Ebba: An Embedded DSL for Bayesian Inference

Linköping University, 17 June 2014

Henrik Nilsson

School of Computer Science

University of Nottingham

Joint work with Tom Nielsen, OpenBrain Ltd

Baysig and Ebba (1)

- Baysig is a Haskell-like language for probabilistic modelling and Bayesian inference developed by OpenBrain Ltd:

  `www.bayeshive.com`

Baysig and Ebba (1)

- Baysig is a Haskell-like language for probabilistic modelling and Bayesian inference developed by OpenBrain Ltd:

  `www.bayeshive.com`

- Baysig programs can in a sense be run both "forwards", to simulate probabilisitic processes, and "backwards", to estimate unknown parameters from observed outcomes:

$$
\begin{aligned}
&\mathit{coinFlips} = \mathrm{prob} \\
&\quad p \sim \mathit{uniform}\ 0\ 1 \\
&\quad \mathrm{repeat}\ 10\ (\mathit{bernoulli}\ p)
\end{aligned}
$$

Baysig and Ebba (2)

- This talk investigates:
  - The possibility of implementing a Baysig-like language as a ***shallow embedding*** (in Haskell).
  - Semantics: an appropriate underlying notion of computation for such a language.

Baysig and Ebba (2)

- This talk investigates:
  - The possibility of implementing a Baysig-like language as a ***shallow embedding*** (in Haskell).
  - Semantics: an appropriate underlying notion of computation for such a language.
- The result is Ebba, short for Embedded Baysig.

Baysig and Ebba (2)

- This talk investigates:
  - The possibility of implementing a Baysig-like language as a ***shallow embedding*** (in Haskell).
  - Semantics: an appropriate underlying notion of computation for such a language.
- The result is Ebba, short for Embedded Baysig.
- Ebba is currently very much a prototype and covers only a small part of what Baysig can do.

Why Embedded Languages?

- For the researcher/implementor:
  - Low implementation effort
  - Ease of experimenting with design and semantics

Why Embedded Languages?

- For the researcher/implementor:
  - Low implementation effort
  - Ease of experimenting with design and semantics
- For the users:
  - reuse: familiar syntax, type system, tools . . .
  - facilitates programmatic use
    - use as component
    - metaprogramming
  - interoperability between DSLs

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Example: Embedded language for working with (infinite) streams where:

- integer literal stands for stream of the integer
- arithmetic operations are pointwise operations on streams.

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Example: Embedded language for working with (infinite) streams where:

- integer literal stands for stream of the integer
- arithmetic operations are pointwise operations on streams.

$$[\![\texttt{1 + 2}]\!]$$

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Example: Embedded language for working with (infinite) streams where:

- integer literal stands for stream of the integer
- arithmetic operations are pointwise operations on streams.

$$[\![\texttt{1 + 2}]\!] = [1, 1, 1, \ldots \;\; [\![\texttt{+}]\!] \;\; [2, 2, 2, \ldots$$

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Example: Embedded language for working with (infinite) streams where:

- integer literal stands for stream of the integer
- arithmetic operations are pointwise operations on streams.

$$\begin{aligned} [\![\mathtt{1} \;\; \mathtt{+} \;\; \mathtt{2}]\!] &= [1, 1, 1, \ldots \;\; [\![\mathtt{+}]\!] \;\; [2, 2, 2, \ldots \\ &= [1 + 2, 1 + 2, 1 + 2, \ldots \end{aligned}$$

Why Shallow Embedding for Ebba? (1)

The nub of embedding is repurposing of the host-language syntax one way or another.

Example: Embedded language for working with (infinite) streams where:

- integer literal stands for stream of the integer
- arithmetic operations are pointwise operations on streams.

$$
\begin{aligned}
[\![\texttt{1 + 2}]\!] &= [1, 1, 1, \ldots \quad [\![\texttt{+}]\!] \quad [2, 2, 2, \ldots \\
&= [1 + 2, 1 + 2, 1 + 2, \ldots \\
&= [3, 3, 3, \ldots
\end{aligned}
$$

Why Shallow Embedding for Ebba? (2)

Two main types of embeddings:

Why Shallow Embedding for Ebba? (2)

Two main types of embeddings:

- Deep: Embedded language constructs translated into abstract syntax tree for subsequent interpretation or compilation.

Why Shallow Embedding for Ebba? (2)

Two main types of embeddings:

- Deep: Embedded language constructs translated into abstract syntax tree for subsequent interpretation or compilation.
  `1 + 2` interpreted as:
  ```
  Add (LitInt 1) (LitInt 2)
  ```

Why Shallow Embedding for Ebba? (2)

Two main types of embeddings:

- Deep: Embedded language constructs translated into abstract syntax tree for subsequent interpretation or compilation.
  `1 + 2` interpreted as:
  `Add (LitInt 1) (LitInt 2)`
- Shallow: Embedded language constructs translated directly into semantics in host language terms.

Why Shallow Embedding for Ebba? (2)

Two main types of embeddings:

- Deep: Embedded language constructs translated into abstract syntax tree for subsequent interpretation or compilation. `1 + 2` interpreted as:
  `Add (LitInt 1) (LitInt 2)`
- Shallow: Embedded language constructs translated directly into semantics in host language terms. `1 + 2` interpreted as:
  `zipWith (+) (repeat 1) (repeat 2)`

Why Shallow Embedding for Ebba? (3)

Shallow embedding:

Why Shallow Embedding for Ebba? (3)

Shallow embedding:

- More direct account of semantics: suitable for research into semantic aspects.

Why Shallow Embedding for Ebba? (3)

Shallow embedding:

- More direct account of semantics: suitable for research into semantic aspects.
- Easier to extend and change than deep embedding: suitable for research into language design.

Why Shallow Embedding for Ebba? (3)

Shallow embedding:

- More direct account of semantics: suitable for research into semantic aspects.
- Easier to extend and change than deep embedding: suitable for research into language design.

(Long term: for reasons of performance, maybe move to a mixed-level embedding.)

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?
> And how certain can we be?

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?
> And how certain can we be?

Example: Suppose a coin is flipped 10 times, and the result is only heads.

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?
> And how certain can we be?

Example: Suppose a coin is flipped 10 times, and the result is only heads.

- Is the coin fair (head and tail equally likely)?

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?
> And how certain can we be?

Example: Suppose a coin is flipped 10 times, and the result is only heads.

- Is the coin fair (head and tail equally likely)?
- Is it perhaps biased towards heads? How much?

Bayesian Data Analysis (1)

A common scenario across science, engineering, finance, . . . :

> Some observations have been made.
> What is/are the cause(s)?
> And how certain can we be?

Example: Suppose a coin is flipped 10 times, and the result is only heads.

- Is the coin fair (head and tail equally likely)?
- Is it perhaps biased towards heads? How much?
- Maybe it’s a coin with two heads?

Bayesian Data Analysis (2)

Bayes' theroem allows such questions to be answered systematically:

$$P(X \mid Y) = \frac{P(Y \mid X) \times P(X)}{P(Y)}$$

where

- $P(X)$ is the ***prior*** probability
- $P(Y \mid X)$ is the ***likelihood*** function
- $P(X \mid Y)$ is the ***posterior*** probability
- $P(Y)$ is the ***evidence***

Bayesian Data Analysis (3)

Assuming a probabilistic model for the observations ***parametrized*** to account for all possible causes

$$\mathrm{P}(data \mid params)$$

and any knowledge about the parameters, $\mathrm{P}(params)$, Bayes' theorem yields the probability for the ***parameters*** given the observations:

$$\mathrm{P}(params \mid data) = \frac{\mathrm{P}(data \mid params) \times \mathrm{P}(params)}{\mathrm{P}(data)}$$

Bayesian Data Analysis (3)

Assuming a probabilistic model for the observations ***parametrized*** to account for all possible causes

$$\mathrm{P}(\mathit{data} \mid \mathit{params})$$

and any knowledge about the parameters, $\mathrm{P}(\mathit{params})$, Bayes' theorem yields the probability for the ***parameters*** given the observations:

$$\mathrm{P}(\mathit{params} \mid \mathit{data}) = \frac{\mathrm{P}(\mathit{data} \mid \mathit{params}) \times \mathrm{P}(\mathit{params})}{\mathrm{P}(\mathit{data})}$$

I.e., ***exactly*** what can be inferred from the observations under the explicitly stated assumptions.

Thomas Bayes, 1702–1761

Thomas Bayes, 1702–1761

Fair Coin (1)

A probabilistic model for a single toss of a coin is that the probability of head is p (a Bernoulli distribution); p is our parameter.

If the coin is tossed n times, the probability for h heads for a given p is:

$$\mathrm{P}(h \,|\, p) = \binom{n}{h} p^h (1-p)^{n-h}$$

(a binomial distribution).

Fair Coin (2)

If we have no knowledge about p, except its range, we can assume a uniformly distributed prior:

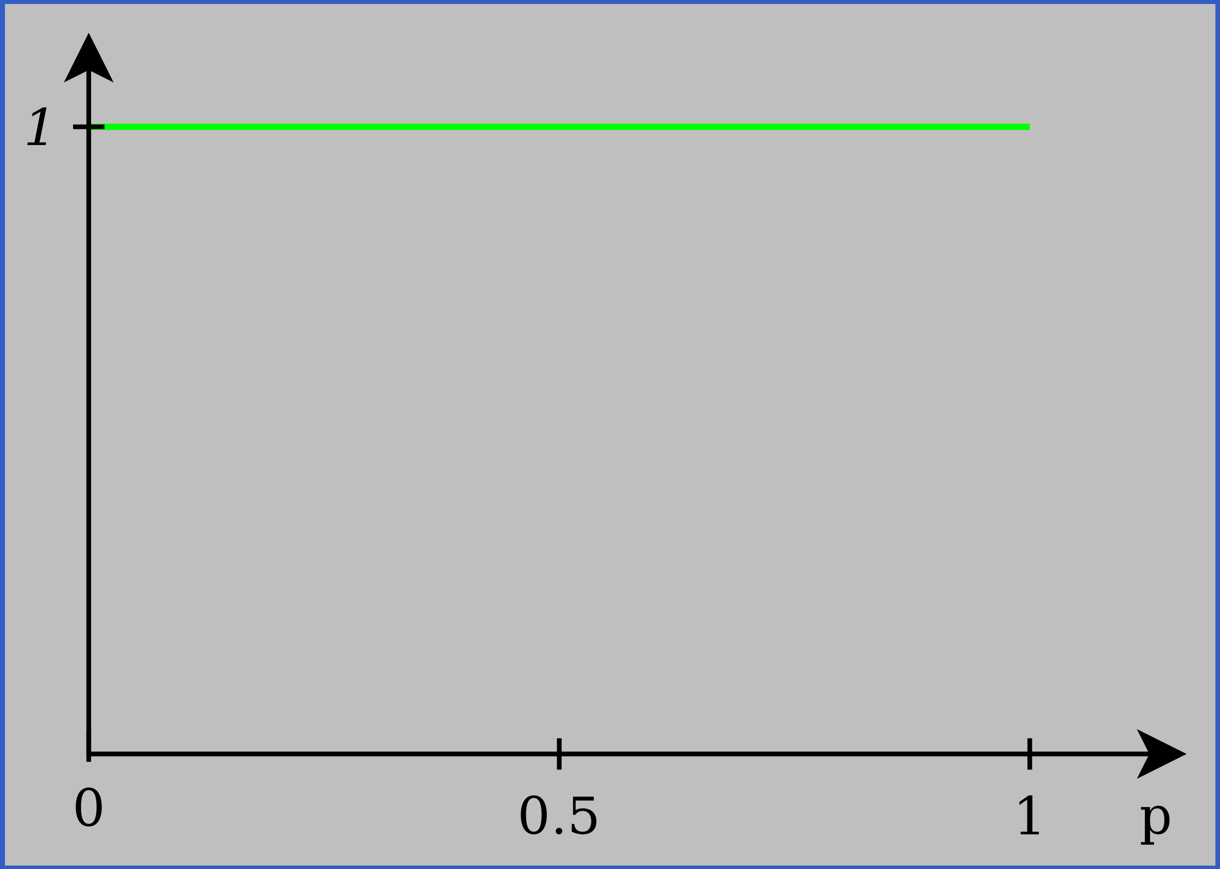
$$\mathrm{P}(p) = \begin{cases} 1 & \text{if } 0 \leq p \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Ignoring the evidence, which is just a normalization constant, we then have:

$$\mathrm{P}(p \,|\, h) \propto \mathrm{P}(h \,|\, p) \times \mathrm{P}(p)$$

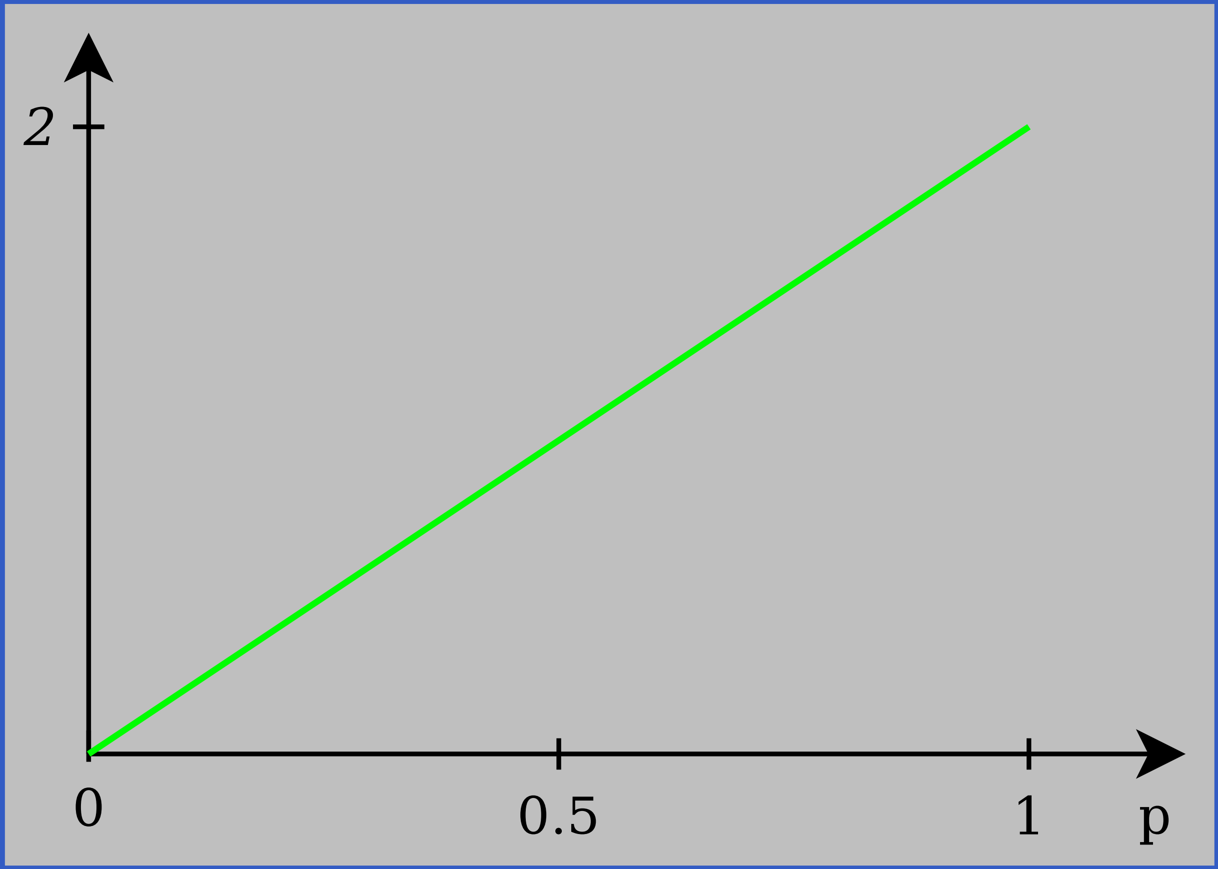
Fair Coin (3)

Distribution for p given no observations:



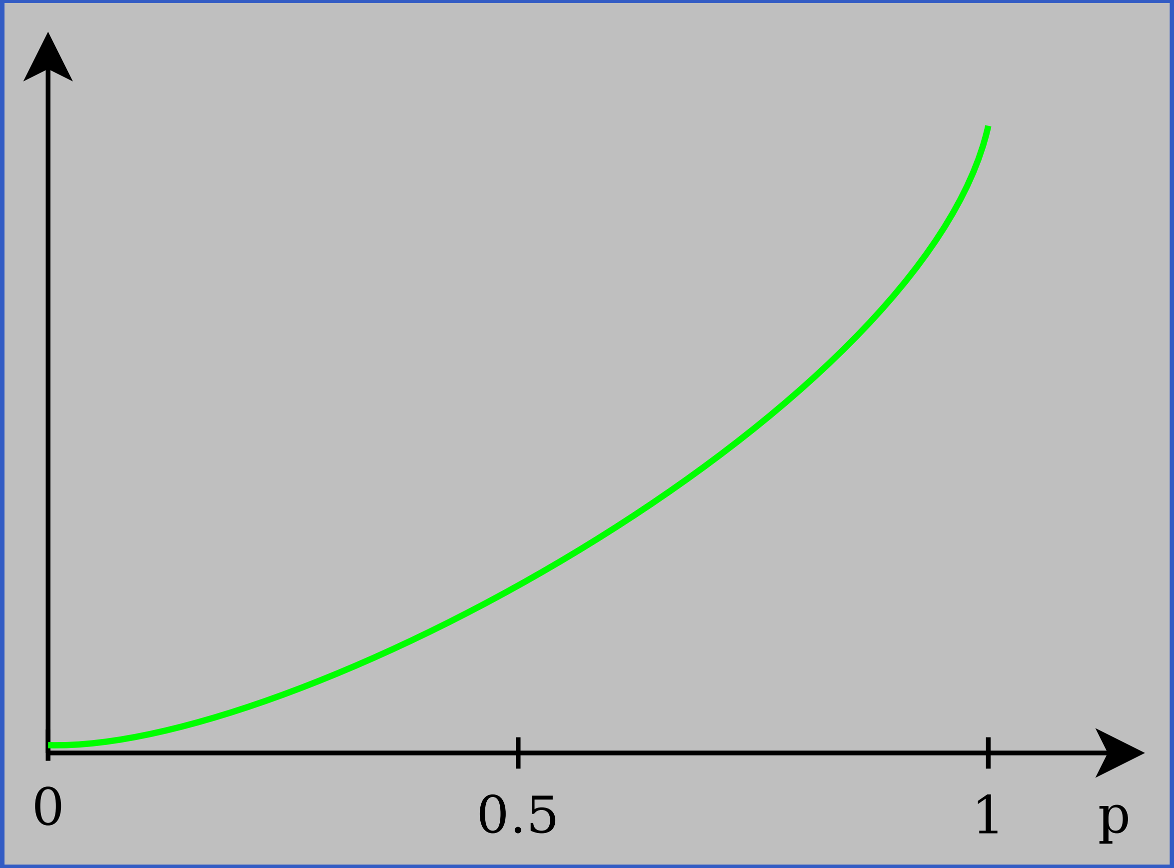
Fair Coin (4)

Distribution for p given 1 toss resulting in head:



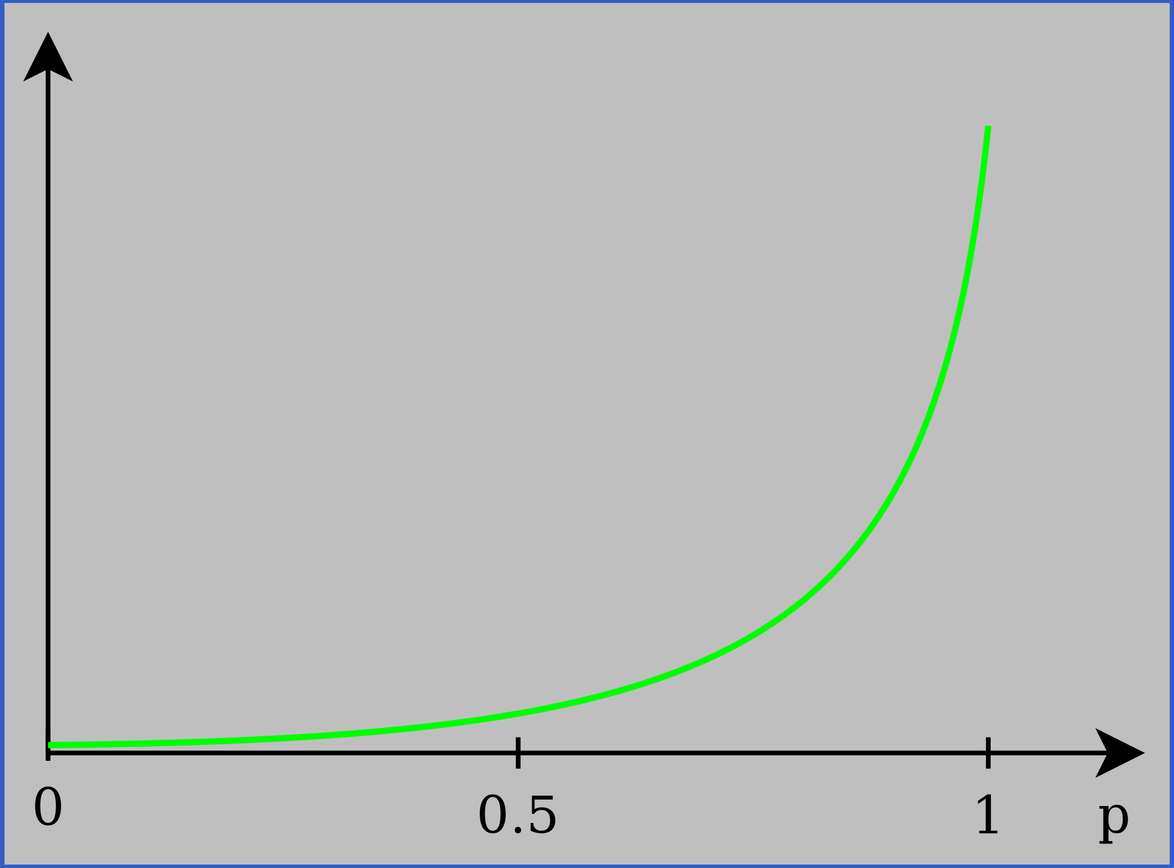
Fair Coin (5)

Distribution for p given 2 tosses resulting in 2 heads:



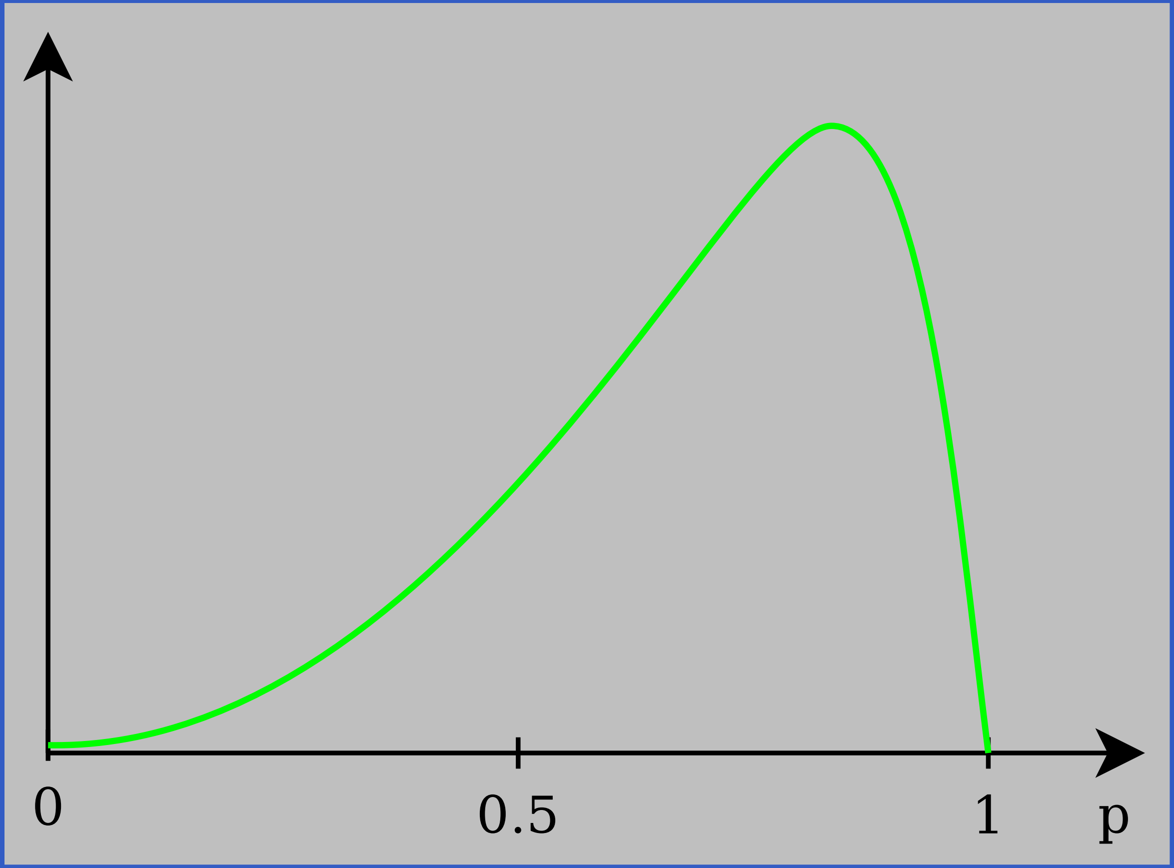
Fair Coin (6)

Distribution for p given many tosses, all heads:



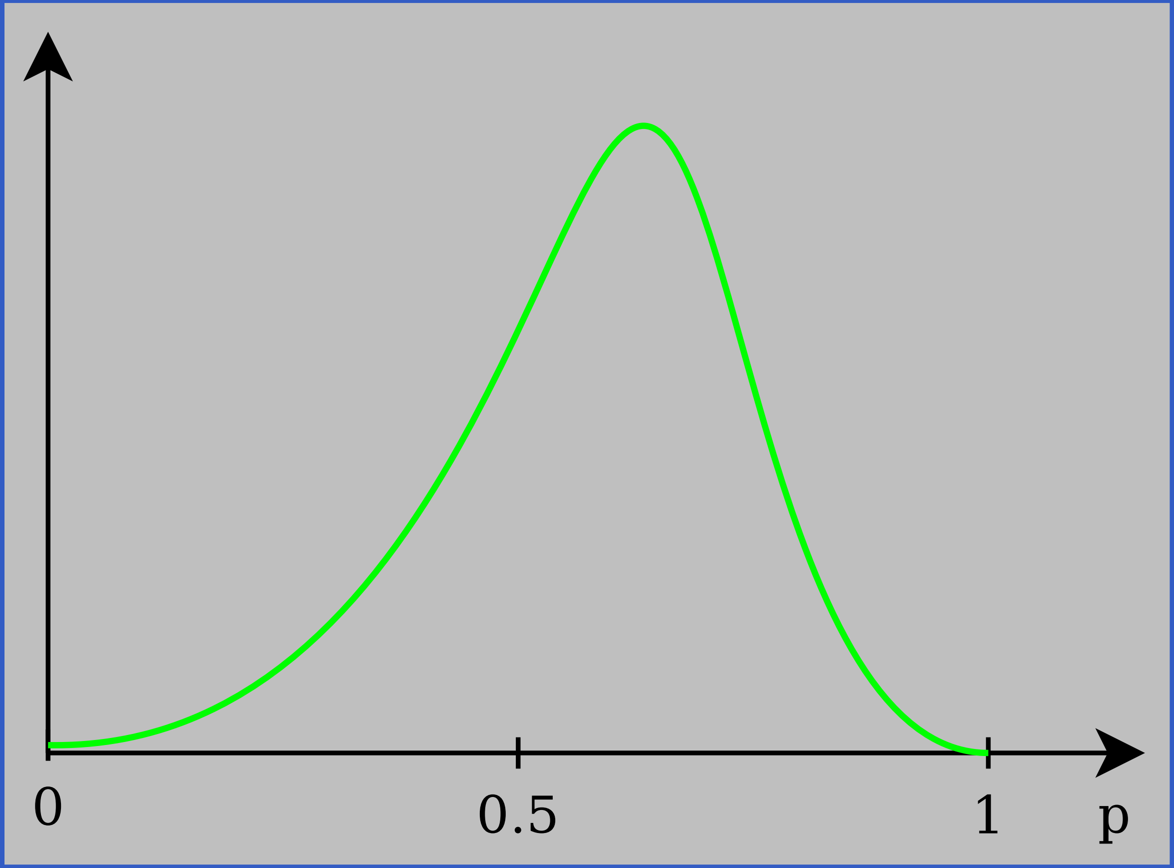
Fair Coin (7)

Distribution for p once finally a tail comes up:



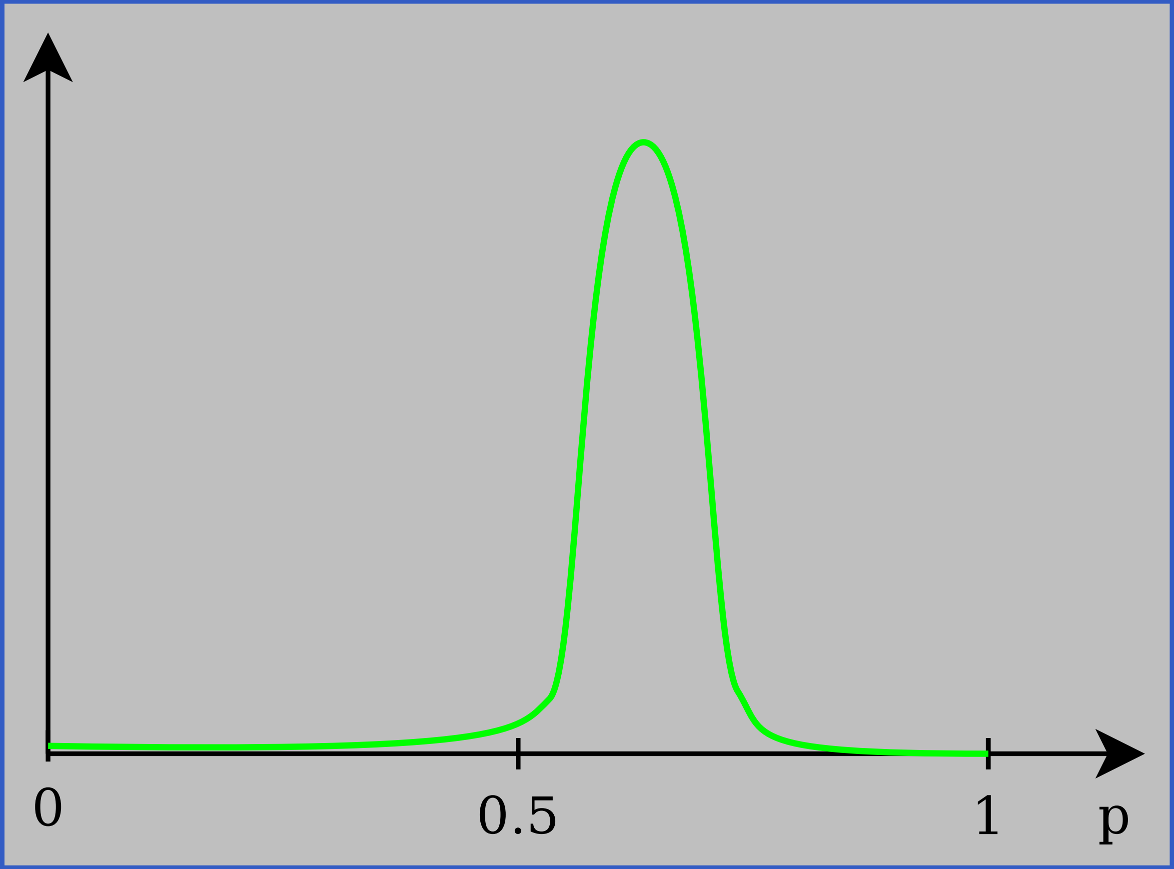
Fair Coin (8)

After a fair few tosses, observing heads and tails:



Fair Coin (9)

Distribution for p after even more tosses:



Fair Coin (10)

As the number of observations grow:

Fair Coin (10)

As the number of observations grow:

- the distribution for the parameter becomes increasingly sharp;

Fair Coin (10)

As the number of observations grow:

- the distribution for the parameter becomes increasingly sharp;
- the significance of the exact shape of the prior diminishes.

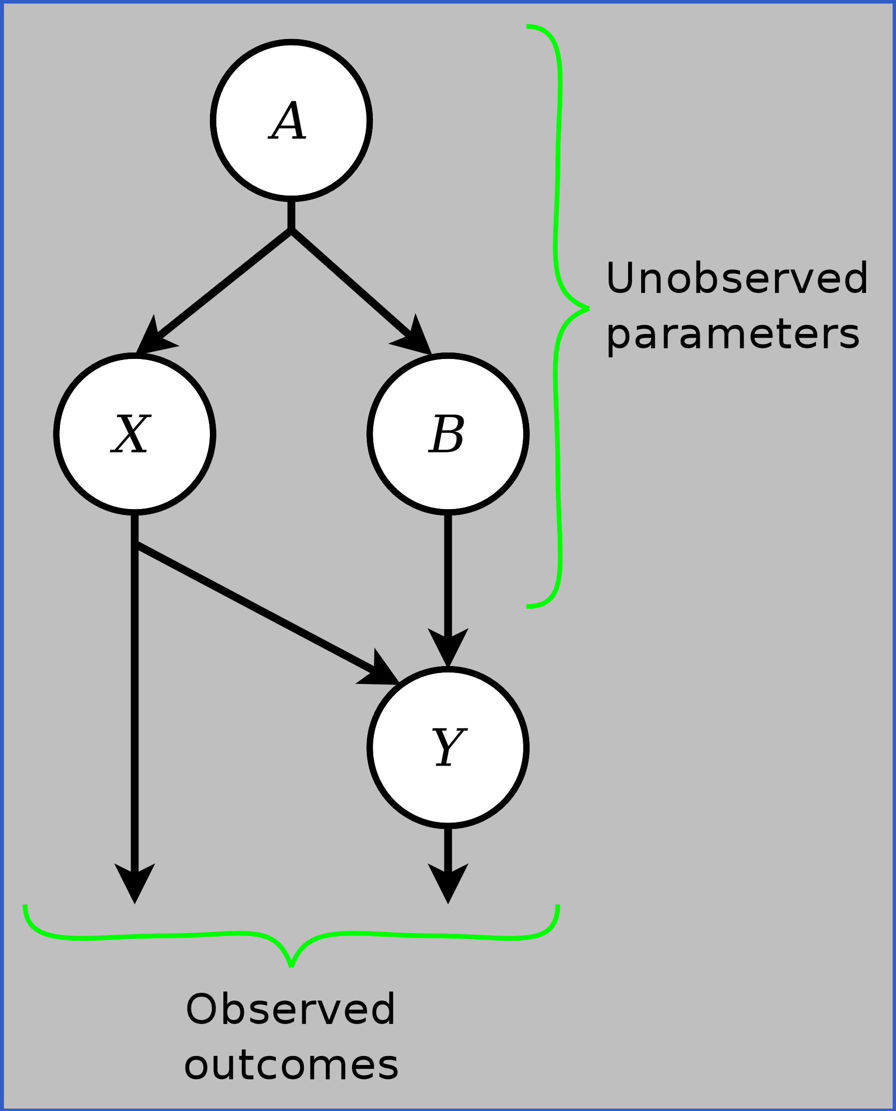
Fair Coin (10)

As the number of observations grow:

- the distribution for the parameter becomes increasingly sharp;
- the significance of the exact shape of the prior diminishes.

Thus, if we trust our model, Bayes' theorem tells us exactly what is justified to believe about the parameter(s) given the observations at hand.

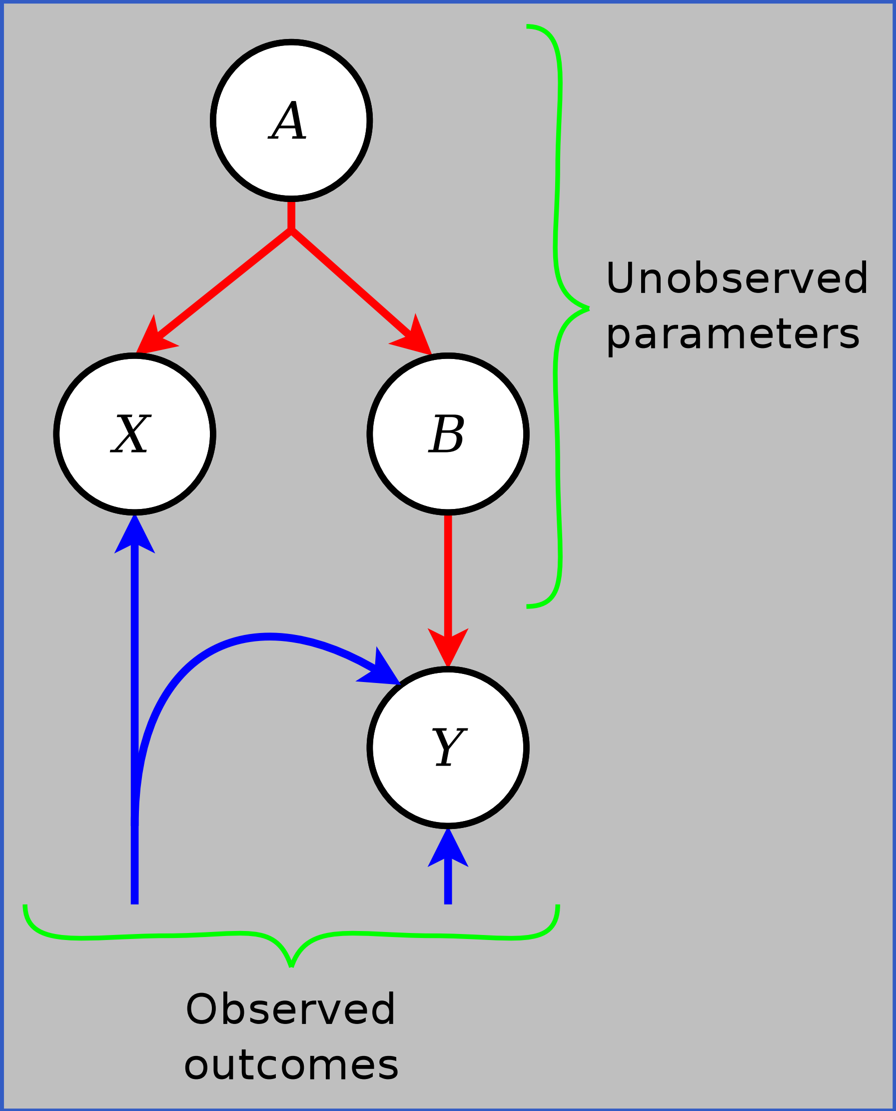
Probabilistic Models



In practice, there are often many parameters (dimensions) and intricate dependences.

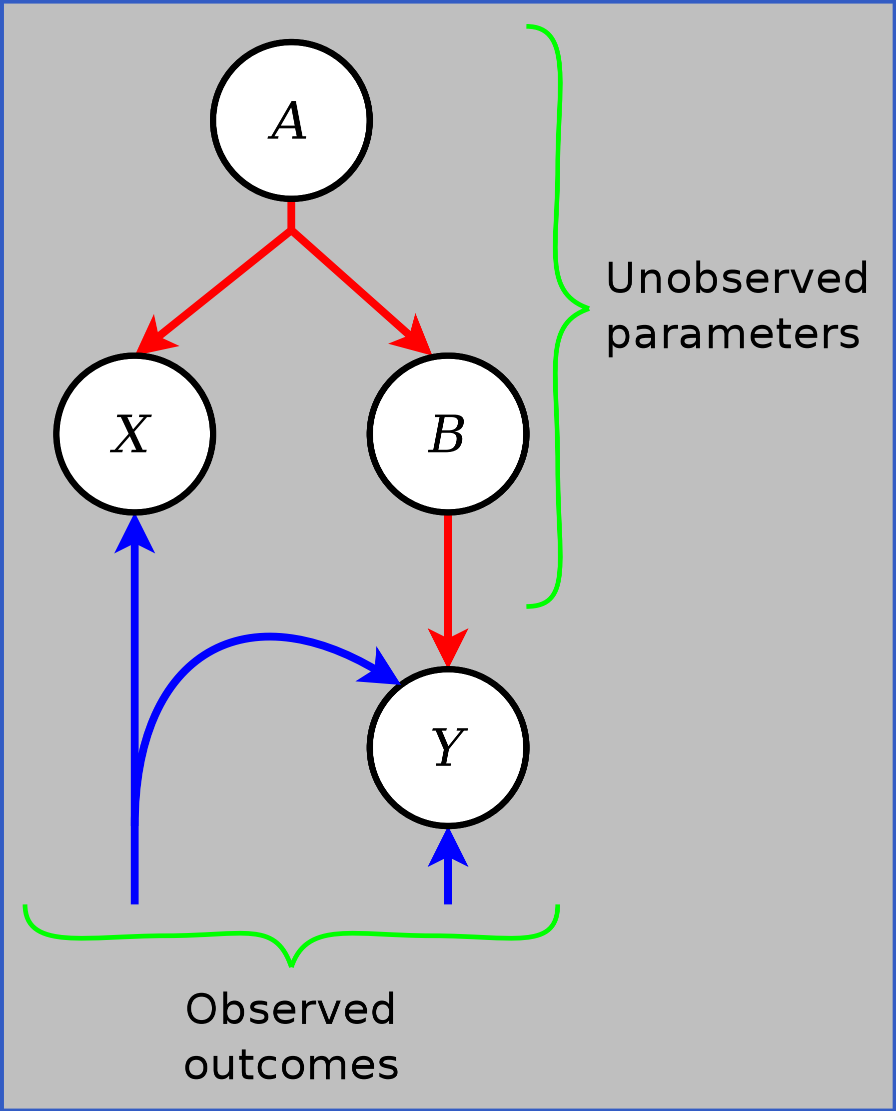
Here, the nodes are random variables with (conditional) probabilities $\mathrm{P}(A)$, $\mathrm{P}(B \mid A)$, $\mathrm{P}(X \mid A)$, $\mathrm{P}(Y \mid B, X)$.

Parameter Estimation (1)



According to Bayes' theorem, a function ***proportional*** to the sought ***probability density function*** $\mathrm{pdf}_{A,B|X,Y}$ is obtained by the "product" of the pdfs for the individual nodes applied to the observed data.

Parameter Estimation (2)



$$\mathrm{pdf}_A : T_A \to \mathbb{R}$$
$$\mathrm{pdf}_{B|A} : T_A \to T_B \to \mathbb{R}$$
$$\mathrm{pdf}_{X|A} : T_A \to T_X \to \mathbb{R}$$
$$\mathrm{pdf}_{Y|B,X} : (T_B, T_X) \to T_Y \to \mathbb{R}$$

Given observations x, y:

$$\begin{aligned}\mathrm{pdf}_{A,B|X,Y}\; a\; b \;\propto\; & \mathrm{pdf}_{Y|B,X}\,(b, \mathrm{x})\;\mathrm{y} \\ & \times \mathrm{pdf}_{X|A}\; a\; \mathrm{x} \\ & \times \mathrm{pdf}_{B|A}\; b\; a \\ & \times \mathrm{pdf}_A\; a\end{aligned}$$

Parameter Estimation (3)

Problem: We only get a function ***proportional*** to the desired pdf as the ***evidence*** in practice is very difficult to calculate.

Parameter Estimation (3)

Problem: We only get a function ***proportional*** to the desired pdf as the ***evidence*** in practice is very difficult to calculate.

However, MCMC (Markov Chain Monte Carlo) methods such as ***Metropolis-Hastings*** allow sampling of the desired distribution. That in turn allows the distribution for any of the parameters to be approximated.

Probabilistic Langauges and Estimation

It is straightforward to turn a general-purpose language into one in which probabilistic ***computations*** can be expressed:

- Imperative: Call a random number generator
- Pure functional: Use the probability monad:

$$
\begin{aligned}
&\mathit{coinFlips} :: \mathit{Int} \rightarrow \mathit{Prob}\,[\,\mathit{Bool}\,]\\
&\mathit{coinFlips}\ n = \mathbf{do}\\
&\quad p \qquad \leftarrow \mathit{uniform}\ 0\ 1\\
&\quad \mathit{flips} \leftarrow \mathit{replicateM}\ \ n\ (\mathit{bernoulli}\ \ p)\\
&\quad \mathit{return}\ \mathit{flips}
\end{aligned}
$$

Probabilistic Langauges and Estimation

However, for ***estimation***, the ***static*** unfolding of the structure of a computation must be a ***finite*** graph.

Probabilistic Langauges and Estimation

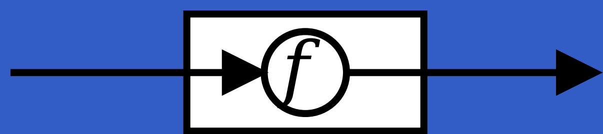
However, for ***estimation***, the ***static*** unfolding of the structure of a computation must be a ***finite*** graph.

But an imperative language/monad allows the rest of a computation to depend in arbitrary ways on result of earlier computation. E.g.:

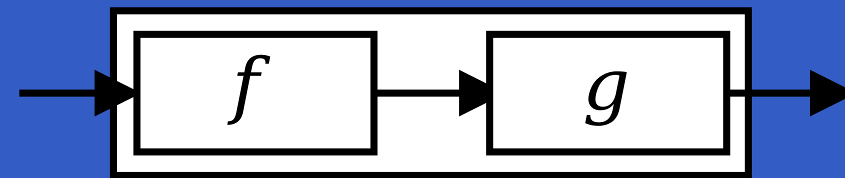
$$
\begin{array}{l}
\mathit{foo}\ n = \mathrm{do} \\
\quad x \leftarrow \mathit{uniform}\ 0\ 1 \\
\quad \mathrm{if}\ x < 0.5\ \mathrm{then} \\
\quad\quad \mathit{foo}\ (n+1) \\
\quad \mathrm{else} \ldots
\end{array}
$$

Probabilistic Languages and Estimation

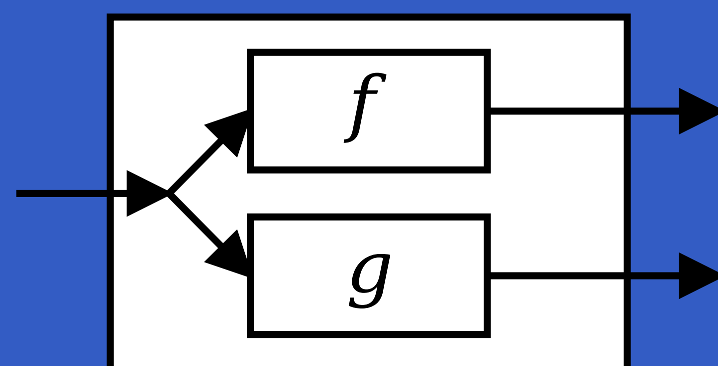
Maybe something like ***arrows*** would be a better fit?



$arr\ f$



$f \ggg g$



$f\ \&\&\&\ g$

Describes networks of interconnected "function-like" objects.

Probabilistic Languages and Estimation

- Arrows offer fine-grained control over available computational features (conditionals, feedback, . . .)

Probabilistic Languages and Estimation

- Arrows offer fine-grained control over available computational features (conditionals, feedback, . . .)
- ***Static structure*** of an arrow computation can be enforced

Probabilistic Languages and Estimation

- Arrows offer fine-grained control over available computational features (conditionals, feedback, . . .)
- ***Static structure*** of an arrow computation can be enforced
- Arrows make the dependences between computations manifest.

Probabilistic Languages and Estimation

- Arrows offer fine-grained control over available computational features (conditionals, feedback, . . .)
- ***Static structure*** of an arrow computation can be enforced
- Arrows make the dependences between computations manifest.
- Conditional probabilities, $a \to Prob\ b$ ***are*** an arrow through the Kleisli construction.

The Conditional Probability Arrow (1)

Central abstraction: $CP\ o\ a\ b$

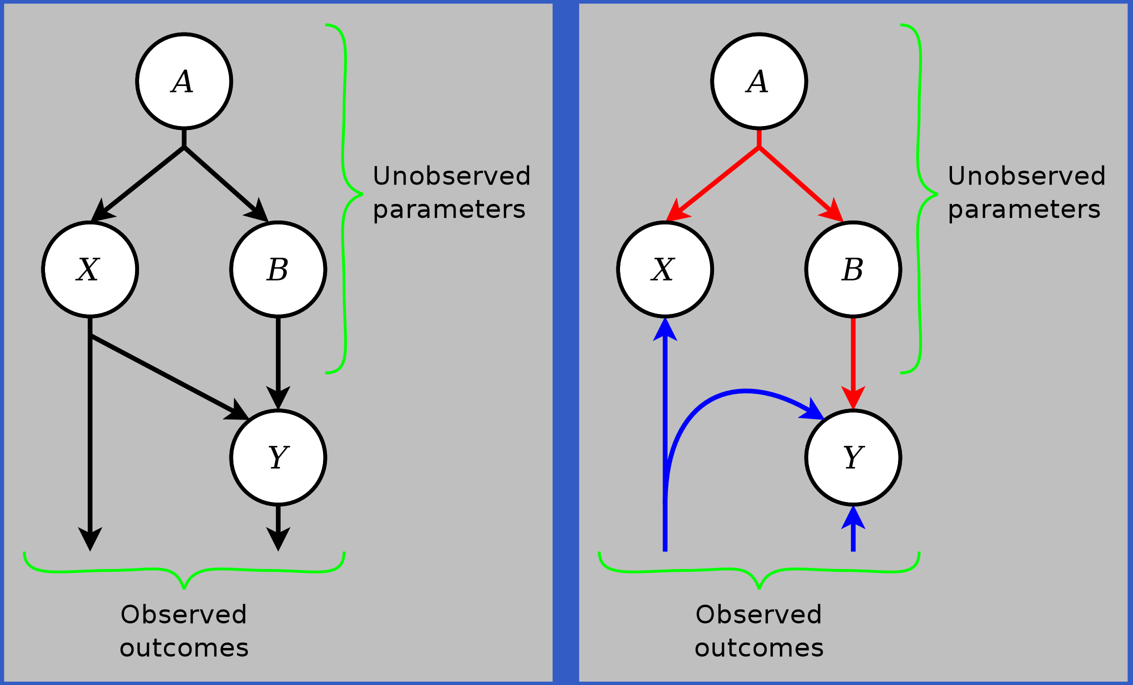
- a: The “given”
- b: The “outcome”
- o: Observability. Describes which parts of the given are observable from the outcome; i.e., for which there exists a pure function mapping (part of) the outcome to (part of) the given.

Observability does ***not*** mean “will be observed”.

The Conditional Probability Arrow (2)

Observability:

- Determined by type-level computation.
- Dictates how information flows in the network in “reverse mode”.



The Conditional Probability Arrow (2)

What kind of arrow?

The Conditional Probability Arrow (2)

What kind of arrow?

- Clearly not a classic arrow . . .

The Conditional Probability Arrow (2)

What kind of arrow?

- Clearly not a classic arrow . . .
- Probably a Constrained, Indexed, Generalized Arrow.

The Conditional Probability Arrow (2)

What kind of arrow?

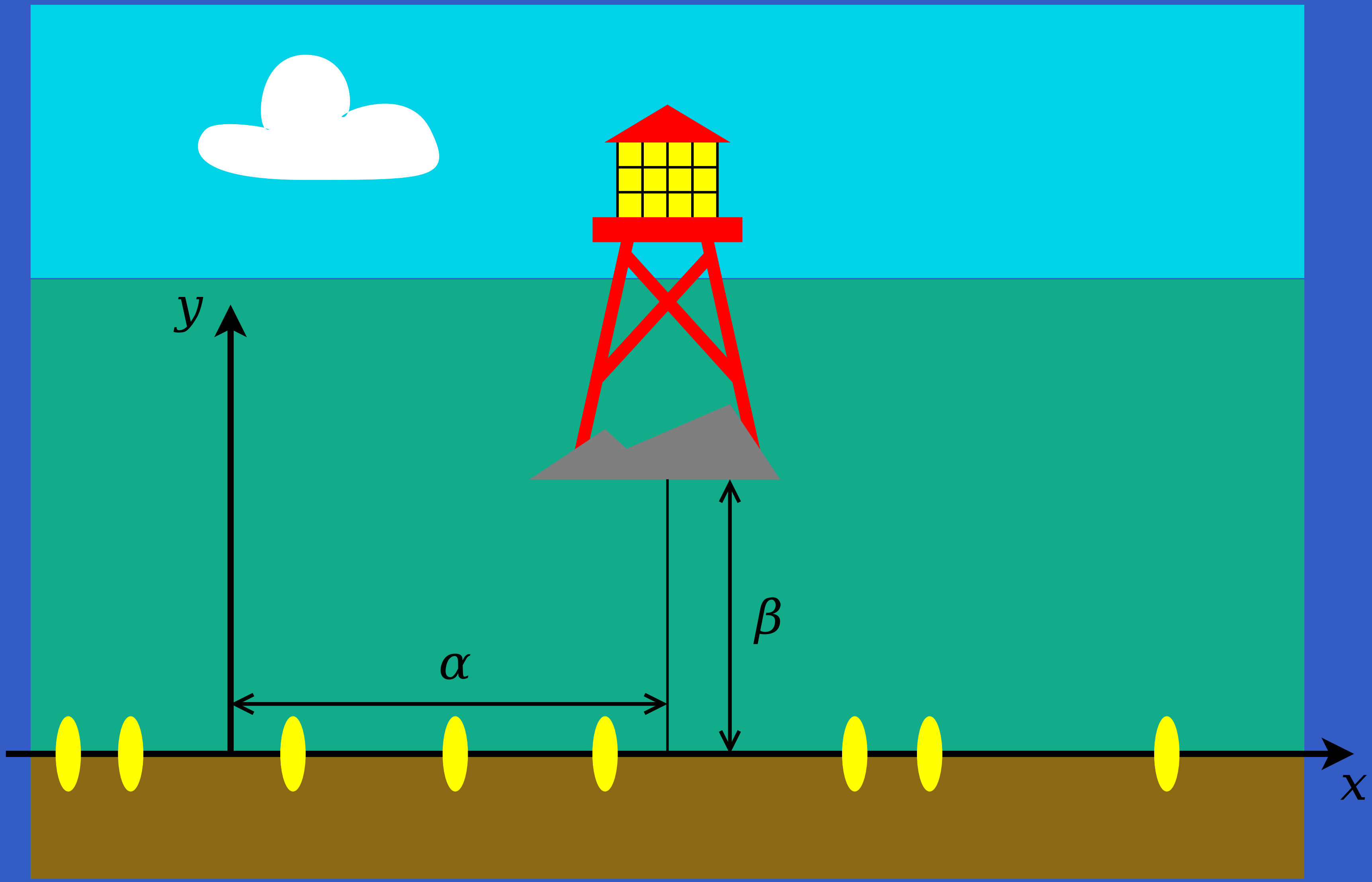
- Clearly not a classic arrow ...
- Probably a Constrained, Indexed, Generalized Arrow.

$$
\begin{aligned}
(\ast\!\ast\!\ast) &:: CP\ o1\ a\ b \rightarrow CP\ o2\ c\ d \rightarrow CP\ (o1 \ast\!\ast\!\ast\ o2)\ (a, c)\ (b, d) \\
(\ggg) &:: Fusable\ o2\ b \\
&\quad \Rightarrow CP\ o1\ a\ b \rightarrow CP\ o2\ b\ c \rightarrow CP\ (o1 \ggg o2)\ a\ c \\
(\ \&\&\&\ ) &:: Selectable\ o1\ o2\ a \\
&\quad \Rightarrow CP\ o1\ a\ b \rightarrow CP\ o2\ a\ c \rightarrow CP\ (o1\ \&\&\&\ o2)\ a\ (b, c)
\end{aligned}
$$

Implementation Sketch

$$
\begin{array}{l}
\textbf{type } \mathit{Parameters} = \mathit{Map}\ \mathit{Name}\ \mathit{ParVal} \\
\textbf{data } \mathit{CP}\ o\ a\ b = \mathit{CP}\ \{ \\
\quad \mathit{cp} \qquad\qquad :: a \rightarrow \mathit{Prob}\ b, \\
\quad \mathit{initEstim} :: a \rightarrow a \rightarrow b \\
\qquad\qquad\qquad \rightarrow \mathit{Prob}\ (b, a, \mathit{Double}, \mathit{Parameters}, E\ o\ a\ b) \\
\} \\
\textbf{data } E\ o\ a\ b = E\ \{ \\
\quad \mathit{estimate} :: \mathit{Bool} \rightarrow a \rightarrow a \rightarrow b \\
\qquad\qquad\qquad \rightarrow \mathit{Prob}\ (b, a, \mathit{Double}, \mathit{Parameters}, E\ o\ a\ b) \\
\}
\end{array}
$$

Example: The Lighthouse (1)



Example: The Lighthouse (2)

An analysis of the problem shows that the lighthouse flashes are Cauchy-distributed along the shore with pdf:

$$\mathrm{pdf}_{\mathrm{lhf}} = \frac{\beta}{\pi(\beta^2 + (x - \alpha)^2)}$$

Example: The Lighthouse (2)

An analysis of the problem shows that the lighthouse flashes are Cauchy-distributed along the shore with pdf:

$$\mathrm{pdf}_{\mathrm{lhf}} = \frac{\beta}{\pi(\beta^2 + (x - \alpha)^2)}$$

The mean and variance of a Cauchy distribution are undefined!

Example: The Lighthouse (2)

An analysis of the problem shows that the lighthouse flashes are Cauchy-distributed along the shore with pdf:

$$\mathrm{pdf}_{\mathrm{lhf}} = \frac{\beta}{\pi(\beta^2 + (x - \alpha)^2)}$$

The mean and variance of a Cauchy distribution are undefined!

Thus, even if we're only interested in α, attempting to estimate it by simple sample averaging is futile.

Example: The Lighthouse (3)

The main part of the Ebba lighthouse model:

$$
\begin{array}{l}
\mathit{lightHouse} :: \mathit{CP}\ \ \mathit{U}\ \ ()\ [\mathit{Double}] \\
\mathit{lightHouse} = \mathbf{proc}\ ()\ \mathbf{do} \\
\quad \alpha \leftarrow \mathit{uniformParam}\ \texttt{"alpha"}\ (-50)\ 50 \prec ()\\
\quad \beta\ \ \leftarrow \mathit{uniformParam}\ \texttt{"beta"}\ 0\ 20 \prec ()\\
\quad \mathit{xs} \leftarrow \mathit{many}\ 10\ \mathit{lightHouseFlash} \prec (\alpha, \beta)\\
\quad \mathit{returnA} \prec \mathit{xs}
\end{array}
$$

Note:

- Arrow-syntax used for clarity: not supported yet.
- Ebba needs refactoring to support data and parameters with arbitrary distributions.

Example: The Lighthouse (4)

Actual code right now:

$$
\begin{aligned}
&\mathit{lightHouse} :: \mathit{CP}\ \ U\ \ ()\ [\mathit{Double}]\\
&\mathit{lightHouse} = (\mathit{uniformParam}\ \texttt{"alpha"}\ (-50)\ 50\\
&\qquad\qquad\qquad\quad \&\&\&\ \mathit{uniformParam}\ \texttt{"beta"}\ 0\ 20)\\
&\qquad\qquad\qquad\ \ggg\ \mathit{many}\ 10\ \mathit{lightHouseFlashes}
\end{aligned}
$$

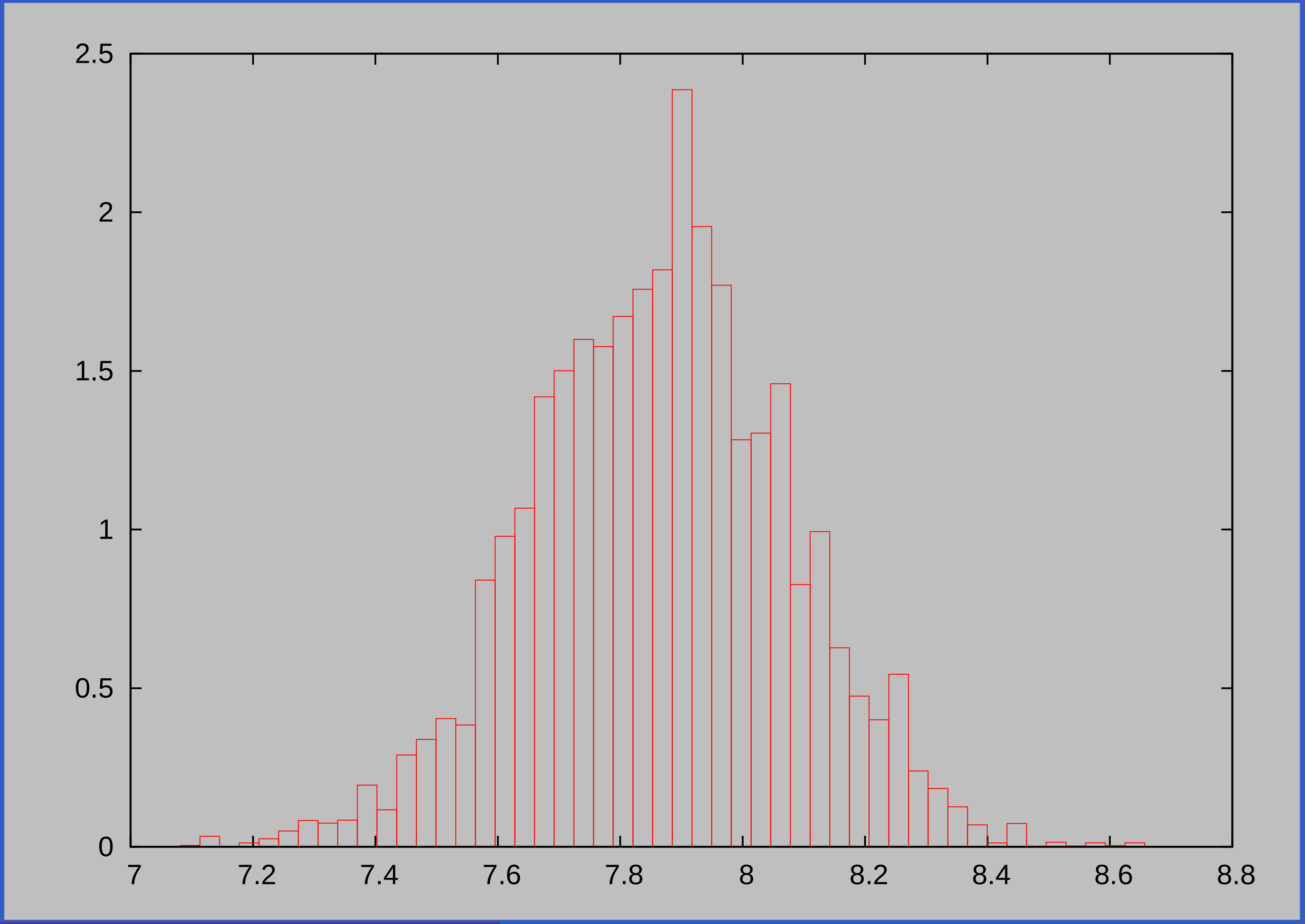
Example: The Lighthouse (5)

To test:

- A vector of 200 detected flashes was generated at random from the model for $\alpha = 8$ and $\beta = 2$. (the “ground truth”).
- The parameter distribution given the outcome sampled 100000 times using Metropolis-Hastings (picking every 10th sample from the Markov chain to reduce correlation between samples).

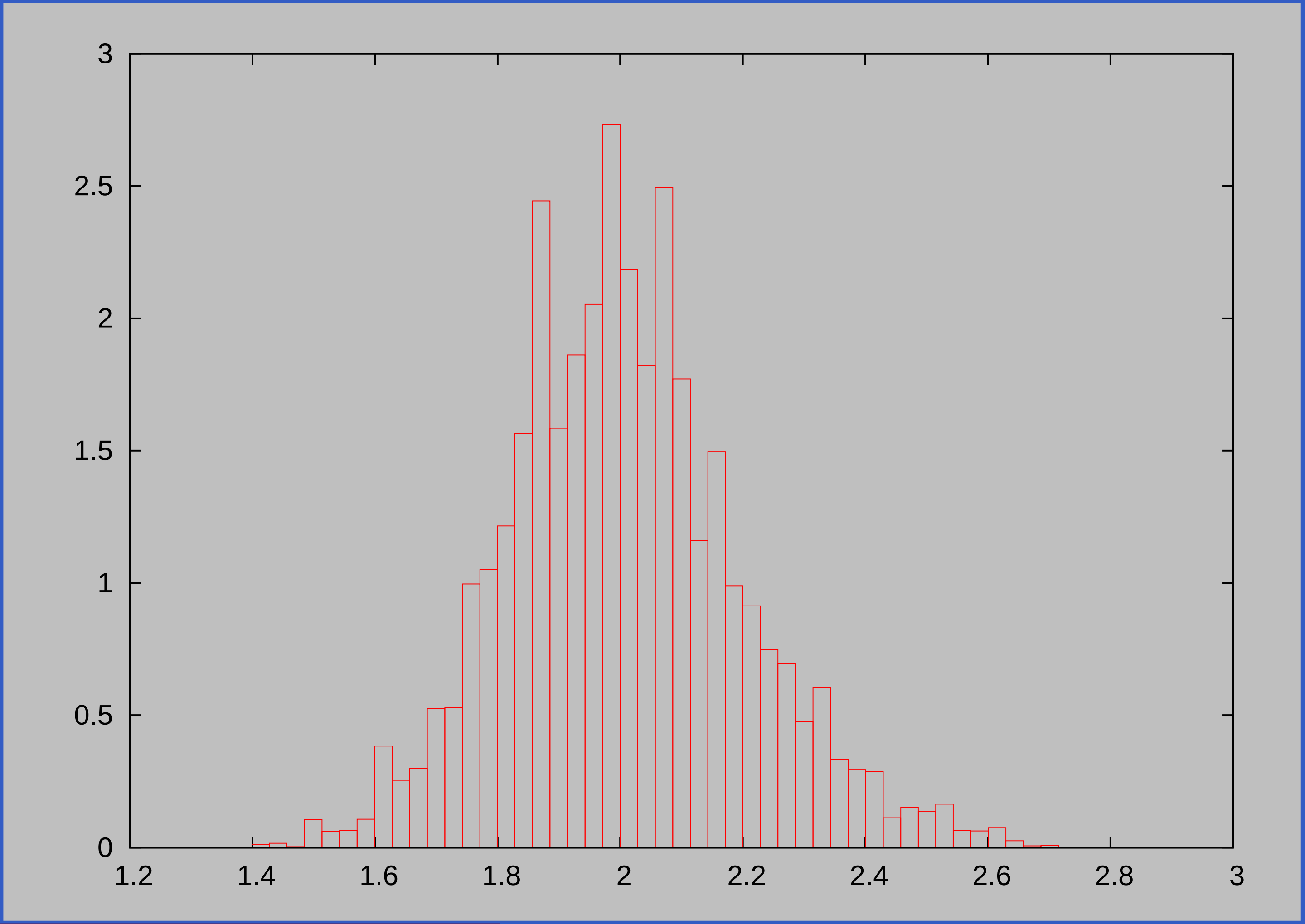
Example: The Lighthouse (6)

Resulting distribution for α:



Example: The Lighthouse (7)

Resulting distribution for β:



What's Next? (1)

- Testing on larger examples, including “hierarchical” models (nested use of $many$).
- Refactoring and the design, in particular:
  - General $\mathbf{data}$ and $parameter$ combinators parametrised on the distributions.
  - Framework for programming with Constrained, Indexed, Generalised Arrows:
    - Type classes $CIGArrow1$, $CIGArrow2$
    - Syntactic support through preprocessor implemented using QuasiQuoting?

What's Next? (2)

- More robust implementation of Metropolis Hastings
- Move towards a deep embedding for estimation?

  Idea: route a variable ***representation*** (name) through the network in place of parameter estimates.
- Support for gradient-based methods thorugh automatic differentiation using similar approach?