Efficient inference with discrete parameters in Stan

Maria Gorinova, Andy Gordon, Charles Sutton, Matthijs Vákár

Hamiltonian Monte Carlo needs $\nabla p(\boldsymbol{\theta}, \mathfrak{D})$, requiring the joint density defined by the model to be (piecewise) continuous.

PROBLEM: HMC **cannot** be used for inference as is if there are **discrete parameters** in the model.

SOLUTION: Automatically marginalize the discrete parameters using information-flow analysis.

$p(y, e, l, d)$

```
real e ~ exponential(r_e);
real l ~ exponential(r_l);
int d ~ discrete_uniform(1, T);

data int y[T];
for (t in 1:T)
   y[t] ~ poisson(t < d ? e : l);
```

$p(y, e, l) = \sum_{d=1}^{T} p(y, e, l, d)$

```
real e ~ exponential(r_e);
real l ~ exponential(r_l);

data int y[T];
real probs[T]; // init at 1
for (d in 1:T)
  probs[d] *= uniform_lpmf(d | 1, T)
  for (t in 1:T){
   probs[d] *=
      poisson_pmf(y[t] | t < d ? e : l);
  }
factor(sum(probs))
```

$p(d \mid y, e, l) \propto p(y, e, l, d)$

```
int d = categorical_rng(probs);
```

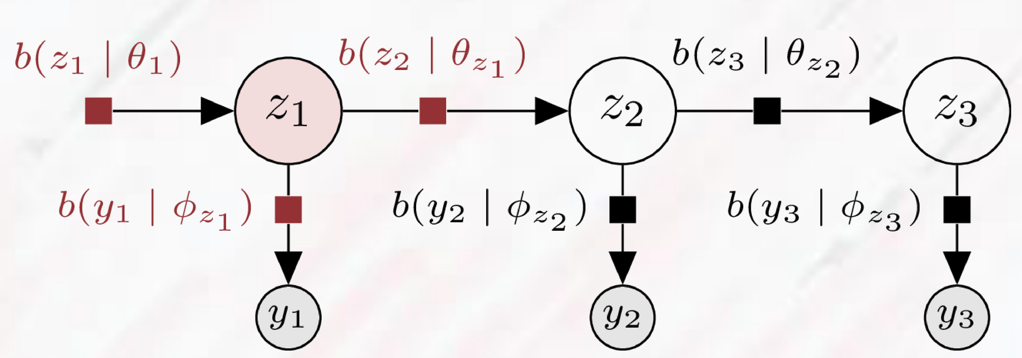
How do we decide which statements go where?

```
real phi0 ~ beta(1, 1);
real theta0 ~ beta(1, 1);

int<2> z1 ~ bernoulli(theta0);
real theta1 = theta0 * z1 + (1 - theta0) * (1 - z1);
int<2> z2 ~ bernoulli(theta1);
real theta2 = theta0 * z2 + (1 - theta0) * (1 - z2);
int<2> z3 ~ bernoulli(theta2);

real phi1 = phi0 * z1 + (1 - phi0) * (1 - z1);
real phi2 = phi0 * z2 + (1 - phi0) * (1 - z2);
real phi3 = phi0 * z3 + (1 - phi0) * (1 - z3);

data real y1 ~ normal(phi1, 1);
data real y2 ~ normal(phi2, 1);
data real y3 ~ normal(phi3, 1);
```

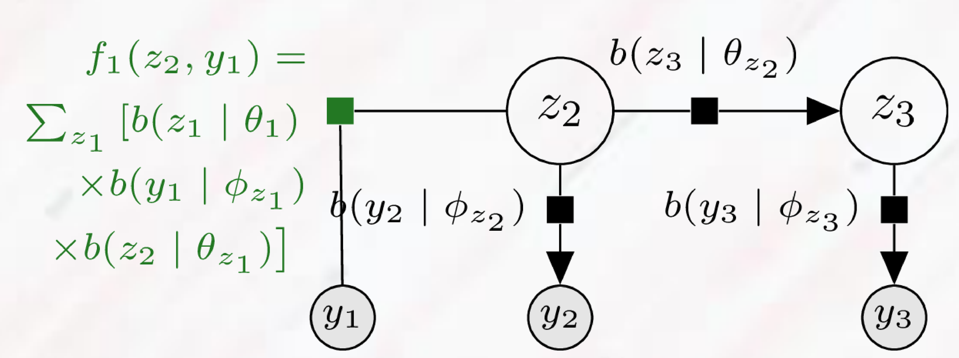
```
phi0 ~ beta(1, 1);
theta0 ~ beta(1, 1);

f1 = φ([int<2> z2]){
   elim(int<2> z1){
      z1 ~ bernoulli(theta0);
      theta1 = (theta0 * z1 + (1 - theta0) * (1 - z1));
      z2 ~ bernoulli(theta1);
      phi1 = (phi0 * z1 + (1 - phi0) * (1 - z1));
      y1 ~ normal(phi1, 1);
   }
}
factor(f1[z2]);

theta2 = (theta0 * z2 + (1 - theta0) * (1 - z2));
z3 ~ bernoulli(theta2);
phi2 = (phi0 * z2 + (1 - phi0) * (1 - z2));
phi3 = (phi0 * z3 + (1 - phi0) * (1 - z3));
y2 ~ normal(phi2, 1);
y3 ~ normal(phi3, 1);

gen(int z1){
   z1 ~ bernoulli(theta0);
   theta1 = (theta0 * z1 + (1 - theta0) * (1 - z1));
   z2 ~ bernoulli(theta1);
   phi1 = (phi0 * z1 + (1 - phi0) * (1 - z1));
   y1 ~ normal(phi1, 1);
}

theta1 = (theta0 * z1 + (1 - theta0) * (1 - z1));
phi1 = (phi0 * z1 + (1 - phi0) * (1 - z1));
```

What if we have many discrete variables? Use variable elimination:

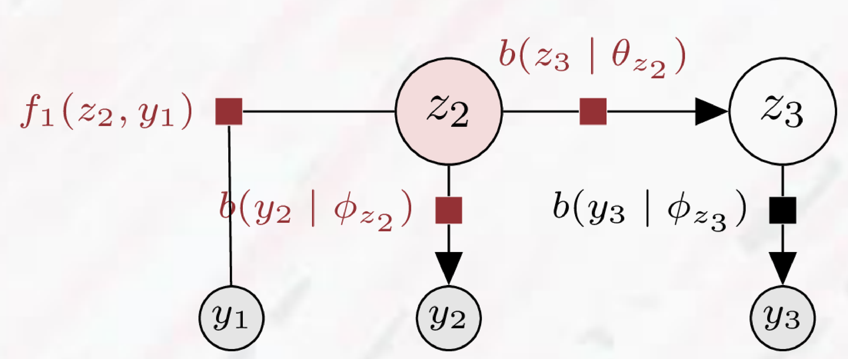
1. To eliminate z1: remove z1 and all its neighboring factors from the graph.

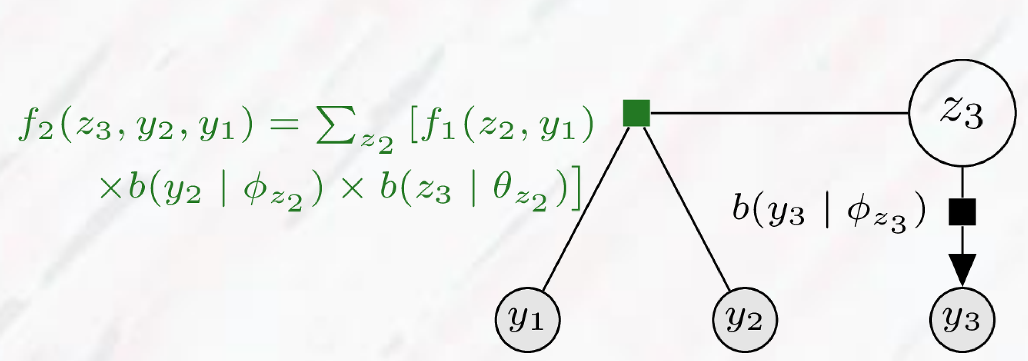


2. Create a new factor by summing out z1 from the product of the removed factors. Connect to z1's neighbors



3. Repeat for each discrete variable, eliminating them one by one…



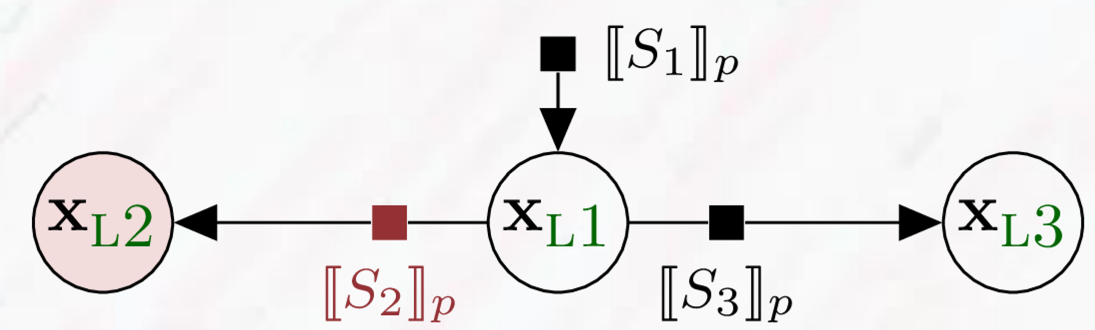


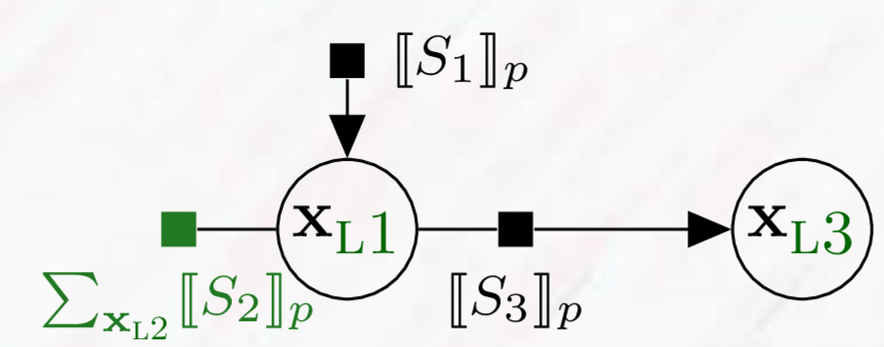
Conditional independence by typing

Each program variable has one of three level types: $\boldsymbol{L1}$, $\boldsymbol{L2}$ or $\boldsymbol{L3}$, which form a semilattice:

$$\boldsymbol{L1} \leq \boldsymbol{L2} \qquad \boldsymbol{L1} \leq \boldsymbol{L3}$$

Continuous parameters are of type $\boldsymbol{L1}$, and the variable we want to eliminate is of type $\boldsymbol{L2}$. We infer the types of other variables, slice the program accordingly, and marginalize efficiently:





m.gorinova@ed.ac.uk @migorinova https://github.com/mgorinova/SlicStan

THE UNIVERSITY of EDINBURGH DATASCIENCE EPSRC CENTRE FOR DOCTORAL TRAINING EPSRC