Letting Turing handle the BUGS

Where we are,
and where we want to go

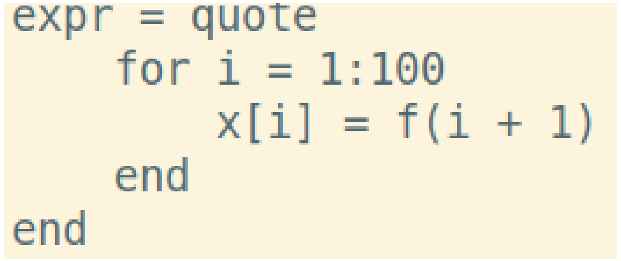
Current state

- Turing.jl: *Evaluator function* with *contexts*
  ...no graphical information!
- GraphPPL (in progress): hand-build graphical models
  ...no front-end; semantics & interface unclear
- Some half-baked ideas on a type system
  ...me inventing semantics from nothing

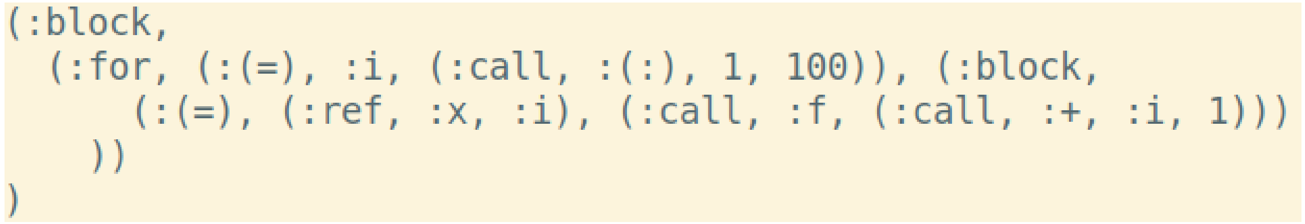
End goal

- Graphical representation for *models*
  → Manipulation, analysis & inference
- Connected to a front-end (DSL) with specified *semantics*
- Eventually: subsume both styles under common *abstraction*

Julia AST



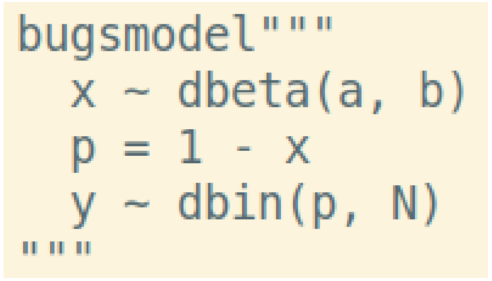
```
expr = quote
    for i = 1:100
        x[i] = f(i + 1)
    end
end
```



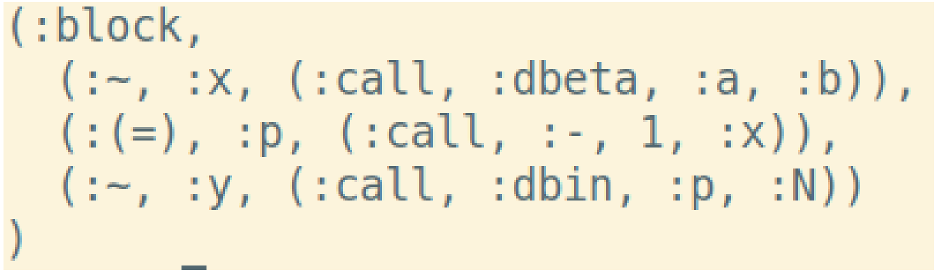
```
(:block,
  (:for, (:(=), :i, (:call, :(:), 1, 100)), (:block,
      (:(=), (:ref, :x, :i), (:call, :f, (:call, :+, :i, 1)))
    ))
)
```

- LISP-like: everything an S-expression
- Extensible and transformable (macros)
- Pre-semantic: no types et al.

BUGS fragment of Julia syntax



```
bugsmodel"""
  x ~ dbeta(a, b)
  p = 1 - x
  y ~ dbin(p, N)
"""
```



```
(:block,
  (:~, :x, (:call, :dbeta, :a, :b)),
  (:(=), :p, (:call, :-, 1, :x)),
  (:~, :y, (:call, :dbin, :p, :N))
)
```

- “Raw”, implicit graph
- Still ordered, unnormalized
- No metadata attached

Where to go

- Transformation to evaluator functions: easy, but uninteresting/insufficient
- Want: graphical model, node (meta)data, dependencies, …
- Correctness: validity, typing, domain constraints, ...

What’s the BUGS way?

- How do graph construction and checking phases interact?
- Role of the data?
- Specified semantics?
- “Lowering”: normalization, treatment of loops, ...

Concrete discussion points

- Phases of model construction/checking
- Usage of data structures
- Relation between “raw model” and data
- Semantics of “raw model” and “instantiated model”