 3
       b <- normal 0 3          a <- normal 0 2
       let f x = a*x + b        let f x = a*x + b
       return f                 return f
```

Dataflow property:
Program lines can be reordered and discarded if dataflow is preserved.
Dataflow symmetries

\[ \text{randlinear} :: \text{Prob (RealNum -> RealNum)} \]
\[ \text{randlinear} = \]
\[ \text{do} \ a \leftarrow \text{normal} \ 0 \ 2 \ \text{do} \ b \leftarrow \text{normal} \ 0 \ 3 \]
\[ b \leftarrow \text{normal} \ 0 \ 3 \ a \leftarrow \text{normal} \ 0 \ 2 \]
\[ \text{let} \ f \ x = a*x + b \ \text{let} \ f \ x = a*x + b \]
\[ \text{return} \ f \ \text{return} \ f \]

\text{de Finetti (1931):}

Independence can be analyzed in terms of reordering ('exchangeability')

Dataflow property:

Program lines can be reordered and discarded if dataflow is preserved.
There is still time to register for the annual LMS/BCS-FACS evening seminar on Thursday 17 November. The speaker:
Staton:
lms.ac.uk/civicrm/event/
Names

let x = fresh-name() in ...

(defmacro two-funcalls (f v)
  (let ((fname gensym))
    (let ((,fname ,f))
      (list (funcall ,fname ,v) (funcall ,fname ,v))))))
Each new customer either sits at a random table or a new table. Chance depends on popularity of tables.
Chinese restaurant process

Each new customer either sits at a random table or a new table. Chance depends on popularity of tables.
Chinese restaurant process

Each new customer either sits at a random table or a new table. Chance depends on popularity of tables.
Each new customer either sits at a random table or a new table. Chance depends on popularity of tables.
Chinese restaurant process

Each new customer either sits at a random table or a new table. Chance depends on popularity of tables.
Example: Non-parametric clustering

Restaurant metaphor:
Each point is a customer, the clusters are the tables.

Non-parametric: we don’t know how many clusters.
Example: Non-parametric clustering

Restaurant metaphor:
Each point is a customer, the clusters are the tables.

Non-parametric: we don’t know how many clusters.
Theorem:

1. TFDAE: (a) a functor $R : \text{NomSet} \to \text{MeasSp}$ that preserves colimits and finite limits.
   (b) a measurable space w/ measurable diagonal.
2. A choice of atomless measure on the space $R(\mathbb{A})$ induces a symmetric monoidal functor extending $R$, $\text{Kleisli}(\text{NameGeneration}) \to \text{Kleisli}(\text{Giry})$

So: apply $R$ to a nominal model to get a measure-theoretic realization.
Indian buffet process

Each new customer takes a set of dishes. Chance depends on popularity of dishes; sometimes also take some new dishes.

Griffiths & Ghahramani, JMLR 2011
Indian buffet process

Each new customer takes a set of dishes. Chance depends on popularity of dishes; sometimes also take some new dishes.

Griffiths & Ghahramani, JMLR 2011
Each new customer takes a set of dishes. Chance depends on popularity of dishes; sometimes also take some new dishes.

Griffiths & Ghahramani, JMLR 2011
Each new customer takes a set of dishes. Chance depends on popularity of dishes; sometimes also take some new dishes.
Indian buffets for feature extraction

Example: what are the different features of the countries of the world?
Indian buffets for feature extraction

Example: what are the different features of the countries of the world?

Restaurant metaphor:
Each country is a customer, the features are the dishes that they take.

Given experimental data where people say which countries are similar, what are the features?

Navarro & Griffiths, NeurIPS 2006
Indian buffets for feature extraction

Example: what are the different features of the countries of the world?

Restaurant metaphor: Each country is a customer, the features are the dishes that they take.

Given experimental data where people say which countries are similar, what are the features?

Navarro & Griffiths, NeurIPS 2006
Theorem:
1. TFDAE: (a) a functor $R : \text{NomSet} \to \text{MeasSp}$ that preserves colimits and finite limits.
   (b) a measurable space w/ measurable diagonal.
2. A choice of atomless measure on the space $R(\mathbb{A})$ induces a symmetric monoidal functor extending $R$, $\text{Kleisli}(\text{NameGeneration}) \to \text{Kleisli}(\text{Giry})$.

So: apply $R$ to a nominal model to get a measure-theoretic realization.
Translation down to traditional prob.

Theorem:
1. TFDAE: (a) a functor \( R : \text{NomSet} \to \text{MeasSp} \)
   that preserves colimits and finite limits.
   (b) a measurable space w/ measurable diagonal.

2. A choice of atomless measure on the space \( R(\mathbb{A}) \) induces
   a symmetric monoidal functor extending \( R \),
   \( \text{Kleisli}(\text{NameGeneration}) \to \text{Kleisli}(\text{Giry}) \)

Challenge:
New symmetries, new programs: new statistical models

So: apply \( R \) to a nominal model to get a measure-theoretic realization.

cf Sabok, Staton, Stein, Wolman, POPL 2021
Programming language foundations for statistics

1. Quick look at probabilistic programming for statistics
2. Function spaces ...
3. ... and understanding them.
4. Symmetries and names