Ebba: An Embedded DSL for Bayesian Inference

Chalmers, 28 May 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 computation developed by OpenBrain Ltd:

  www.bayeshive.com

- Baysig supports parameter estimation; i.e., programs can in a sense be run both "forwards" and "backwards".
- This talk investigates the possibility of implementing a Baysig-like language as a (shallow) embedding in Haskell.

Baysig and Ebba (2)

- 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 an embedded version?
  - Ease of experimentation
  - Metaprogramming
  - Ease of use as component
  - Investigation into appropriate notion of computation supporting both probabilistic computation and parameter estimation.

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:

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

where

- $\mathrm{P}(X)$ is the ***prior*** probability
- $\mathrm{P}(Y \mid X)$ is the ***likelihood*** function
- $\mathrm{P}(X \mid Y)$ is the ***posterior*** probability
- $\mathrm{P}(Y)$ 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:

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

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

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:

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

(a binomial distribution).

Fair Coin (2)

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

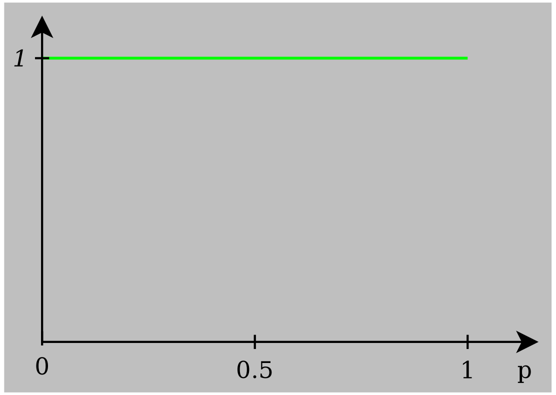
$$\mathrm{P}(p) = \begin{cases} 1 & \text{if } 0 \leq p \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

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

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

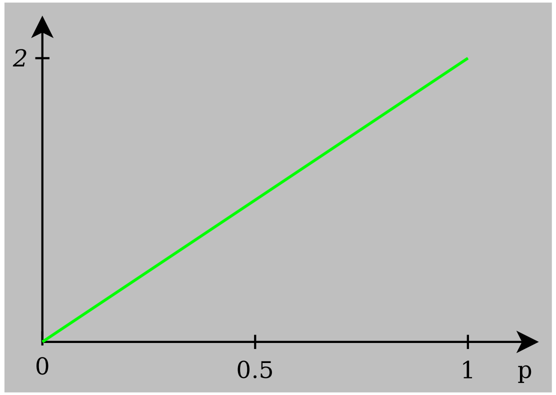
Fair Coin (3)

Distribution for p given no observations:



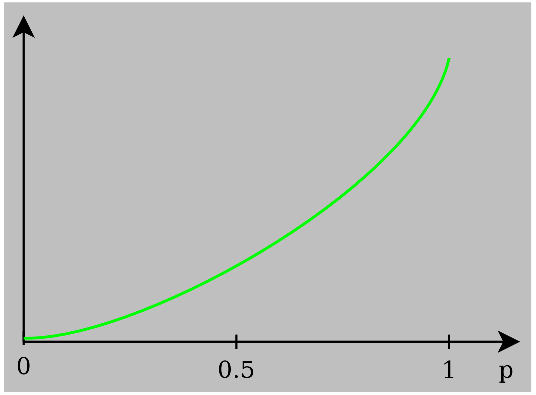
Fair Coin (4)

Distribution for p given 1 toss resulting in head:



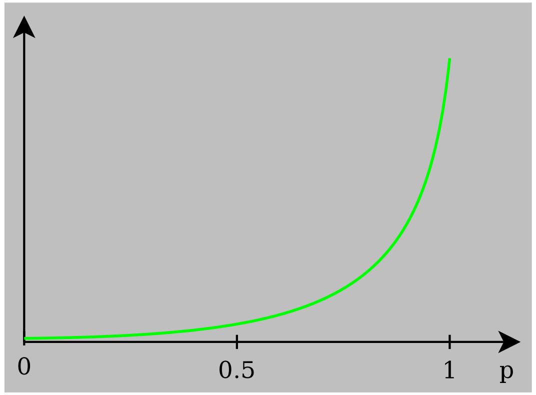
Fair Coin (5)

Distribution for p given 2 tosses resulting in 2 heads:



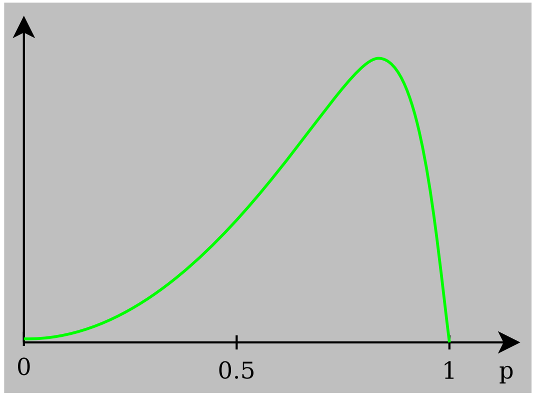
Fair Coin (6)

Distribution for p given many tosses, all heads:



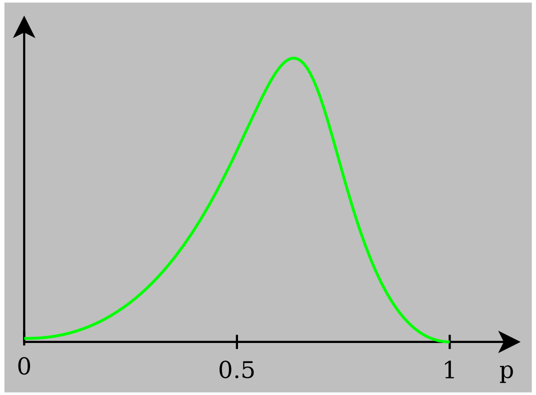
Fair Coin (7)

Distribution for p once finally a tail comes up:



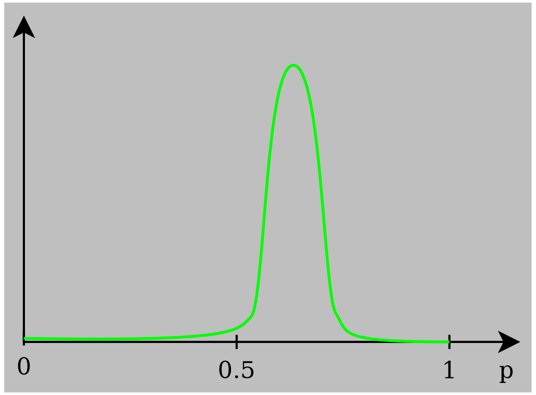
Fair Coin (8)

After a fair few tosses, observing heads and tails:



Fair Coin (9)

Distribution for p after even more tosses:



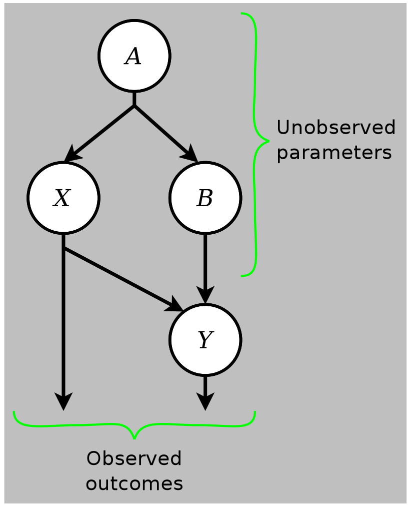
Fair Coin (10)

As the number of observations grow:

- 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.

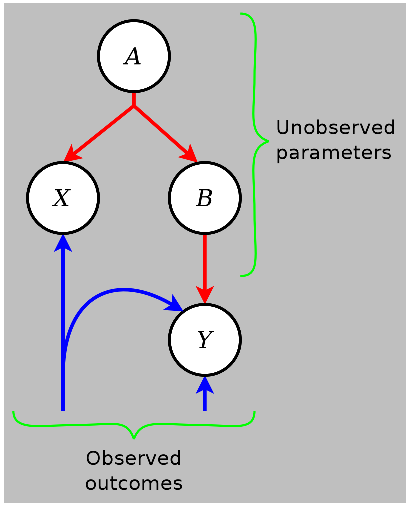
Probabilistic Models



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

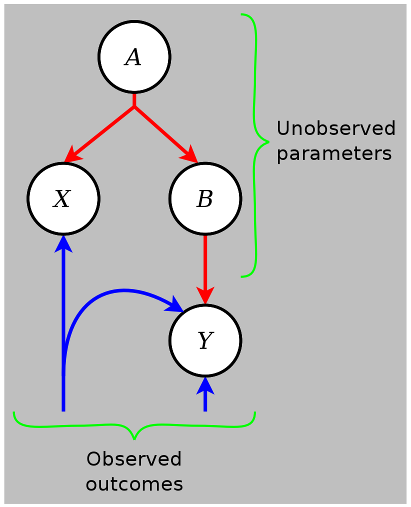
Here, the nodes are random variables with (conditional) probabilities $\mathrm{P}(A)$, $\mathrm{P}(B \mid A)$, $\mathrm{P}(X \mid A)$, $\mathrm{P}(Y \mid B, X)$.

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)



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

Given observations x, y:

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

Parameter Estimation (3)

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

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.

Metropolis-Hastings

Let $\bar{p}$ be the parameter vector and $f(\bar{p})$ be the function proprtional to the pdf of the distribution.

1. Pick a start state $\bar{p}$ at random.
2. Generate a new candidate state $\bar{p}'$ by perturbing the current state $\bar{p}$ a little.
3. If $f(\bar{p}') \geq f(\bar{p})$, keep $\bar{p}'$ and make it the new current state.
4. Otherwise keep $\bar{p}'$ probabilistically, with lower likelihood the worse $\bar{p}'$ is compared with $\bar{p}$.
5. Repeat from 2.

Probabilistic Langauges and Estimation

It is straightforward to turn a general-purpose language into one in which probabilistic models can be expressed. In a pure functional setting, we can use the probability monad:

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

Probabilistic Langauges and Estimation

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

This suggests that a monad is a ***too general*** notion of computation for probabilistic models on which we wish to perform estimation, at least in combination with general recursion, as allows computations to be computed ***dynamically***.

Probabilistic Languages and Estimation

Maybe something like arrows would be a better fit?

- The structure of an arrow computation ***is static*** (unless arrow application/similar is available)
- Arrows makes the dependences between computations manifest.
- Conditional probabilities, $a \rightarrow \mathit{Prob}\ b$ ***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.

Note: "Local", modular, composable notion.
Does ***not*** mean "will be observed".

The Conditional Probability Arrow (2)

What kind of arrow?

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

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

Observability (1)

Observability is described by (nested) tuples of:

- $\mathbf{data}\ U$: Unobservable
- $\mathbf{data}\ O\ (p :: [Nat])$: Observable from position p.

E.g. $(U, O\ [1, 2])$ means that fst of the given is unobservable, while snd can be observed from $snd \circ fst$ of the outcome.

Observability (2)

Type functions are used to compute observability of the arrow combinators. E.g. recall

$$
\begin{aligned}
(\ggg) &:: Fusable\ o2\ b \\
&\quad \Rightarrow CP\ o1\ a\ b \to CP\ o2\ b\ c \to CP\ (o1 \ggg o2)\ a\ c
\end{aligned}
$$

The type function $(\ggg)$ is defined as:

$$
\begin{aligned}
&\mathbf{type\ family}\ o1 \ggg o2 \\
&\mathbf{type\ instance}\ U \quad\quad\ \ggg o = U \\
&\mathbf{type\ instance}\ (O\ p) \quad \ggg o = Prj\ p\ o \\
&\mathbf{type\ instance}\ (o1, o2) \ggg o = (o1 \ggg o, o2 \ggg o)
\end{aligned}
$$

Implementation (1)

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

Implementation (2)

Arguments to $\mathit{initEstim}$ and $\mathit{estimate}$:

- Keep ($\mathit{estimate}$ only)
- Estimate of the given
- Fused estimate and observation of given (for computation of summand of the logarithm of the overall pdf for the parameters given the data).
- Observation of the outcome

Implementation (3)

Result from $\mathit{initEstim}$ and $\mathit{estimate}$:

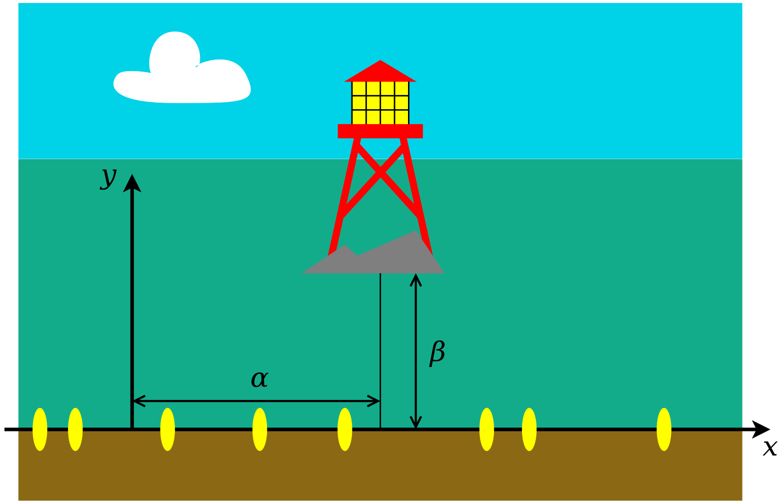
- Estimate of the outcome
- Observation of the given
- Summand of logarithm of overall pdf
- Parameters (estimated or observed)
- Continuation (Yampa-style)

Implementation (4)

Implementation of the estimator of $(\ggg)$:

$$
\begin{aligned}
&e1 \ggg e2 = E\ \$\ \lambda k\ x\_e\ x\_f\ z\_o \to \textbf{do}\\
&\quad fp \leftarrow \mathit{mfix}\ \$\ \lambda{\sim}(\_, y\_f') \to \textbf{do}\\
&\qquad (y\_e, x\_o, \mathit{lpds1}, \mathit{ps1}, e1') \leftarrow \mathit{estimate}\ e1\ k\ x\_e\ x\_f\ y\_f'\\
&\qquad (z\_e, y\_o, \mathit{lpds2}, \mathit{ps2}, e2') \leftarrow \mathit{estimate}\ e2\ k\ y\_e\ y\_f'\ z\_o\\
&\qquad \textbf{let}\ y\_f = \mathit{fuse}\ (\mathit{obs}\ e2)\ y\_e\ y\_o\\
&\qquad \mathit{return}\ ((z\_e, x\_o, \mathit{lpds1} + \mathit{lpds2}, M.\mathit{union}\ \mathit{ps1}\ \mathit{ps2},\\
&\qquad\qquad\qquad e1' \ggg e2'),\\
&\qquad\qquad\quad y\_f)\\
&\quad \mathit{return}\ (\mathit{fst}\ fp)
\end{aligned}
$$

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:

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

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:

```
lightHouse :: CP U () [Double]
lightHouse = proc () do
  α ← uniformParam "alpha" (−50) 50 ≺ ()
  β ← uniformParam "beta" 0 20 ≺ ()
  xs ← many 10 lightHouseFlash ≺ (α, β)
  returnA ≺ xs
```

Note:

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

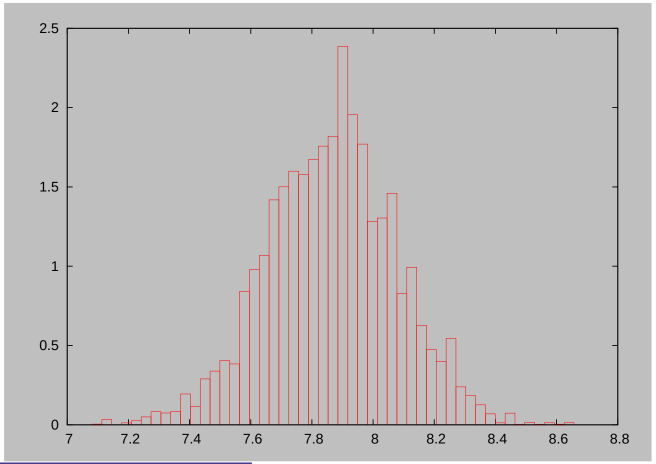
Example: The Lighthouse (4)

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).

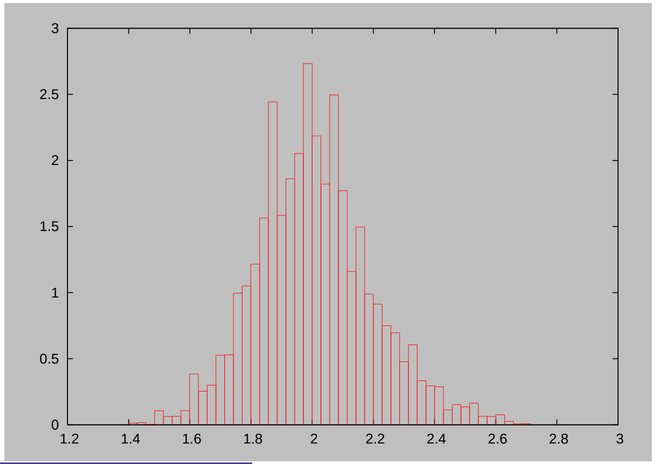
Example: The Lighthouse (5)

Resulting distribution for α:



Example: The Lighthouse (6)

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 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)

Probably something like:

$$
\begin{array}{l}
\mathbf{class}\ (CIGArrow1\ \ a1, CIGArrow1\ \ a2) \\
\qquad \Rightarrow CIGArrow2\ \ a1\ \ a2\ \mathbf{where} \\
\quad \mathbf{type}\ CompT\ \ a1\ \ a2 :: * \to * \to * \\
\quad \mathbf{type}\ CompC\ \ a1\ \ a2\ \ b\ \ c\ \ d :: Constraint \\
\quad \mathbf{type}\ CompC\ \ a1\ \ a2\ \ b\ \ c\ \ d = () \\
\quad (\ggg) :: CompC\ \ a1\ \ a2\ \ b\ \ c\ \ d \\
\qquad\qquad \Rightarrow a1\ \ b\ \ c \to a2\ \ c\ \ d \\
\qquad\qquad\quad \to (CompT\ \ a1\ \ a2)\ \ b\ \ d \\
\quad \ldots
\end{array}
$$

What's Next? (3)

- 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.