Implementations of PPL via Transformational Compilation

Wingate, Stuhlmuller, Goodman
Presented by: Lakhan Kamireddy

Abstract

GOAL of the paper: To provide a tractable (meaning controllable) technique to convert any programming language to a probabilistic programming language.

How: By "naming" the random choices in the program and maintaining a database of return values for random choices to implement an MCMC inference engine. By controlling the values in the database, execution traces of program are controlled - allows us to construct key operations for MH Algorithm.

Results: They showed examples to use the technique on both a functional PL and an imperative PL. Created a new language called Stochastic Matlab (Imperative PL), and Bher (Functional PL)

Setup

(Defn.) Unconditioned Probabilistic Program: Parameterless function *f* with an arbitrary mix of stochastic and deterministic elements. Function <=> Program interchangeably

Stochastic elements of *f* must come from a set of known, fixed Elementary Random Primitives or ERPs. ERPs are building blocks. For e.g in Matlab, ERPs may be *rand*, *randn*

Each type $t \in \tau$ (τ is set of ERP types) is a parametric family of distributions $p_t(x|\Theta_t)$, where Θ_t are parameters of distribution

Gaussian-Gamma mixture model

```
for i=1:1000
  if (rand > 0.5)
    X(i) = randn;
  else
    X(i) = gammarnd;
  end;
end;
```

E.g.Stochastic Matlab program

- A total of 2000 random choices will be made in f

- Let $f_{k|x1,\dots,xk-1}$ be k^{th} ERP encountered while executing f, let x_k be value it returns. (condit. on values of prev ERPs)

- We denote by *x*, all of the random choices made by f, meaning f defines the probability distribution *p*(*x*).

- $p(x) = \prod^{K}_{k=1} p_{tk}(x_k | \Theta_{tk}, x_1, \dots, x_{k-1})$

Reason abt posterior cond distr

Posterior conditional distribution: $p(x_{\backslash c}|x_c)$

Possible inference methods: 1) Rejection Sampling

But it is *intractable*! How to control the execution of *f* ? We have a better way!

Proposed Approach:

1. Give each f_k a "name" : need not be unique, can depend on previous x_k's or on program state

2. Rewrite source code of *f* to generate *f'* , replacing random functions f_k with deterministic functions f_k'. f_k' *s* use their name to look up current value x_k in database and return. If not found, they sample x_k, store it in database & return.

MCMC in Trace Space

We control execution trace of *f'* by manipulating the values in the database. We can do **MCMC** inference by being able to make proposals, score them, accept/ reject them.

- **Our MCMC Algorithm**: Define database **D** as mapping N -> τ x X x L x Θ

N: name of random choice, $\boldsymbol{\tau}$ is ERP type, **X**: return value, $\boldsymbol{\Theta}$: ERP parameters, **L**: random value's likelihood.

Let us say we implemented a ***trace_update(D)*** procedure like this - When a return value is not found in D, the value is sampled from appropriate ERP $x \sim p_{tc}(.| \Theta_c)$, its likelihood is computed, corresponding entry in database is updated.

MCMC in Trace Space

Given a current trace *x* and score *p(x)*, we proceed by reconsidering one random choice x_k. Then equip -

Each ERP type with proposal kernel $K_t(x'|x,\Theta)$, use this to gen. proposals to x_k.

After the proposal we call trace_update to generate a new trace x', and compute its likelihood *p(x')*.

It is a product of any reused random choices, and any new randomness that was sampled. We get overall score, which is MH accept ratio.

$$\alpha = \min \left\{ 1, \frac{p(x')\, K_t\, (x|x',\Theta)}{p(x)\, K_t\, (x'|x,\Theta)} \right\}$$

Naming Random Variables

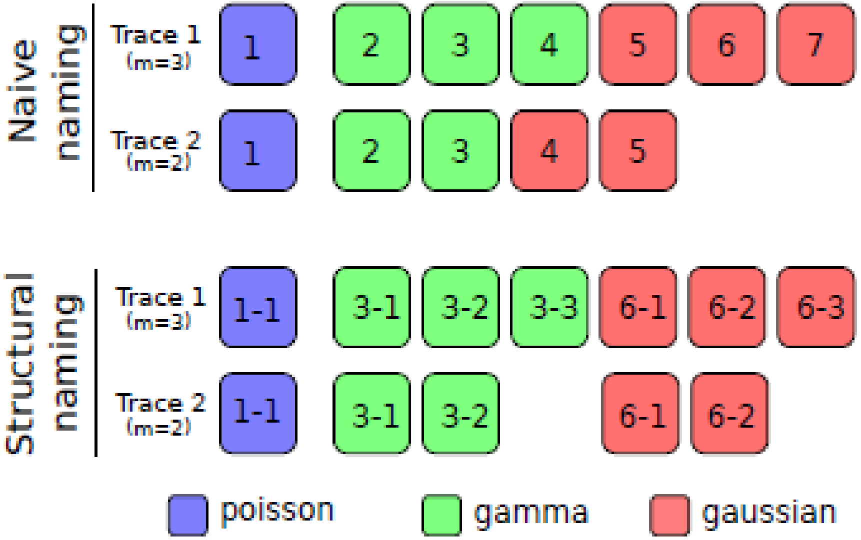
Naming scheme: chose wisely. Desirable properties ?

Problem with naive naming (sequential): Type mismatch at time 4 as well as general sequence misalignment.

Structural naming: Desired behavior, we reuse maximum no. of random choices.

Imperative: struc pos with an abstract stack trace augmented by comb of func ident, line num, loop iter num

Functional: struc pos with stack trace, func ident



Imperative Naming Specification

The **imperative** naming specification: The name of an f_k is the state of the tree stacks when f_k is encountered.

- Begin executing fk with empty **function**, **line** and **loop** stacks.
- When new function, push unique func. id on func stack, push 0 on line stack.
- When new line, increment last val on line stack
- When starting a loop, push 0 on loop stack
- When iterating a loop, incr last val on loop stack
- When exiting loop, pop loop stack
- When exiting func, pop func stack, line stack

Transformational Compilation

BHER:

Algorithm 5 A Church program that s
metric distribution.

```
(begin
  (define geometric
    (lambda (p)
      (if (flip p)
          1
          (+ 1 (geometric p)))))
  (geometric .7))
```

→

Algorithm 6 Transformed version of the Bher program shown in Algorithm 5. 'a1 to 'a4 are the names generated for function applications.

```
((lambda (addr)
   (begin
     (define geometric
       (lambda (addr p)
         (if (flip (cons 'a1 addr) p)
             1
             (+ (cons 'a2 addr)
                1
                (geometric (cons 'a3 addr) p)))))
     (geometric (cons 'a4 addr) 0.7)))
 '(top))
```