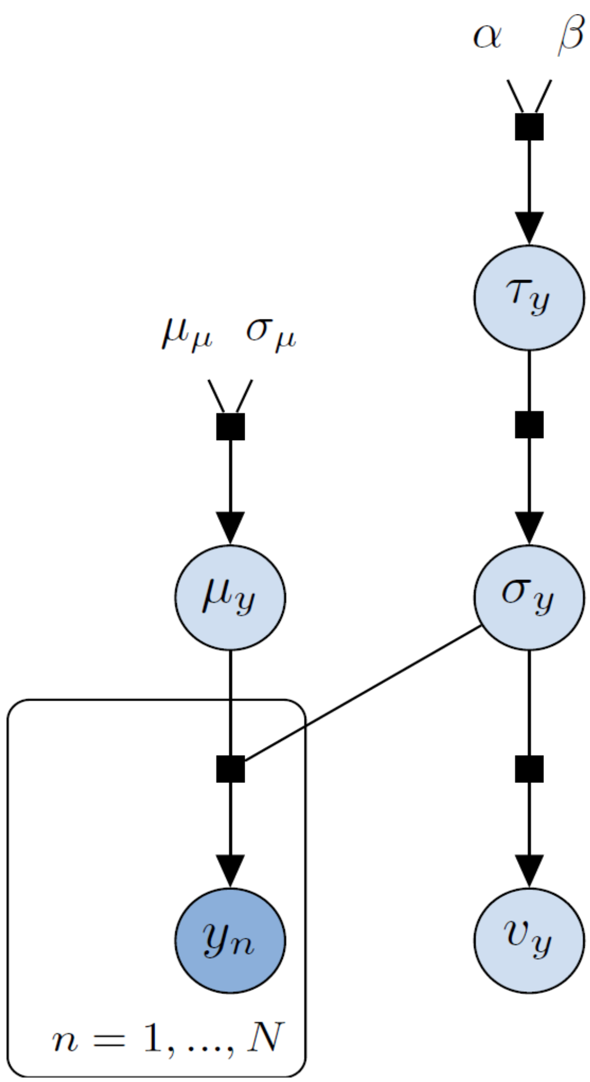
SlicStan

A Blockless Stan-like Language

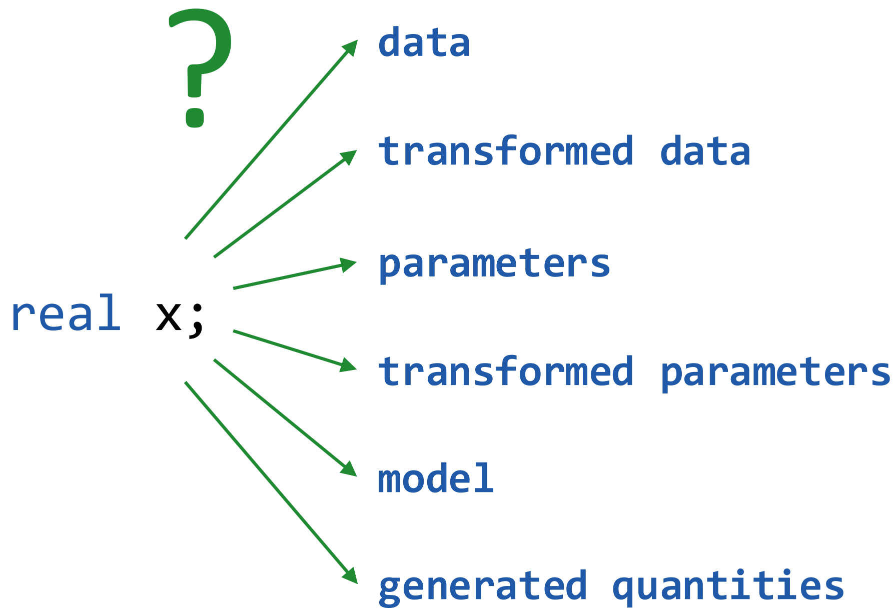
Maria I. Gorinova, Andrew D. Gordon, Charles Sutton




```
data {
   int N;
   real y[N];
   real mu_mu;
   real sigma_mu;
}
transformed data {
   real alpha = 0.1;
   real beta = 0.1;
}
parameters {
   real mu_y;
   real tau_y;
}
transformed parameters {
   real sigma_y;
   sigma_y = pow(tau_y, -0.5);
}
model {
   tau_y ~ gamma(alpha, beta);
   mu_y ~ normal(mu_mu, sigma_mu);
   y ~ normal(mu_y, sigma_y);
}
generated quantities {
   real variance_y;
   variance_y = sigma_y * sigma_y;
}
```

What is the problem?

- Lack of compositionality
- Inflexible user-defined functions
- Non-trivial to optimise



SlicStan vs. Stan

```
real alpha = 0.1;
real beta = 0.1;
real tau_y ~ gamma(alpha, beta);

data real mu_mu;
data real sigma_mu;
real mu_y ~ normal(mu_mu, sigma_mu);

real sigma_y = pow(tau_y, -0.5);
data int N;
data real[N] y ~ normal(mu_y, sigma_y);

real variance_y = pow(sigma_y, 2);
```

```
data {
   int N;
   real y[N];
   real mu_mu;
   real sigma_mu;
}
transformed data {
   real alpha = 0.1;
   real beta = 0.1;
}
parameters {
   real mu_y;
   real tau_y;
}
transformed parameters {
   real sigma_y;
   sigma_y = pow(tau_y, -0.5);
}
model {
   tau_y ~ gamma(alpha, beta);
   mu_y ~ normal(mu_mu, sigma_mu);
   y ~ normal(mu_y, sigma_y);
}
generated quantities {
   real variance_y;
   variance_y = sigma_y * sigma_y;
}
```

Neal’s Funnel

SlicStan

```
def nc_normal(real m, real s) {
   real x_raw ~ normal(0, 1);
   return s * x_raw + m;
}
real y = nc_normal(0, 3);
real x = nc_normal(0, exp(y/2));
```

Stan

```
parameters {
   real y_raw;
   real x_raw;
}
transformed parameters {
   real y;
   real x;
   y = 3.0 * y_raw;
   x = exp(y/2) * x_raw;
}
model {
   y_raw ~ normal(0, 1);
   x_raw ~ normal(0, 1);
}
```

Information Flow

- Transfer of information between two variables

```
y = x + 5

if x > 5 then y = 1 else y = 0
```

Information Flow

$$\text{PUBLIC} < \text{SECRET}$$

`p:` PUBLIC`, s:` SECRET

✓ `s = p`

✗ `p = s`

✗ `if s then p = 1 else p = 0`

Information Flow in Stan

```
data {
   int N;
   real y[N];
   real mu_mu;
   real sigma_mu;
}
transformed data {
   real alpha = 0.1;
   real beta = 0.1;
}
parameters {
   real mu_y;
   real tau_y;
}
transformed parameters {
   real sigma_y;
   sigma_y = pow(tau_y, -0.5);
}
model {
   tau_y ~ gamma(alpha, beta);
   mu_y ~ normal(mu_mu, sigma_mu);
   y ~ normal(mu_y, sigma_y);
}
generated quantities {
   real variance_y;
   variance_y = sigma_y * sigma_y;
}
```

DATA

$<$

MODEL

$<$

GENQUANT

Key idea

- Find all **possible roles** a variable can have during inference, w.r.t. the **information flow**:

$$\text{DATA} < \text{MODEL} < \text{GENQUANT}$$

- `data real x` → `level(x)` = DATA
- `real x, ≠` → `level(x)` ≥ MODEL
- `x = foo(y)` → `level(x) ≥ level(y)`
- `x ~ foo(y)` → `level(x)` ≤ MODEL `and level(y)` ≤ MODEL

Key idea

- Find all **possible roles** a variable can have during inference, w.r.t. the **information flow**:

$$\mathrm{DATA} < \mathrm{MODEL} < \mathrm{GENQUANT}$$

  - `data real x` → `level(x) =` DATA
  - `real x, ≠` → `level(x) ≥` MODEL
  - `x = foo(y)` → `level(x) ≥ level(y)`
  - `x ~ foo(y)` → `level(x) ≤` MODEL `and level(y) ≤` MODEL
- Not unique… Which one do we choose?

Performance ordering

Block	Execution	Level
`data`	---	DATA
`transformed data`	once	DATA
`parameters`	---	MODEL
`transformed parameters`	once per leapfrog	MODEL
`model`	once per leapfrog	MODEL
`generated quantities`	once per sample	GENQUANT

$$\text{DATA} \leq \text{GENQUANT} \leq \text{MODEL}$$

Key insight

- Find all **possible roles** a variable can have during inference, w.r.t. the **information flow**:

$$\mathrm{DATA} \quad < \quad \mathrm{MODEL} \; < \quad \mathrm{GENQUANT}$$

- Choose the most **optimal role**, w.r.t. the **performance ordering**:

$$\mathrm{DATA} \quad \leq \quad \mathrm{GENQUANT} \quad \leq \quad \mathrm{MODEL}$$

Translation to Stan

1. Elaboration: calls to user-defined functions are statically unrolled.
2. Transformation: the SlicStan code is shredded into different Stan program blocks.

SlicStan

```
real alpha = 0.1;
real beta = 0.1;
real tau_y ~ gamma(alpha, beta);

data real mu_mu;
data real sigma_mu;
real mu_y ~ normal(mu_mu, sigma_mu);

real sigma_y = pow(tau_y, -0.5);
data int N;
data real[N] y;
y ~ normal(mu_y, sigma_y);

real variance_y = pow(sigma_y, 2);
```

SlicStan

```
DATA real alpha = 0.1;
DATA real beta = 0.1;
MODEL real tau_y ~ gamma(alpha, beta);

data DATA real mu_mu;
data DATA real sigma_mu;
MODEL real mu_y ~ normal(mu_mu, sigma_mu);

MODEL real sigma_y = pow(tau_y, -0.5);
data DATA int N;
data DATA real[N] y;
y ~ normal(mu_y, sigma_y);

GENQUANT real variance_y = pow(sigma_y, 2);
```

SlicStan

```
DATA real alpha = 0.1;
DATA real beta = 0.1;
MODEL real tau_y ~ gamma(alpha, beta);

data DATA real mu_mu;
data DATA real sigma_mu;
MODEL real mu_y ~ normal(mu_mu, sigma_mu);

MODEL real sigma_y = pow(tau_y, -0.5);
data DATA int N;
data DATA real[N] y;
y ~ normal(mu_y, sigma_y);

GENQUANT real variance_y = pow(sigma_y, 2);
```

Stan

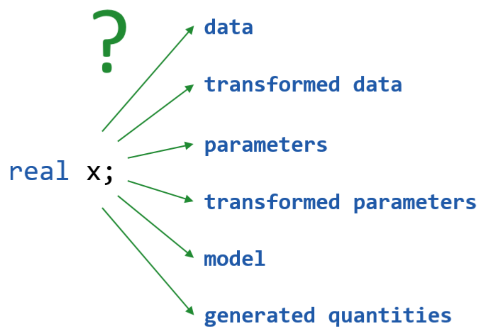
```
data {
   int N;
   real y[N];
   real mu_mu;
   real sigma_mu;
}
transformed data {
   real alpha = 0.1;
   real beta = 0.1;
}
parameters {
   real mu_y;
   real tau_y;
}
transformed parameters {
   real sigma_y;
   sigma_y = pow(tau_y, -0.5);
}
model {
   tau_y ~ gamma(alpha, beta);
   mu_y ~ normal(mu_mu, sigma_mu);
   y ~ normal(mu_y, sigma_y);
}
generated quantities {
   real variance_y;
   variance_y = sigma_y * sigma_y;
}
```

What comes next?

- User-defined functions vectorization.
- Deducing type constraints automatically.
- Formalising the semantics of Stan.

SlicStan: Improving Probabilistic Programming using Information Flow Analysis

Maria I. Gorinova, Andrew D. Gordon, Charles Sutton



$$\mathrm{DATA} < \mathrm{MODEL} < \mathrm{GENQUANT}$$

Information Flow

$$\mathrm{DATA} \leq \mathrm{GENQUANT} \leq \mathrm{MODEL}$$

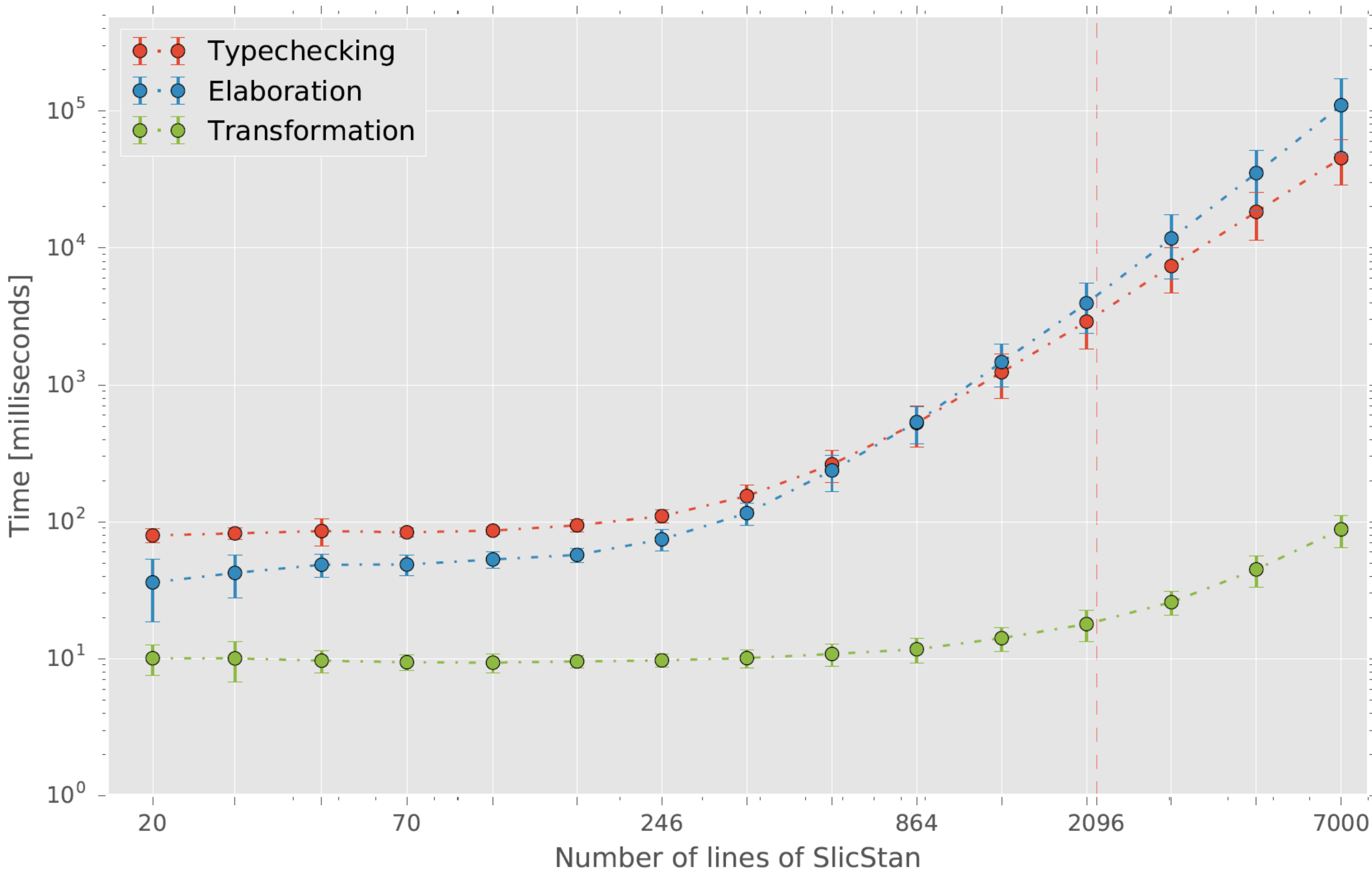
Performance Ordering

Email: m.gorinova@ed.ac.uk

SlicStan PPS page: https://tiny.cc/slicstan

THE UNIVERSITY of EDINBURGH

EPSRC


Typechecking
Elaboration
Transformation
Time [milliseconds]
10^0
10^1
10^2
10^3
10^4
10^5
20
70
246
864
2096
7000
Number of lines of SlicStan