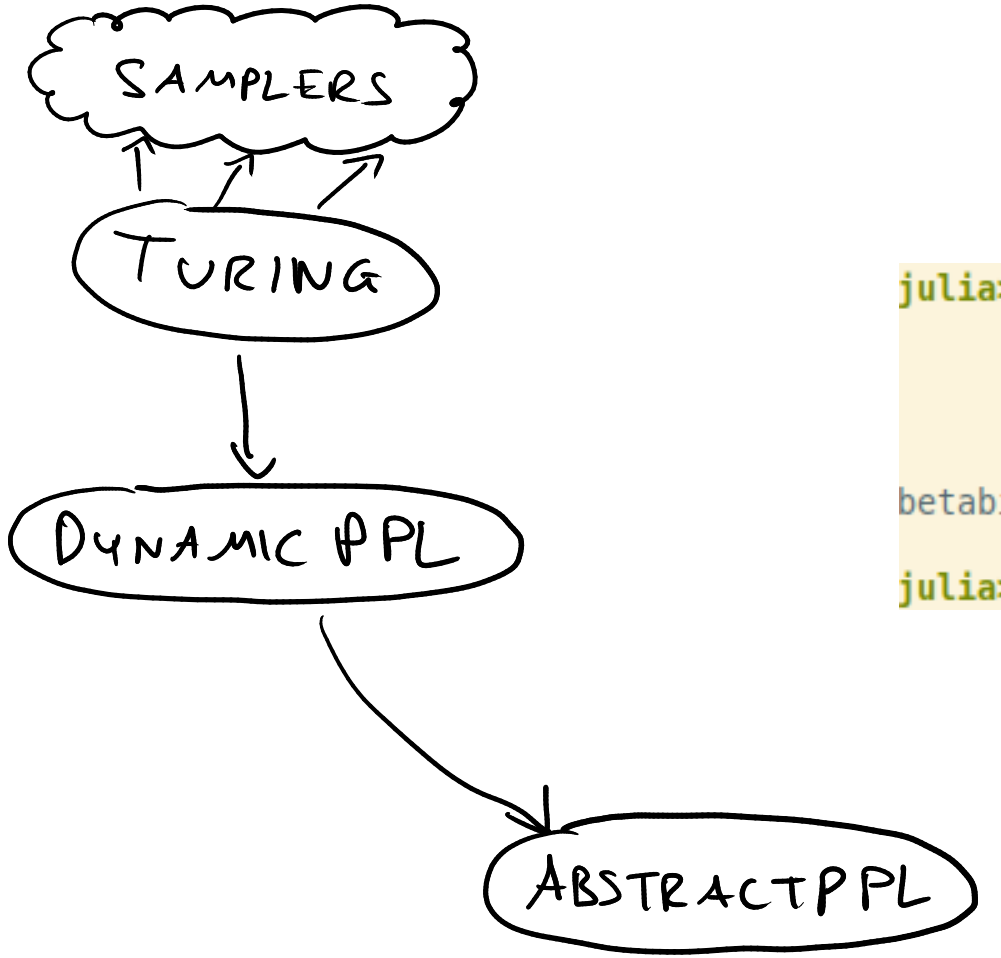
TURING IN CONTEXT




```
julia> @model function betabinomial(y, N, a = 2, b = 2)
           x ~ Beta(a, b)
           p = 1 - x
           y ~ Binomial(N, p)
       end
betabinomial (generic function with 2 methods)

julia> m = betabinomial([1,3,3,2,3], 5)
```

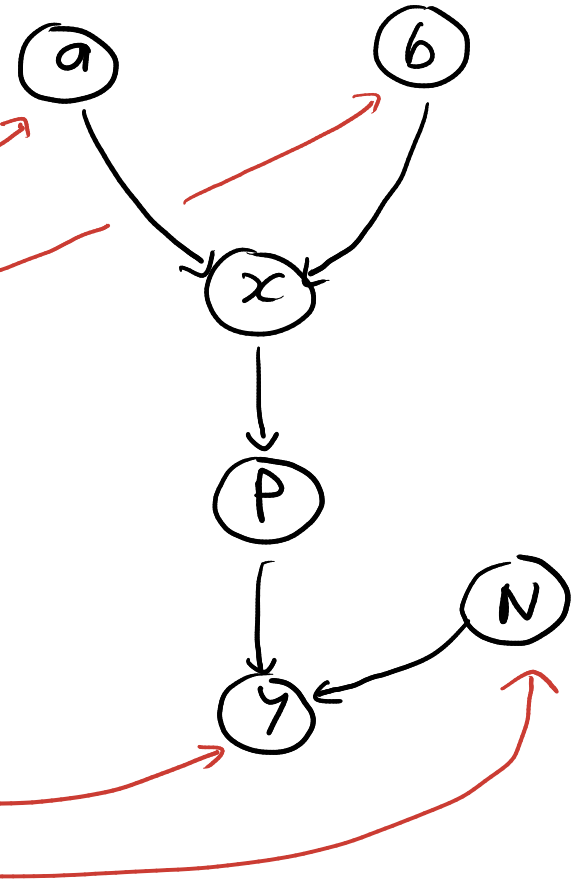
THE GUTS OF TURING

```
function betabinomial(__model__::DynamicPPL.Model, __varinfo__::DynamicPPL.AbstractVarInfo, __context__::DynamicPPL.AbstractContext, y, N, a, b; )¬
    begin¬
        begin¬
            var"##vn#319" = (AbstractPPL.VarName){:x}()¬
            var"##isassumption#320" = begin¬
                if (DynamicPPL.contextual_isassumption)(__context__, var"##vn#319")¬
                    if !((DynamicPPL.inargnames)(var"##vn#319", __model__)) || (DynamicPPL.inmissings)(var"##vn#319", __model__)¬
                        true¬
                    else¬
                        x === missing¬
                    end¬
                else¬
                    false¬
                end¬
            end¬
            if var"##isassumption#320"¬
                begin¬
                    (var"##value#322", __varinfo__) = (DynamicPPL.tilde_assume!!)(__context__, (DynamicPPL.unwrap_right_vn)((DynamicPPL.check_tilde_rhs)(Beta(a, b)), va
                    x = var"##value#322"¬
                    var"##value#322"¬
                end¬
            else¬
                if !((DynamicPPL.inargnames)(var"##vn#319", __model__))¬
                    x = (DynamicPPL.getvalue_nested)(__context__, var"##vn#319")¬
                end¬
                (var"##value#321", __varinfo__) = (DynamicPPL.tilde_observe!!)(__context__, (DynamicPPL.check_tilde_rhs)(Beta(a, b)), x, var"##vn#319", __varinfo__)¬
                var"##value#321"¬
            end¬
        end¬
        p = 1 - x¬
        begin¬
            var"##retval#327" = begin¬
                var"##vn#323" = (AbstractPPL.VarName){:y}()¬
                var"##isassumption#324" = begin¬
                    if (DynamicPPL.contextual_isassumption)(__context__, var"##vn#323")¬
                        if !((DynamicPPL.inargnames)(var"##vn#323", __model__)) || (DynamicPPL.inmissings)(var"##vn#323", __model__)¬
                            true¬
                        else¬
                            ⋮
```

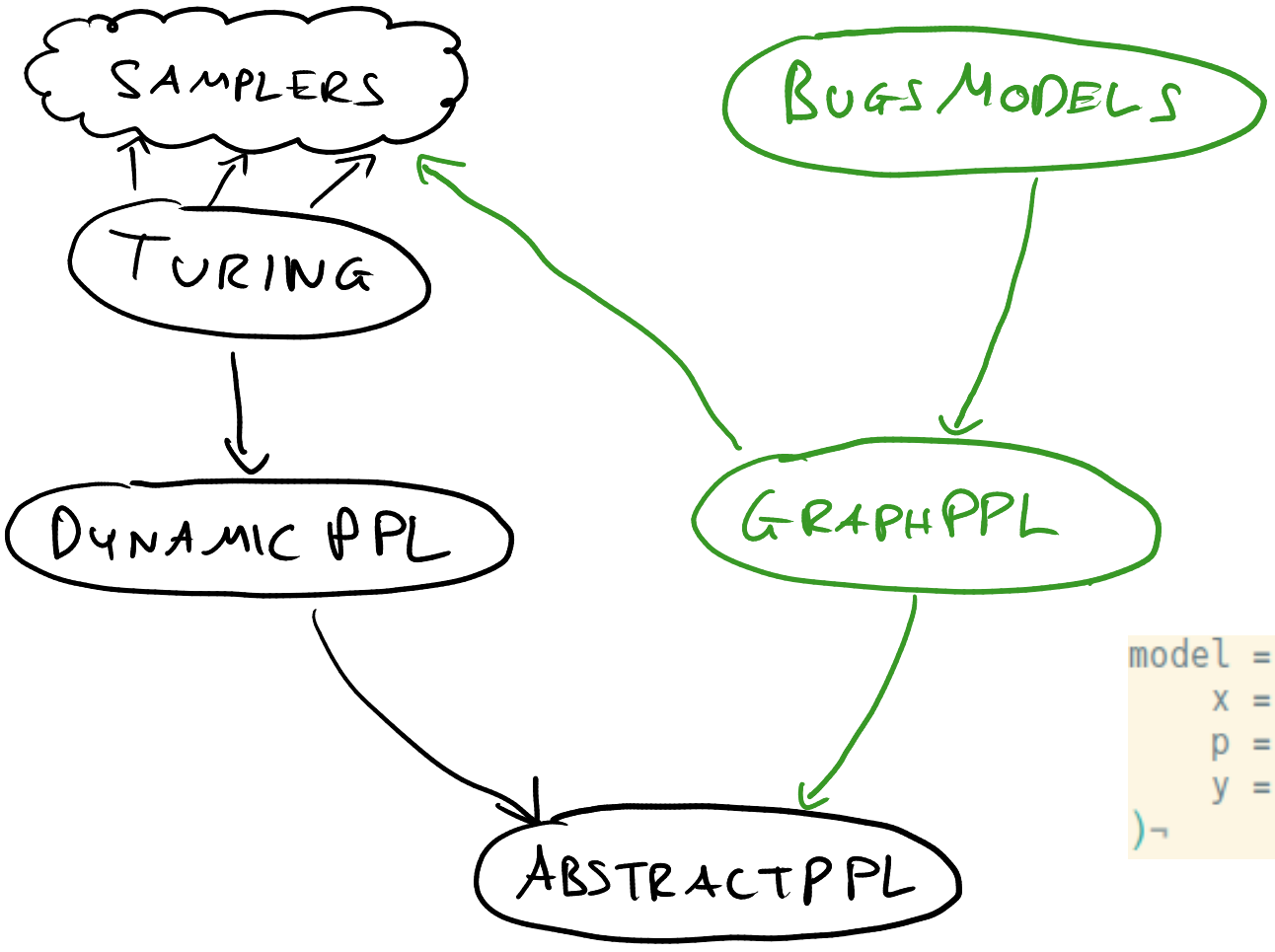
$x \sim \text{Beta}(a, b)$

BUGS, SIMPLIFIED

```
model {¬
    x ~ dbeta(a,b)¬
    p <- 1 - x¬
    y ~ dbin(p, N)¬
}¬
¬
data = list(N = 5, a = 2, b = 2, y = c(1,3,3,2,3))
```



A Possible Future




```
julia> bugsmodel"""
            x ~ dbeta(a,b)
            p <- 1 - x
            y ~ dbin(p, N)
        """
quote
    $(Expr(:~, :x, :(dbeta(a, b))))
    p = 1 - x
    $(Expr(:~, :y, :(dbin(p, N))))
end
```

```
model = (¬
    x = (0.0, () -> Beta(a, b), :Stochastic),¬
    p = (0.0, (x,) -> 1 - x, :Logical),¬
    y = (0.0, (p,) -> Binomial(N, p), :Stochastic)¬
)¬
```

~~PRIMITIVE STAN~~ ADVANCED METAPROGRAMMING

```
julia> Meta.show_sexpr(m)
(:block,
  (:~, :x, (:call, :Beta, :a, :b)),
  (:(=), :p, (:call, :-, 1, :x)),
  (:~, :y, (:call, :Binomial, :N, :p))
)
```

$x \sim dbeta(a,b)$
$p = 1 - x$
$y \sim dbin(p,N)$

```
julia> to_density(m, :x; y = [1,3,3,2,3], N = 5, a = 2, b = 2)
quote
    function (x,)
        __target__ = 0.0
        y = [1, 3, 3, 2, 3]
        N = 5
        a = 2
        b = 2
        __target__ += logpdf(Beta(a, b), x)
        p = 1 - x
        __target__ += logpdf(Binomial(N, p), y)
        return __target__
    end
end
```

Dirty Secrets...

```
function bugs_to_julia(s)¬
    # remove parentheses around loops¬
    s = replace(s, r"for\p{Zs}*\((.*)\)\p{Zs}*{" => s"for \1 {")¬
    ¬
    s = replace(¬
        s,¬
        "<-" => "=",¬
        # blocks in if and for replaced by respective delimiters (; ≃ \n)¬
        "{" => ";",¬
        "}" => "end",¬
        # empty slices (with lookahead to replace multiple in a series)¬
        r"\[\p{Zs}*\]" => "[:]",¬
        r"\[\p{Zs}*(?=,)" => "[:",¬
        r",\p{Zs}*(?=[,\]])" => ",:",¬
        # ignore reserved words (\b is word boundary)¬
        r"\b(in|for|if|C|T)\b" => s"\1",¬
        # ignore floats (could otherwise overlap with identifiers: ., E, e)¬
        r"(((\p{N}+\.\p{N}+)|(\p{N}+\.?))([eE][+-]?\p{N}+)?)" => s"\1",¬
        # wrap variable names in var-strings (to allow variable names with .)¬
        r"((?:(?:\p{L}\p{M}*)|\.)(?:(?:\p{L}\p{M}*)|\.|\p{N})*)" => s"var\"\1\"", ¬
    )¬
¬
    # special censoring/truncation syntax is converted to function calls, with `nothing`¬
    # inserted for left-out bounds¬
    s = replace(¬
        s,¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*T\p{Zs}*\(\p{Zs}*,(.+)\)" => s"truncated(\1, nothing, \2)",¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*T\p{Zs}*\((.+),\p{Zs}*\)" => s"truncated(\1, \2, nothing)",¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*T\p{Zs}*\((.+),(.+)\)" => s"truncated(\1, \2, \3)",¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*C\p{Zs}*\(\p{Zs}*,(.+)\)" => s"censored(\1, nothing, \2)",¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*C\p{Zs}*\((.+),\p{Zs}*\)" => s"censored(\1, \2, nothing)",¬
        r"(var\"[^\"]+\"\([^~<=]*\))\p{Zs}*C\p{Zs}*\((.+),(.+)\)" => s"censored(\1, \2, \3)",¬
    )¬
¬
    return s¬
end¬
```

SEMANTIC FORMALIZATION

TYPE QUESTIONS?

```
x[1] <- 10¬
x[2] ~ dnorm()¬
for (i = 1:x[1]) { ... }¬
```

```
x[2] ~ dnorm()¬
for (i = 1:x[1]) { ... }¬
¬
data = list(x = c(10, NA), ...)
```

C(10, NA)

```
x ~ dbeta(a,b)¬
p <- 1 - x¬
y ~ dbin(p, N)¬
```

```
x ~ dnorm(mu, tau)¬
tau ~ dgamma(r, lambda)¬
```

```
x ~ dnorm(mu, tau)¬
tau ~ dnorm(10, 1)
```

TYPING RULES!

$$\frac{T_1 <: T_2}{\text{Tensor}\{T_1,\ k\} <: \text{Tensor}\{T_2,\ k\}}$$

$$\frac{\Gamma \vdash f :: T_1, \ldots, T_n \to U \quad \Gamma \vdash x_1 :: V_1 @ \alpha_1, \ldots, x_n :: V_n @ \alpha_n \quad V_1 <: T_1, \ldots, V_n <: T_n}{\Gamma \vdash f(x_1, \ldots, x_n) :: U @ (\alpha_1 \sqcup \ldots \sqcup \alpha_n)}$$

A PATH FORWARD

- SIMPLE PPL : EVALUATION & MANIPULATION
  → PAVAN / GSOC
- ABSTRACTPPL & FRIENDS : CONSOLIDATING ABSTRACTIONS
  → TURING TEAM & OTHERS
- BUGS MODELS
  - SPECIFYING SYNTAX & SEMANTICS
  - ABSTRACT INTERPRETATION (TRACE INFERENCE, COLOR CHECKING, SET CONSTRAINTS, AD, ...)
  - R ⇌ JULIA MAPPING (DISTRIBUTIONS, LINKS)