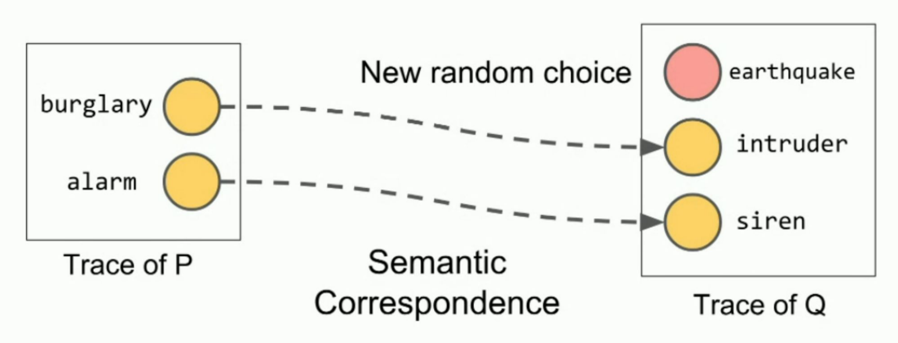
Incremental Inference for Probabilistic Programs

Authors: *Marco Cusumano-Towner, Benjamin Bichsel, Timon Gehr, Martin Vechev, Vikash K. Mansinghka*



TU Delft

Presented by: Jeroen Groenheide and Onno Verberne

Contributions of the paper

What?

A novel approach for approximate sampling based on incremental inference

Why?

Avoid expensive sampling computation by sampling from a ***known*** program

How?

Adapt traces from one program to another using a *Trace translator*

Optimise adapted traces (*samples*) using sequential Monte Carlo

Incremental inference

Given two probabilistic programs P and Q, and samples of P obtained using an existing inference algorithm, generate samples for Q by leveraging the samples for P.

- Construct a *trace translator* to adapt samples of P into samples of Q
- Compute *weights* for the adapted traces and reweight like in SMC
- Optionally perform *resampling* proportional to the computed weights
- Use *MCMC sampling* intermittently to increase approximation quality
- The generated output traces store all the inferred properties

Applications

When?

- When the posterior distributions of the programs are “close enough”
- When the programs are variants of the same model
- When the data the models are conditioned on is changed
- When the prior assumptions of the models are changed
- When model changes originate from an automated process

Else?

- Proceed with standard non-incremental inference
- Use the available traces to warm-start samplers like MCMC **?**

Applications

When?

- When the posterior distributions of the programs are "close enough"
- When the programs are variants of the same model

Why not always use MCMC with the available traces?

- When the prior assumptions of the models are changed

Traces might contain different number of random choices.
MCMC is less efficient when programs are very similar.

- Proceed with standard non-incremental inference
- Use the available traces to warm-start samplers like MCMC **?**

Getting technical

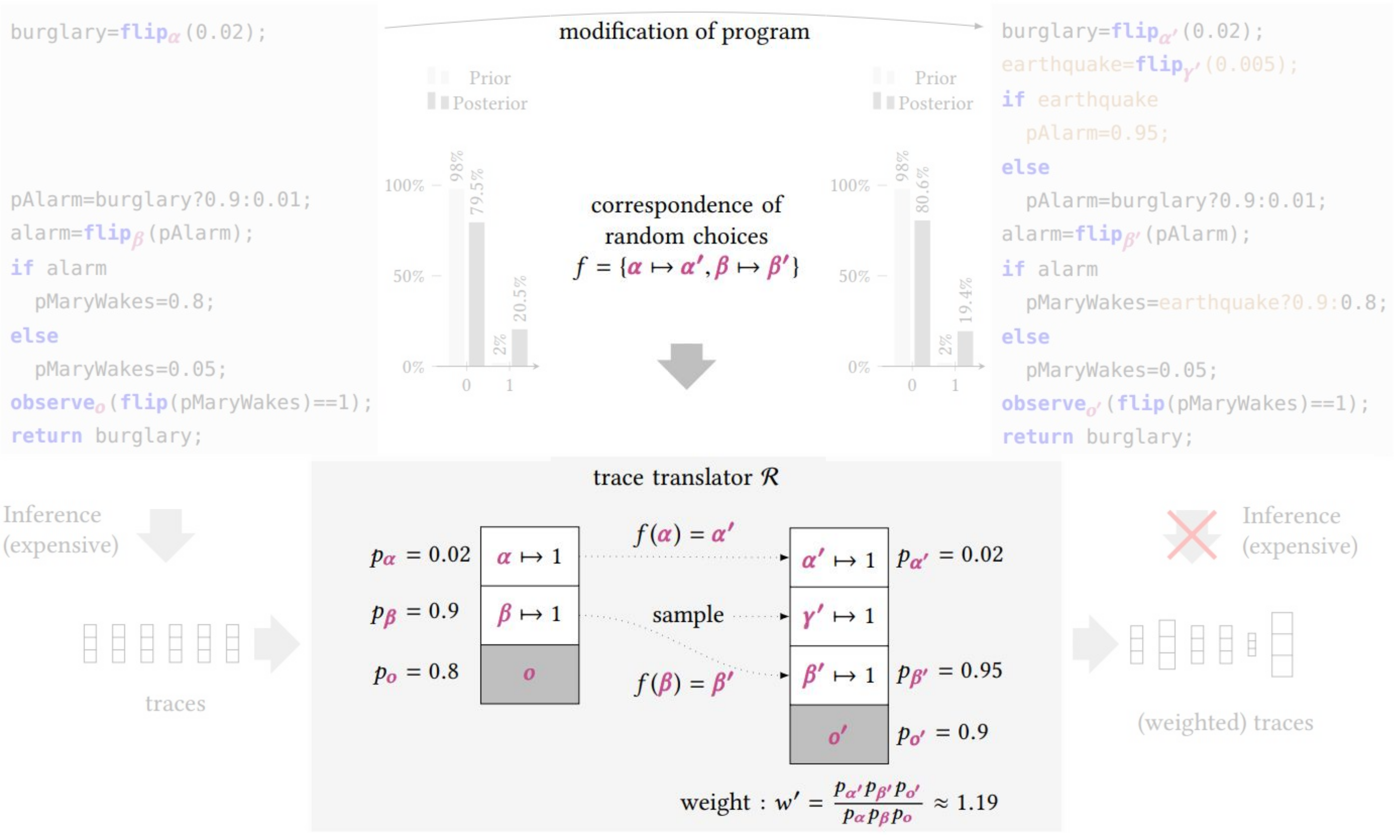
Incremental inference

Given two probabilistic programs P and Q, and samples of P obtained using an existing inference algorithm, generate samples for Q by leveraging the samples for P.

- Construct a ***trace translator*** to adapt samples of P into samples of Q
- Compute ***weights*** for the adapted traces and reweight like in SMC
- Optionally perform *resampling* proportional to the computed weights
- Use *MCMC sampling* intermittently to increase approximation quality
- The generated output traces store all the inferred properties

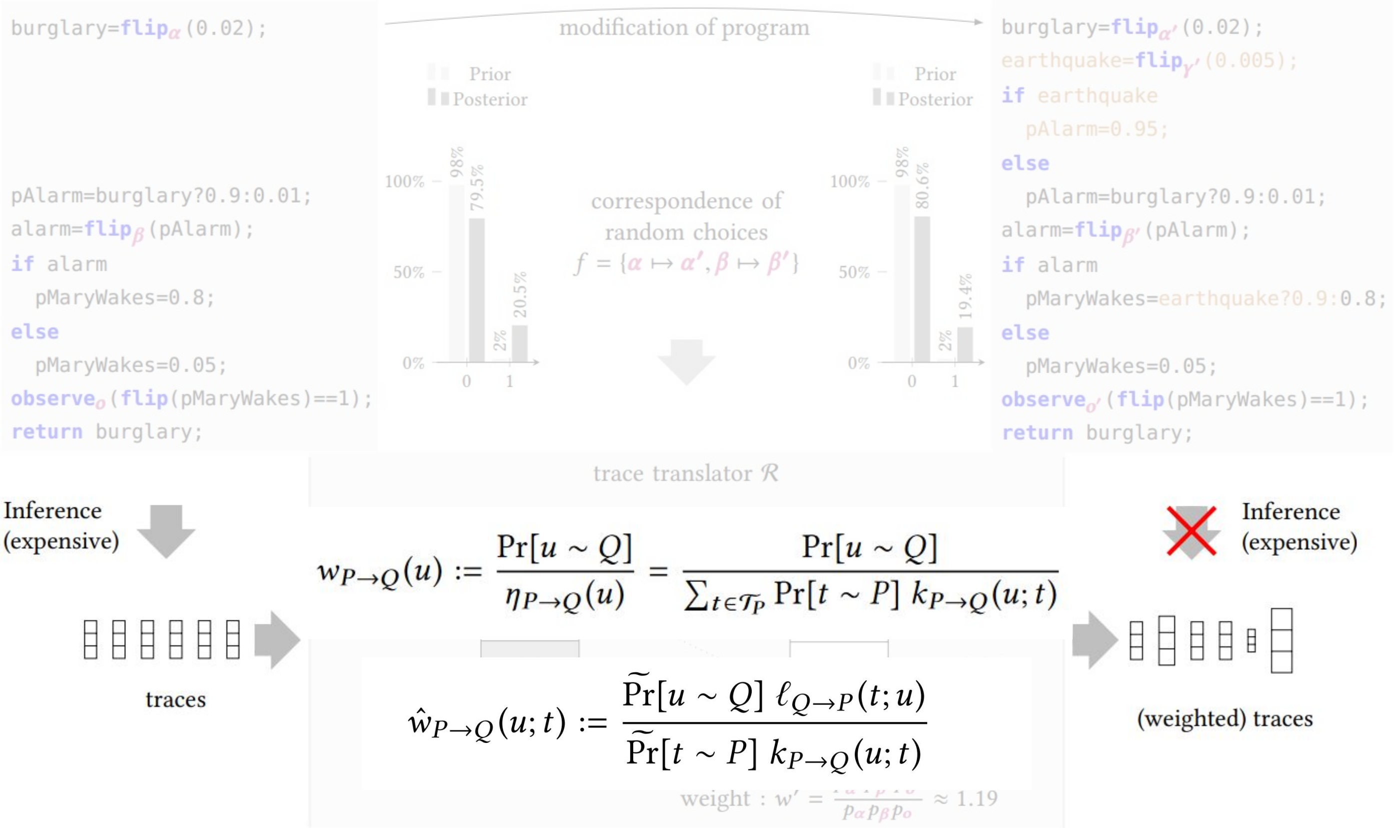
TU Delft

- Construct a ***trace translator*** to adapt samples of P into samples of Q



TU Delft

- Compute ***weights*** for the adapted traces and reweight like in SMC



TU Delft

Incremental inference

Given two probabilistic programs P and Q, and samples of P obtained using an existing inference algorithm, generate samples for Q by leveraging the samples for P.

- Construct a *trace translator* to adapt samples of P into samples of Q
- Compute *weights* for the adapted traces and reweight like in SMC
- Optionally perform *resampling* proportional to the computed weights
- Use ***MCMC sampling*** intermittently to increase approximation quality
- The generated output traces store all the inferred properties

TU Delft

Incremental inference

Given two probabilistic programs *P* and *Q*, and samples of *P* obtained using an existing inference algorithm, generate samples for *Q* by leveraging the samples for *P*.

Why can we use MCMC sampling here, but not before?

- Construct a *trace translator* to adapt samples of P into samples of Q

We apply MCMC to valid traces for target program Q

- Optionally perform *resampling* proportional to the computed weights
- Use ***MCMC sampling*** intermittently to increase approximation quality
- The generated output traces store all the inferred properties

Implementation

Step-by-Step

1. Establish correspondence function **f** between variables
2. Initialize P-score and Q-score to 0 (for weighting)
3. Run target program **Q** once to obtain trace **u**
4. For every variable in **u**, and every available trace **t**:
    a. Is the variable in trace **t**?
        i. **Yes?** Take the value from **t**.
           Increase Q-score by log probability of corresponding choice in **Q**
        ii. **No?** Sample a new value from the prior of **Q**.
    b. Is this an observation?
        i. Increase Q-score by log probability of observation
5. P-score = sum of log probabilities of choices and observations of **t**
6. Log weight = P-score - Q-score

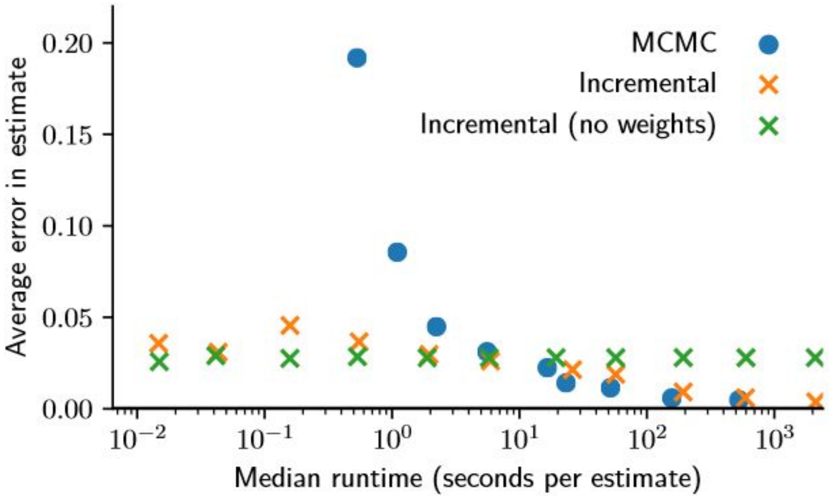
$$\hat{w}_{P\to Q}(u;t) := \frac{\widetilde{\Pr}[u \sim Q]\, \ell_{Q\to P}(t;u)}{\widetilde{\Pr}[t \sim P]\, k_{P\to Q}(u;t)}$$

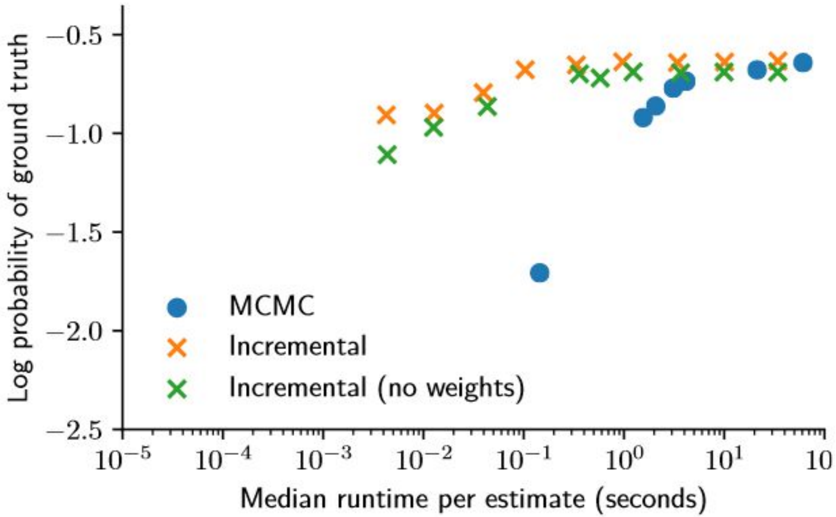
Recap

1. Establish variable correspondences
2. Collect traces **t** from original program **P**
3. Translate traces **t** to traces **u** in target program **Q**
4. Calculate weights for each step
5. Resample traces **u** according to weight (Optional)
6. Apply MCMC to traces **u** to improve their quality

The experiments (were bad)

Experimental results





Two different test setups
Showing different metrics

TU Delft

Questions?