Soss: Declarative Probabilistic Programming via Runtime Code Generation

Chad Scherrer and Taine Zhao
RelationalAI University of Tsukuba

Define a model

```
julia> m = @model X begin
           n = size(X,1)
           k = size(X,2)
           w ~ Normal(0,1) |> iid(k)
           Xw = X * w
           y ~ For(n) do j
               Normal(Xw[j], 0.1)
           end
       end;
```

Sample from the prior predictive distribution

```
julia> X = randn(5,2)
julia> y = rand(m(X=X)).y
5-element Array{Float64,1}:
  0.3768704663975828
  1.4621562961167427
 -1.6208481557729684
  0.1637164408878848
 -0.7876617758997144
```

Sample from the posterior predictive distribution

```
julia> pred = predict(m(X=X), post);
julia> y_rep = particles(pred).y
5-element Array{Particles{Float64,1000},1}:
  0.465 ± 0.11
  1.4 ± 0.12
 -1.64 ± 0.12
  0.212 ± 0.1
 -0.757 ± 0.13
```

Posterior Predictive Checks (here Bayesian p-values)

```
julia> y_rep .< y
5-element Array{Particles{Bool,1000},1}:
 0.198 ± 0.4
 0.69 ± 0.46
 0.57 ± 0.5
 0.307 ± 0.46
 0.415 ± 0.49
```

Sample from the posterior

```
julia> post = dynamicHMC(m(X=X), (y=y,));
julia> particles(post)
(w = Particles{Float64,1000}[1.12 ± 0.097, 0.424 ± 0.035],)
```

A Soss model is a function from its arguments to a distribution over named tuples

Model's type includes the model itself

```
julia> typeof(m).parameters[1]
NamedTuple{(:X,),T} where T<:Tuple

julia> typeof(m).parameters[2]
TypeEncoding(begin
    n = size(X, 1)
    k = size(X, 2)
    w ~ Normal(0, 1) |> iid(k)
    Xw = X * w
    y ~ For(n) do j
            Normal(Xw[j], 0.1)
        end
end)

julia> typeof(m).parameters[3]
TypeEncoding(Main)
```

Model fields index into expressions

```
julia> m.args
1-element Array{Symbol,1}:
 :X

julia> m.dists
( w = :(Normal(0, 1) |> iid(k))
, y = :(For(n) do j
      Normal(Xw[j], 0.1)
  end)
)

julia> m.vals
( n = :(size(X, 1))
, k = :(size(X, 2))
, Xw = :(X * w)
)
```

Runtime codegen, staged compilation

```
julia> Soss.sourceLogpdf(m)
quote
    _ℓ = 0.0
    n = size(X, 1)
    k = size(X, 2)
    _ℓ += logpdf(Normal(0, 1) |> iid(k), w)
    Xw = X * w
    _ℓ += logpdf(For(n) do j
                Normal(Xw[j], 0.1)
            end, y)
    return _ℓ
end

julia> Soss.sourceRand(m)
:(_rng->begin
          n = size(X, 1)
          k = size(X, 2)
          w = rand(_rng, iid(k, Normal(0, 1)))
          Xw = X * w
          y = rand(_rng, For(((j,)->begin
                                Normal(Xw[j], 0.1)
                            end), n))
          (n = n, k = k, Xw = Xw, w = w, y = y)
      end)
```

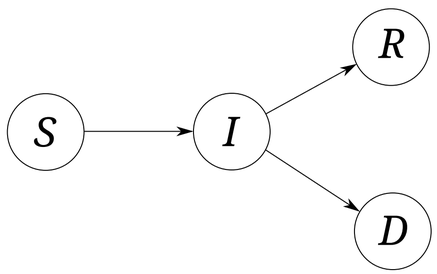
Forward and reverse DAG edges

```
julia> digraph(m).N
Dict{Symbol,Set{Symbol}} with 6 entries:
  :Xw => Set([:y])
  :w  => Set([:Xw])
  :n  => Set([:y])
  :k  => Set([:w])
  :y  => Set{Symbol}()
  :X  => Set([:Xw, :n, :k])

julia> digraph(m).NN
Dict{Symbol,Set{Symbol}} with 6 entries:
  :Xw => Set([:w, :X])
  :w  => Set([:k])
  :n  => Set([:X])
  :k  => Set([:X])
  :y  => Set([:Xw, :n])
  :X  => Set{Symbol}()
```

```
julia> Soss.statements(m)
6-element Array{Soss.Statement,1}:
 Soss.Arg(:X)
 Soss.Assign(:n, :(size(X, 1)))
 Soss.Assign(:k, :(size(X, 2)))
 Soss.Assign(:Xw, :(X * w))
 Soss.Sample(:w, :(Normal(0, 1) |> iid(k)))
 Soss.Sample(:y, :(For(n) do j
       Normal(Xw[j], 0.1)
   end))
```

A simplified epidemiology Markov transition

```
mstep = @model pars,state begin
    # Parameters
    α = pars.α        # Daily transmission rate
    β = pars.β        # Daily recovery rate
    γ = pars.γ        # Daily case fatality rate

    # Starting counts
    s0 = state.s      # Susceptible
    i0 = state.i      # Infected
    r0 = state.r      # Recovered
    d0 = state.d      # Deceased
    n = s0 + i0 + r0  # Population

    # Transitions between states
    si ~ Binomial(s0, α * i0 / n)
    ir ~ Binomial(i0, β)
    id ~ Binomial(i0 - ir, γ)

    # Updated counts
    next = ( s = s0 - si
           , i = i0 + si - ir - id
           , r = r0 + ir
           , d = d0 + id
           )
end;
```



Sample from the posterior

```
julia> ppost
(γ = 0.000102 ± 7.1e-7, β = 0.00314 ± 3.7e-6, α = 0.00975 ± 6.9e-6)
```

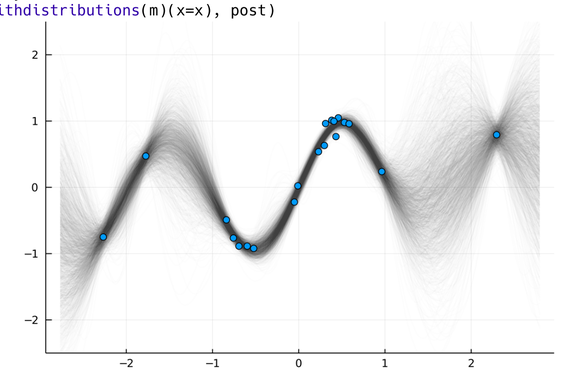
Markov process model using the transition

```
m = @model s0 begin
    α ~ Uniform()
    β ~ Uniform()
    γ ~ Uniform()
    pars = (α=α, β=β, γ=γ)
    x ~ MarkovChain(pars, mstep(pars=pars, state=s0))
end
```

```
# R₀
julia> ppost.α / (ppost.β + ppost.γ)
Particles{Float64,1000}
 3.00623 ± 0.0042

# Case fatality rate
julia> ppost.γ / (ppost.β + ppost.γ)
Particles{Float64,1000}
 0.0314686 ± 0.000203

# Implied infection duration
julia> 1/(ppost.β + ppost.γ)
Particles{Float64,1000}
 308.335 ± 0.376
```

Gaussian process with kernel uncertainty using AbstractGPs.jl

```
m = @model x begin
    # Priors.
    α ~ LogNormal(0.0, 0.1)
    ρ ~ LogNormal(0.0, 1.0)
    σ ~ LogNormal(0.0, 1.0)

    # Covariance function.
    kernel = α^2 * transform(SEKernel(), 1/(ρ*√2))

    # GP (implicit zero-mean).
    gp = GP(kernel)

    # Sampling Distribution (MvNormal likelihood).
    y ~ gp(x, σ^2 + 1e-6)  # add 1e-6 for numerical stability.
end


# Get some fake data
x = randn(20)

y =  sinpi.(x) .+ 0.1 .* randn(20);

# Sample from the posterior
post = dynamicHMC(m(x=x), (;y=y))


pred = [posterior(p._y_dist, y)
    for p in predict(Soss.withdistributions(m)(x=x), post)
]
```



Soss builds on many other libraries.
Thank you to the authors who've made it possible!

```
(Soss) pkg> status
Project Soss v0.15.3
Status `~/git/Soss.jl/Project.toml`
  [0bf59076] AdvancedHMC v0.2.25
  [76274a88] Bijectors v0.8.4
  [324d7699] CategoricalArrays v0.8.2
  [d360d2e6] ChainRulesCore v0.9.10
  [163ba53b] DiffResults v1.0.2
  [31c24e10] Distributions v0.23.8
  [ced4e74d] DistributionsAD v0.6.9
  [bbc10e6e] DynamicHMC v2.2.0
  [1a297f60] FillArrays v0.8.14
  [f6369f11] ForwardDiff v0.10.12
  [6b9d7cbe] GeneralizedGenerated v0.2.7
  [86223c79] Graphs v0.10.3
  [c8e1da08] IterTools v1.3.0
  [b964fa9f] LaTeXStrings v1.2.0
  [5078a376] LazyArrays v0.16.16
  [6fdf6af0] LogDensityProblems v0.10.3
  [bdcacae8] LoopVectorization v0.8.26
  [d8e11817] MLStyle v0.4.6
  [1914dd2f] MacroTools v0.5.5
  [dbb5928d] MappedArrays v0.2.2
  [0987c9cc] MonteCarloMeasurements v0.9.5
  [d9ec5142] NamedTupleTools v0.13.6
  [3cdcf5f2] RecipesBase v1.1.0
  [189a3867] Reexport v0.2.0
  [ae029012] Requires v1.0.2
  [37e2e3b7] ReverseDiff v1.4.3
  [21efa798] SIMDPirates v0.8.25
  [efcf1570] Setfield v0.7.0
  [55797a34] SimpleGraphs v0.6.3
  [ec83eff0] SimplePartitions v0.3.0
  [b2aef97b] SimplePosets v0.1.3
  [276daf66] SpecialFunctions v0.10.3
  [4c63d2b9] StatsFuns v0.9.5
  [84d833dd] TransformVariables v0.3.10
  [de0858da] Printf
  [9a3f8284] Random
  [10745b16] Statistics
```