Automatic Discovery of Static Structures in Probabilistic Programs

Revised title: Automatic Alignment of Sequential Monte Carlo Inference in Higher-Order Probabilistic Programs

Daniel Lundén[1] David Broman[1] Fredrik Ronquist[2] Lawrence M. Murray[3]

[1]KTH Royal Institute of Technology, Stockholm, Sweden

[2]Swedish Museum of Natural History, Stockholm, Sweden [3]Uppsala University, Uppsala, Sweden

Financially supported by the Swedish Foundation for Strategic Research.

Our view of probabilistic programming languages

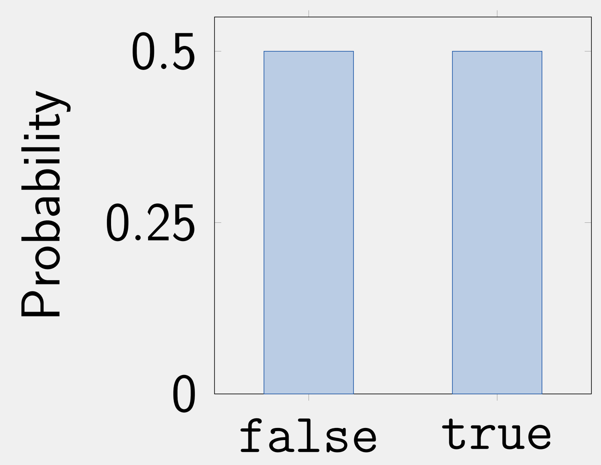
- Programming languages with
  - constructs for **sampling** from probability distributions, and
  - constructs for **conditioning on data**.
- Additionally, if a language includes
  - **stochastic branching**, and
  - **recursion**,

  we call it a **universal** probabilistic programming language.
- Commonly used family of inference algorithms: **Sequential Monte Carlo (SMC)** methods

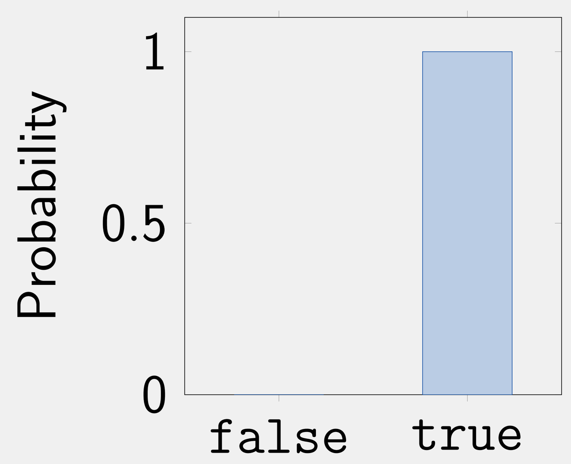
The problem with SMC for probabilistic programming

- A toy model over booleans. Simply executing the program has equal probability of returning `true` and `false`, always with the weight -1.

```
if flip() then {
  weight(-1000)
  weight(999)
  false
} else {
  weight(-1)
  true
}
```



- WebPPL result running SMC:



- Anglican error at runtime running SMC:

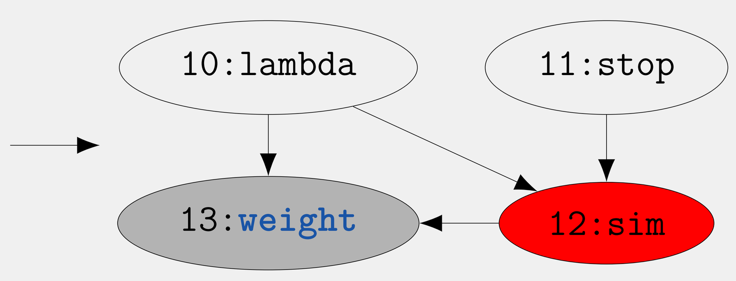
```
some observe directives are not global
```

Solution: Automatic alignment of `weight` calls

- We do a static analysis on a program, identifying **aligned** calls to weight. The **red** node indicates we cannot align anything within `sim` due to stochastic branching and recursion.

```
function sim(stop, lambda) {
  t ~ exponential(lambda)
  if t <= stop then {
    weight(2.0)
    sim(stop-t, lambda+0.1)
  } else t
}

function model() {
  lambda ~ gamma(1.0, 1.0)
  stop ~ gamma(10.0, 3.0)
  sim = sim(stop, lambda)
  weight(sim+lambda)
  lambda
}
```



The key: Discovering static structures

- A minimal target language for the static analysis:

  $\mathbf{t} ::= x \mid c \mid \lambda x.\mathbf{t} \mid \mathbf{t}_1\ \mathbf{t}_2 \mid \texttt{fix}\ \mathbf{t} \mid \texttt{if}\ \mathbf{t}_1\ \texttt{then}\ \mathbf{t}_2\ \texttt{else}\ \mathbf{t}_3$
  $\mid \texttt{sample}\ \mathbf{t} \mid \texttt{weight}\ \mathbf{t}$

- Example 1:

  $(\lambda x.\texttt{if sample}\ \mathit{dist}\ \texttt{then}\ (x\ c_1)\ \texttt{else}\ c_2)\ (\lambda y.y)$

- Example 2:

  $(\lambda a.(\lambda b.a\ b)\ (\lambda c.c))\ (\lambda d.\texttt{if sample}\ \mathit{dist}\ \texttt{then}\ (d\ c_1)\ \texttt{else}\ c_2)$

Approach: 0-CFA

1. **Label** the program:

   $((\lambda x.(\texttt{if}\ (\texttt{sample}\ \mathit{dist}^1)^2\ \texttt{then}\ (x^3\ c_1^4)^5\ \texttt{else}\ c_2^6)^7)^8\ (\lambda y.y^9)^{10})^{11}$

2. **Generate constraints** for the program:

$$\begin{aligned}\{\ &\{\textbf{stoch}\} \subseteq S_2, \{(\lambda y.\cdot^9)^{10}\} \subseteq S_{10}, \{(\lambda x.\cdot^7)^8\} \subseteq S_8,\\ &S_y \subseteq S_9, S_5 \subseteq S_7, S_6 \subseteq S_7, S_x \subseteq S_3,\\ &\{(\lambda x.\cdot^7)^8\} \subseteq S_8 \Rightarrow S_{10} \subseteq S_x, \{(\lambda x.\cdot^7)^8\} \subseteq S_8 \Rightarrow S_7 \subseteq S_{11},\\ &\{(\lambda y.\cdot^9)^{10}\} \subseteq S_8 \Rightarrow S_{10} \subseteq S_x, \{(\lambda y.\cdot^9)^{10}\} \subseteq S_8 \Rightarrow S_9 \subseteq S_{11},\\ &\{(\lambda x.\cdot^7)^8\} \subseteq S_3 \Rightarrow S_4 \subseteq S_x, \{(\lambda x.\cdot^7)^8\} \subseteq S_3 \Rightarrow S_7 \subseteq S_5,\\ &\{(\lambda y.\cdot^9)^{10}\} \subseteq S_3 \Rightarrow S_4 \subseteq S_y, \{(\lambda y.\cdot^9)^{10}\} \subseteq S_3 \Rightarrow S_9 \subseteq S_5\ \}\end{aligned}$$

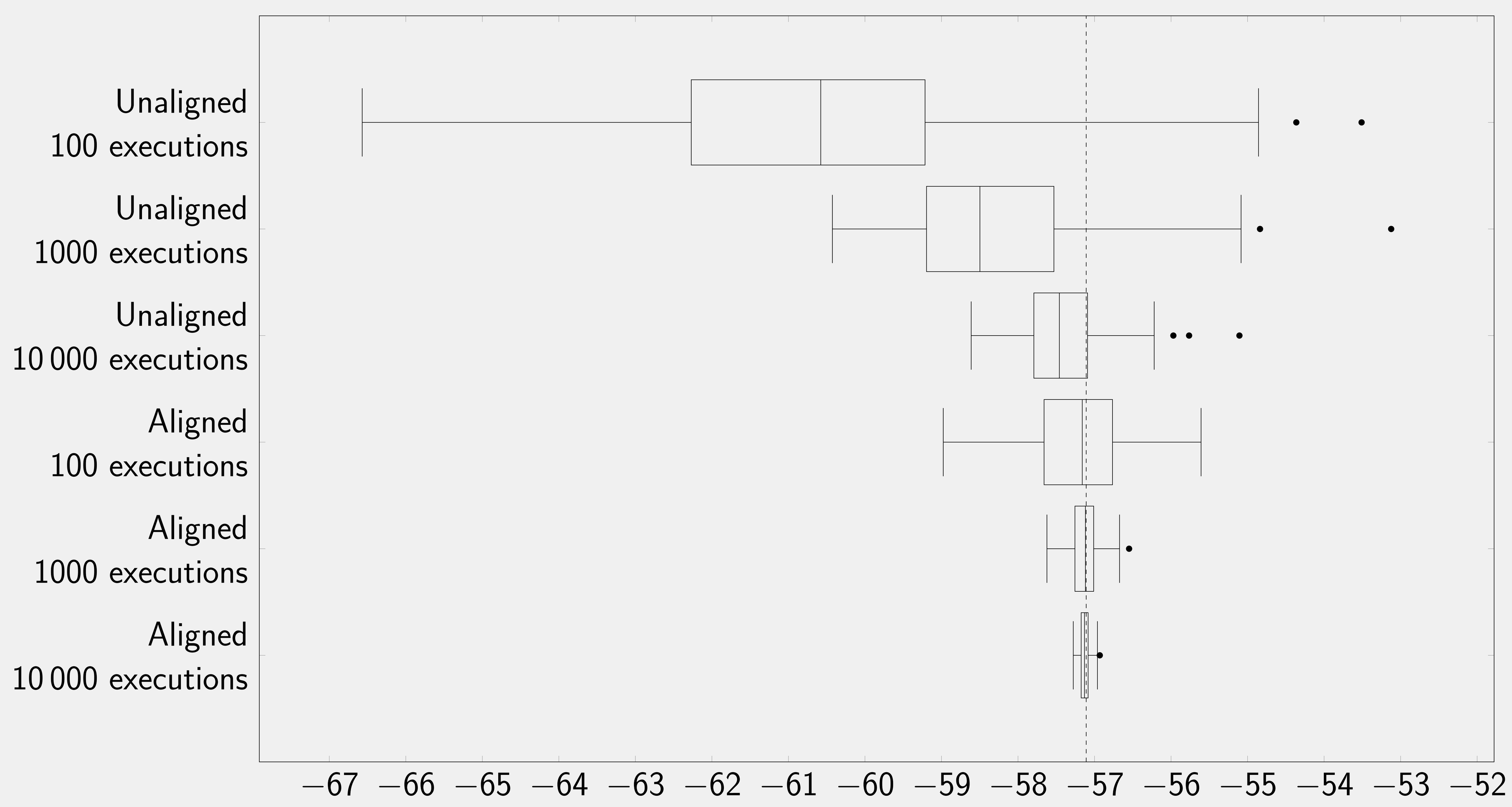
3. **Solve the constraints** using the standard 0-CFA algorithm:

$$\begin{array}{lll} S_y = \varnothing & S_x = \{(\lambda y.\cdot^9)^{10}\} & S_1 = \varnothing\\ S_2 = \{\textbf{stoch}\} & S_3 = \{(\lambda y.\cdot^9)^{10}\} & S_4 = \varnothing\\ S_5 = \varnothing & S_6 = \varnothing & S_7 = \varnothing\\ S_8 = \{(\lambda x.\cdot^7)^8\} & S_9 = \varnothing & S_{10} = \{(\lambda y.\cdot^9)^{10}\}\\ S_{11} = \varnothing. & & \end{array}$$

4. **Find all dynamic (i.e., not static) terms**: $\{3, 4, 5, 6, 9, 10\}$
5. **Transform** all dynamic `weight` calls to `dweight` calls

Case study: Estimating a normalizing constant

- We have evaluated aligned SMC on a model from phylogenetics for estimating a normalizing constant, with significant decrease in runtime and increase in accuracy compared to unaligned SMC. The dashed line is the true normalizing constant, and the numbers on the left show the number of SMC particles.



This project is financially supported by the Swedish Foundation for Strategic Research (ASSEMBLE RIT15-0012).