What is Probabilistic Programming?

CS

- Parameters
- Algorithm
- Output

ML

- $\theta$
- $p(X|\theta)$
- $X$

PP

- Parameters
- Algorithm
- Observations
Probabilistic Programming Goals

(i) Accelerate iteration over models
   - Code is easier to read and write than math
   - Lower technical barrier of entry to development of new models

(ii) Accelerate iteration over inference procedures
    - Computer language is an abstraction barrier
      - Inference procedures can be tested against a library of models
      - Inference procedures become “compiler optimizations”

(iii) Enable development of more expressive models
    - Probabilistic programs can express a superset of graphical models
    - Modern machine learning models are tens of lines of code
Anglican

“A Church of England Venture”

http://www.robots.ox.ac.uk/~fwood/anglican/
Venture* Syntax Subset

[assume symbol <expr>]
[observe <expr> <val>]
[predict <expr>]

Assume : variable declaration
Observe : data
Predict : printout

All <>’s are Scheme/Lisp expressions

(<func> <arg> ... <arg>)

* http://probcomp.csail.mit.edu/venture/
Probabilistic programs are constrained generative models with uncertainty represented by random variables that are assigned values by stochastic procedure calls

“Running” a probabilistic program outputs samples of plausible random variable assignments
Anglican Expressivity

- Programmimg language
  - Wide collection of built-in stochastic procedures
- Turing complete
  - Functions are first class objects
  - (eval <expr>) and (apply <func> ...) supported
- Recursion
- Memoisation
- Model family
  - All computable generative models

```lisp
[assume fib (lambda (n)
  (cond ((= n 0) 1) ((= n 1) 1)
    (else (+ (fib (- n 1)) (fib (- n 2))))))]
[assume r (poisson 4)]
[assume l (if (< 4 r) 6 (+ (fib (* 3 r)) (poisson 4))))]
[observe (poisson 1) 6]
[predict r]
```
Anglican Program : Hidden Markov Model

```lisp
[assume initial-state-dist (list (/ 1 3) (/ 1 3) (/ 1 3))]
[assume get-state-transition-dist (lambda (s)
  (cond ((= s 0) (list .1 .5 .4)) ((= s 1) (list .2 .2 .6))
       ((= s 2) (list .15 .15 .7))))]
[assume transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state)))]
[assume get-state (mem (lambda (index))
  (if (<= index 0) (discrete initial-state-dist)
   (transition (get-state (- index 1)))))]
[assume get-state-observation-mean (lambda (s)
  (cond ((= s 0) -1) ((= s 1) 1) ((= s 2) 0)))]
[observe (normal (get-state-obs-mean (get-state 1)) 1) .9]
[observe (normal (get-state-obs-mean (get-state 2)) 1) .8]
...
[observe (normal (get-state-obs-mean (get-state 16)) 1) -1]
[predict (get-state 0)]
[predict (get-state 1)]
...
[predict (get-state 16)]
```
Anglican: Particle MCMC Inference

Wood, van de Meent, Mansinghka “A new approach to probabilistic programming inference.” AISTATS, 2014

Wingate et al “Lightweight implementations of probabilistic programming languages via transformational compilation” AISTATS, 2011
Making Anglican Fast

- Generative models written in C, same inference algorithms
- Compiles to parallel code, ideal for multi-core CPUs
- Approximately 100x faster

Rapid Model Prototyping Case Study

- Decode sleep state from EEG
  - Solution: explicit duration HMM*
    - Traditional approach to model specification and inference
      - >700 lines (Python, https://github.com/mikedewar/EDHMM)
      - Months of development and debugging
    - Anglican
      - ~100 lines
      - Few days of development

Expressivity Case Study: Probabilistic Program Synthesis

1. Sample program from meta-generative model
2. Run program(s) with summary statistic observations
3. Approximate maxima in space of programs

Inference
Meta-Program for Prob. Prog. Synthesis

[ASSUME productions ...]
[ASSUME expression (list `lambda `(…) (productions …))]
[ASSUME my-sampler (eval expression)]

[OBSERVE (mean (apply-n (my-sampler 5.7 3.5) 100)) 5.7]
[OBSERVE (std (apply-n (my-sampler 5.7 3.5) 100)) 3.5]
[OBSERVE (kurt (apply-n (my-sampler 5.7 3.5) 100)) 0.0]
[OBSERVE (skew (apply-n (my-sampler 5.7 3.5) 100)) 0.0]

[PREDICT expression]
or
[PREDICT (apply-n (my-sampler -3.5 7.2) 100)]
Synthesis of Probabilistic Programs

Production rules for one meta-generator of samplers

[ASSUME productions
  (lambda (mysymbol level)
    (if (= mysymbol 'expr-real)
      ((lambda (output)
          (if (= output 0)
            (list 'get-real-constant (get-real-constant-id))
            (if (= output 1)
              (nth (list 'a 'b) (categorical (list 0.5 0.5)))
              (if (= output 2)
                (list (nth (list 'uniform-continuous (quote +) (quote *) (quote safe-div)) (categorical (list 0.25 0.25 0.25 0.25)))
                  (productions 'expr-real (+ level 1)) (productions 'expr-real (+ level 1)))
                (if (= output 3)
                  (list (nth (list 'sqrt 'safe-log) (categorical (list 0.5 0.5))) (productions 'expr-real (+ level 1)))
                (if (= output 4)
                  (list 'if (list (quote <) 'stack-id 10)
                    (list 'my-sampler (productions 'expr-real (+ level 1))
                      (productions 'expr-real (+ level 1)) (list (quote +) 'stack-id 1)) 0.0)
                    (list 'if (productions 'expr-bool (+ level 1)) (productions 'expr-real (+ level 1))
                      (productions 'expr-real (+ level 1))))))))
  )
  (categorical
    ((lambda (a b)
        (list a a b b b b b)
      ) (/ (- 1 (power 0.75 level)) 2) (/ (power 0.75 level) 4))))
  (if (= mysymbol 'expr-bool)
    (list (quote <) (productions 'expr-real (+ level 1)) (productions 'expr-real (+ level 1))))进程中]
Synthesis of Probabilistic Programs

Sampled samplers (i.e. probabilistic generative programs)

[ASSUME my-sampler
 (lambda (a b stack-id)
   (uniform-continuous
    (get-real-constant 1)
    (if (< (if (< (safe-log (get-real-constant 2)) b)
            (get-real-constant 1) (get-real-constant 3)) a)
     (safe-log (if (< (get-real-constant 3) (get-real-constant 1))
                 (get-real-constant 1) a))
     b))))]

[ASSUME my-sampler
 (lambda (a b stack-id)
   (safe-sqrt (uniform-continuous a (get-real-constant 1))))]
Synthesis of Probabilistic Programs

Samples from sampled samplers
Conclusion

- New paradigm in probabilistic modeling
  - Automatic inference
  - Rapid model prototyping
  - Completely new models

- Collaborations welcome

- Thanks to the team and sponsors
Synthesis of Probabilistic Programs

Samples of generative models (i.e. probabilistic programs)

[ASSUME my-sampler
  (lambda (a b stack-id)
    (+ (uniform-continuous
        (if (< b (get-real-constant 1))
          a
          (safe-div
            (get-real-constant 2)
            (if (< (get-real-constant 2) b)
              a
              (if (< (get-real-constant 2) (get-real-constant 2))
                (get-real-constant 2)
                (safe-log b)))))))
    (if (< (get-real-constant 2) (safe-div b a))
      (safe-sqrt (safe-sqrt
                   (if (< (get-real-constant 2) (get-real-constant 2))
                     (get-real-constant 1)
                     (get-real-constant 2))))
      (if (< a (get-real-constant 2)) (get-real-constant 3) a))) b)])
[ASSUME my-sampler
  (lambda (a b stack-id)
    (uniform-continuous
      b
      (uniform-continuous
        (if (< (if (< b (if (< a (get-real-constant 1))
              b (get-real-constant 2)) (sqrt a) b) a)
          (if (< stack-id 10)
              (my-sampler (get-real-constant 3)
                (+ a (get-real-constant 2)) (+ stack-id 1))
          0.0)
          (if (< (sqrt b) (get-real-constant 4)) (if (< a b)
              (* (get-real-constant 5) b)
              (get-real-constant 4)) b))
          (safe-div
            (get-real-constant 1)
            (if (< a b) (get-real-constant 4) (safe-log b)))))]]
Statistically efficient inference

HMM - Single-site MH

HMM - PMCMC

forward-backward
that single-threaded implementations of PMCMC suffer from the necessity of
before emitting batched simulations equal to the number of particles (here 100)
(note that PMCMC requires completing a number of what should be achievable via parallelism. Carefully
wall clock time; the apply axes report the number of it is the number of particles multiplied by the number
over 25 runs with different styles.

In Figures 1a-d there are three panels that report simulations. 1) and 2) illustrate our claims; 3) and 4) are in-
programs encode models with very few free parameters. 1) and 2) illustrate our claims; 3) and 4) are in-
ical model. It was designed to test for correctness of
branching with deterministic recursion program that
structure of Gaussians with fixed hyperparameters (DP
learning an uncollapsed Dirichlet process (DP) mix-
HMM) with continuous observations (HMM: Pro-

The vertical red line in the apply plot indicates the
mean with the largest number of states. The vertical green line indicates the
number of states for the first

The DP mixture program corresponds to a clustering

```
[assume class-generator (crp 1.72)]
[assume class (mem (lambda (n) (class-generator)))]
[assume var (mem (lambda (c) (* 10 (/ 1 (gamma 1 10)))))]
[assume mean (mem (lambda (c) (normal 0 (var c))))]
[assume u (lambda ()) (list (class 1) (class 2) ...
   (class 9) (class 10))]
[assume K (lambda ()) (count (unique (u))))]
[assume means (lambda (i c)
   (if (= i c) (list (mean c))
       (cons (mean i) (means (+ i 1) c)))]
[assume stds (lambda (i c)
   (if (= i c) (list (sqrt (* 10 (var c))))
       (cons (var i) (stds (+ i 1) c)))]
[observe (normal (mean (class 1)) (var (class 1))) 1.0]
[observe (normal (mean (class 2)) (var (class 2))) 1.1]
...
[observe (normal (mean (class 10)) (var (class 10))) 0]
[predict (u)]
[predict (K)]
[predict (means 1 (K))]
[predict (stds 1 (K))]
...
Figure 1: Comparative conditional measure test performance: PMCMC with 100 particles vs. RDB.

They are also how we compare different probabilistic programming inference engines.

5 Inference Engine Comparison

We compare PMCMC to RDB measuring convergence rates for an illustrative set of conditional measure test programs. Results from four such tests are shown in Figure 1 where the same program is interpreted using both inference engines. PMCMC is found to converge faster for conditional measure test programs that correspond to expressive probabilistic graphical models with rich conditional dependencies.

DP Mixture Results
Marsaglia Code

[assume (marsaglia-normal mean var)]
(beg)
(define x (uniform-continuous -1.0 1.0))
(define y (uniform-continuous -1.0 1.0))
(define s (+ (* x x) (* y y)))
(if (< s 1)
(+ mean (* (sqrt var)
(* x (sqrt (* -2 (/ (log s) s)))))))
(marsaglia-normal mean var)))

[assume sigma-squared 2]
[assume mu (marsaglia-normal 1 5)]
[observe (normal mu sigma-squared) 9]
[observe (normal mu sigma-squared) 8]
[predict mu]
·
They are also how we compare different probabilistic programming inference engines.

5 Inference Engine Comparison

We compare PMCMC to RDB measuring convergence rates for an illustrative set of conditional measure test programs. Results from four such tests are shown in Figure 1 where the same program is interpreted using both inference engines. PMCMC is found to converge faster for conditional measure test programs that correspond to expressive probabilistic graphical models with rich conditional dependencies.

The four test programs are: 1) a program that corresponds to state estimation in a hidden Markov model.
Branching Code

```lisp
[assume fib (lambda (n)
    (cond ((= n 0) 1) ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2)))))))
[assume r (poisson 4)]
[assume l (if (< 4 r) 6 (+ (fib (* 3 r)) (poisson 4)))]
[observe (poisson l) 6]
[predict r]
```

The branching program has no corresponding graphical model. It was designed to test for correctness of inference in programs with control logic and execution paths that can vary in random procedure call cardinality. It also illustrates mixing in a model where, as shown in the fourth plot, there is a large mismatch between the prior and the posterior. Because there is only one observation and just a single named random variable PMCMC and RDB should and does achieve essentially indistinguishable performance normalized to simulation, time and apply count.
Figure 1: Comparative conditional measure test performance: PMCMC with 100 particles vs. RDB.

They are also how we compare different probabilistic programming inference engines.

5 Inference Engine Comparison

We compare PMCMC to RDB measuring convergence rates for an illustrative set of conditional measure test programs. Results from four such tests are shown in Figure 1 where the same program is interpreted using both inference engines. PMCMC is found to converge faster for conditional measure test programs that correspond to expressive probabilistic graphical models with rich conditional dependencies.
Opportunity

Inference optimization by program reordering

```
[assume K (lambda () (count (unique (u))))]
[assume means (lambda (i c)
  (if (= i c) (list (mean c))
    (cons (mean i) (means (+ i 1) c)))
)
]
[assume stds (lambda (i c)
  (if (= i c) (list (sqrt (* 10 (var c))))
    (cons (var i) (stds (+ i 1) c)))
)
]
[observe (normal (mean (class 1)) (var (class 1))) 1.0
]
[observe (normal (mean (class 2)) (var (class 2))) 1.1
]
...
[observe (normal (mean (class 10)) (var (class 10))) 0
]
[predict (u)
]
[predict (K)
]
[predict (means 1 (K))
]
[predict (stds 1 (K))
]

The DP mixture program corresponds to a clustering with unknown mean and variance problem modelled via a Dirichlet process mixture of one-dimensional Gaussians with unknown mean and variance (normal-gamma priors). The KL divergence reported is between the running sample estimate of the distribution over the number of clusters in the data and the ground truth distribution over the same. The ground truth distribution over the number of clusters was computed for this model and data by exhaustively enumerating all partitions of the data (1.0, 1.1, 1.2, -10, -15, -20, .01, .1, .05, 0), analytically computing evidence terms by exploiting conjugacy, and conditioning on partition cardinality. The fourth plot shows the posterior distribution over the number of classes in the data computed by both methods relative to the ground truth.

This program was written in a way that was intentionally antagonistic to PMCMC in that the continuous class likelihood parameters were not marginalized out and the `observe` statements were not organized in an optimal ordering. Despite this, PMCMC outperforms RDB per simulation, wall clock time, and apply count as well.

5.3 Branching

```
[assume fib (lambda (n)
  (cond ((= n 0) 1) ((= n 1) 1)
    (else (+ (fib (- n 1)) (fib (- n 2))))))
]
[assume r (poisson 4)
]
[assume l (if (< 4 r) 6 (+ (fib (* 3 r)) (poisson 4)))
]
[observe (poisson l) 6
]
[predict r
]
...
```

The branching program has no corresponding graphical model. It was designed to test for correctness of inference in programs with control logic and execution paths that can vary in random procedure call cardinality. It also illustrates mixing in a model where, as shown in the fourth plot, there is a large mismatch between the prior and the posterior. Because there is only one observation and just a single named random variable PMCMC and RDB should and does achieve essentially indistinguishable performance normalized to simulation, time and apply count.

5.4 Marsaglia

```
[assume (marsaglia-normal mean var)
  (begin
    (define x (uniform-continuous -1.0 1.0))
    (define y (uniform-continuous -1.0 1.0))
    (define s (+ (* x x) (* y y)))
    (if (< s 1)
      (+ mean (* (sqrt var) (* x (sqrt (* -2 (/ (log s) s))))))
      (marsaglia-normal mean var)))
]
[assume sigma-squared 2
]
[assume mu (marsaglia-normal 1 5)
]
[observe (normal mu sigma-squared) 9
]
[observe (normal mu sigma-squared) 8
]
[predict mu
]
...
```

Marsaglia is a test program included here for completeness. It is an example of a type of program for which PMCMC sometimes may not be more efficient. Marsaglia is the name given to the rejection form of the Box-Muller algorithm for sampling from a Gaussian. The Marsaglia test program corresponds to an inference problem in which observed quantities are drawn from a Gaussian with unknown mean and this unknown mean is generated by an Anglican implementation of the Marsaglia algorithm for sampling from a Gaussian. The KS axis is a Kolmogorov-Smirnov test statistic computed by finding the maximum deviation between the accumulating sample and analytically derived ground truth cumulative distribution functions (CDF). Equal-cost PMCMC, RDB, and ground truth CDFs are shown in the fourth plot.

Because Marsaglia is a recursive rejection sampler it may require many recursive calls to itself. We conjecture that RDB may be faster than PMCMC here because, while PMCMC pays no statistical cost, it does pay a computational cost for exploring program traces that include many random procedure calls that lead to rejections whereas RDB, due to the implicit geometric prior on program trace length, effectively avoids paying excess computational costs deriving from unnecessarily long traces.

```
(a) HMM
(b) DP Mixture
```

Figure 2: Effect of program line permutations
Another Opportunity

Massive parallelism

(a) HMM

(b) DP Mixture