A New Approach to Probabilistic Programming Inference

Frank Wood
fwood@robots.ox.ac.uk
http://www.robots.ox.ac.uk/~fwood
Avances In Scalable Bayesian Computation
March 3-7, 2014 Banff
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/

Wood, van de Meent, Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Venture*-Inspired Syntax and Semantics

```plaintext
[assume symbol <expr>]
[observe <expr> <const>]
[predict <expr>]
```

Assume : variable declaration
Observe : data
Predict : printout

All „’s are Scheme/Lisp expressions

```
(<proc> <arg> ... <arg>)
```

* http://probcomp.csail.mit.edu/venture/*
Anglican Interpretation

```
[assume sigma-squared 2]
[assume mu (marsaglia-normal 1 5)]
[observe (normal mu sigma-squared) 9]
[observe (normal mu sigma-squared) 8]
[predict mu]
```

```
mu, 4.579108417224186
mu, 5.84552255327749
mu, 4.579108417224186
mu, 5.84552255327749
mu, 5.518797656065681
mu, 5.518797656065681
mu, 4.579108417224186
mu, 5.84552255327749
mu, 5.0603250422963155
mu, 5.518797656065681
mu, 6.359401703719326
mu, 4.714582404953012
mu, 6.359401703719326
```

Probabilistic programs are constrained generative models with uncertainty represented by random variables that are assigned values by stochastic procedure calls.

“Running” an Anglican program outputs “predict”-ed posterior samples of random variable assignments conditioned on observed data.
Anglican Expressivity

- Programming language
  - Wide collection of built-in stochastic procedures
  - Turing complete
    - Procedures are first class objects
    - \((\text{eval } <\text{expr}>\) and \((\text{apply } <\text{proc}>) \ldots\) 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 l) 6)
(predict r)
```
Related Work

- STAN [Stan Dev. Team, 2013]
- Infer.NET [Minka, Winn et al, 2010]
- IBAL [Pfeffer, 2001]
- BLOG [Milch et al, 2004]
  - Random Database [Wingate, Stuhlmüller et al, 2011]
- Venture* [Mansinghka, et al, in prep]
  - Anglican [Wood, van de Meent, Mansinghka, 2014]

* http://probcomp.csail.mit.edu/venture/
Execution Trace Preliminaries

- “Exploring” the space of execution traces = inference

- Different program traces arise from application of Stochastic Procedures (SPs)
  
  \[ \text{flip, normal, discrete, poisson, dirichlet, gamma, …} \]

- An execution trace is uniquely determined by return values for all SP applications

- Observations weight execution traces
Execution Trace Joint

- Observes
  - Likelihood terms
    \[ \tilde{p}(y, x) \equiv \prod_{n=1}^{N} p(y_n | \theta_{t_n}, x_n) \tilde{p}(x_n | x_{n-1}) \]

- Assumes
  - Generate sequences of stochastic procedure applications
    \[ \tilde{p}(x_n | x_{n-1}) \]
    \[ = \prod_{k=1}^{\left| x_n \setminus x_{n-1} \right|} p(x_{n,k} | \theta_{t_n,k}, x_{n,1:k-1}, x_{n-1}) \].

- Observed value
- Parameter of observation distribution
- Type of observation distribution
- Interpreter memory state
- Parameter of stochastic procedure
- Type of stochastic procedure

Manuscript under review by AISTATS 2014
MCMC / Random DB Review

Sample from joint constrained by observes

$$\tilde{p}(x|y) \propto \tilde{p}(y, x)$$

Joint is a fixed syntactically allowable reordering of conditionals

$$\tilde{p}(y, x) \equiv \tilde{p}(x_1)\tilde{p}(x_2|x_1)\cdots \tilde{p}(x_n|x_{n-1})p(y_1|x_1)\cdots p(y_n|x_n)$$

Metropolis-Hastings

$$\min \left( 1, \frac{p(y|x')p(x')q(x|x')}{p(y|x)p(x)q(x'|x)} \right)$$

Single stochastic procedure (SP) output

$$q(x'|x) = \frac{\kappa(x'_{m,j}|x_{m,j})}{p(x'|x)} \frac{p(x'|x'|\cap x)}{p(x'_{m,j}|x'\cap x)}$$

Probability of new part of proposed execution trace

Number of SP’s in original trace

Probability of new SP return value (sample) given trace prefix
Proposal distribution == the prior

\[ \kappa(x'_{m,j} | x_{m,j}) = p(x'_{m,j} | x' \cap x) \]

Full MH acceptance ratio

Number of SP applications in original trace

\[ \frac{p(y | x') p(x') | x}{p(y | x) p(x) | x'} \]

Number of SP applications in new trace

\[ \frac{p(x' | x \cap x')}{p(x' | x \cap x)} \]

Probability of generating proposal trace continuation given current trace beginning

Probability of regenerating current trace continuation given proposal trace beginning

- Random Database [Wingate, Stuhlmüller et al, 2011]
MCMC / Random DB Criticism

• Proposing from the prior is suboptimal
• Computation is wasted
  • Short-circuiting -> MIT Venture
• MH is local*
SMC for Prob. Prog. Inference

Inference Goal
• Given samples (interpreter memory states)
  \[ x_{n-1}^{(\ell)} \sim \tilde{p}(x_{n-1}|y_{1:(n-1)}) \]
• sample from (find program traces that reflect the next observe).
  \[ \tilde{p}(x_n|y_{1:n}) \]

SMC / SIR Approach
• Propose from the prior = interpret the program up to next observe
  \[ q(\hat{x}_n^{(\ell)}|x_{n-1}^{(\ell)}, y_{1:n}) \equiv p(\hat{x}_n^{(\ell)}|x_{n-1}^{(\ell)}) \]
• Calc. importance weights (weight by observe outer application likelihood)
  \[ \hat{w}_n^{(\ell)} = p(y_n|\hat{x}_n^{(\ell)}) \]
  and resample (new interpreter memory states)
  \[ x_n^{(\ell)} \sim \sum_{\ell} \hat{w}_n^{(\ell)} \delta_{\hat{x}_n^{(\ell)}} \]
  \[ w_n^{(\ell)} = \hat{w}_n^{(\ell)} / \sum_{j} \hat{w}_n^{(j)} \]
Anglican = PMCMC for Prob. Prog.

- Example: Particle Gibbs
  - MH w/ accept prob. = 1
  - “Retained particle” $x^*$
- Non-local
  - Potentially changes many variable values at once

[Holenstein 2009; Andrieu, Doucet, Holenstein 2010; etc]

### Algorithm 1 PMCMC for Prob. Prog. Inference

```
\begin{algorithm}
L ← number of particles
S ← number of sweeps
\{\bar{w}_N^{(l)}, \bar{x}_N^{(l)}\} ← Run SMC
for s < S do
    \{\cdot, \bar{x}_N^{(s)}\} ← r(1, \{1/L, \bar{x}_N^{(l)}\})
    \{\cdot, \bar{x}_0^{(s)}\} ← initialize L − 1 interpreters
    for d ∈ ordered lines of program do
        for l < L − 1 do
            \bar{x}_n^{(l)} ← fork(\bar{x}_n^{(l)})
        end for
        if directive(d) == “assume” then
            for l < L − 1 do
                \bar{x}_n^{(l)} ← interpret(d, \bar{x}_n^{(l)})
            end for
            \{\bar{x}_n^{(l)}\} ← \{\bar{x}_n^{(l)}\} ∪ \bar{x}_n^{*}
        else if directive(d) == “predict” then
            interpret(d, \bar{x}_n^{(l)})
        end for
        interpret(d, \bar{x}_n^{*})
    else if directive(d) == “observe” then
        for l < L − 1 do
            \{\bar{w}_n^{(l)}, \bar{x}_n^{(l)}\} ← interpret(d, \bar{x}_n^{(l)})
        end for
        \mathcal{T} ← r(L − 1, \{\bar{w}_n^{(l)}, \bar{x}_n^{*}\} ∪ \{\bar{w}_n^{*}, \bar{x}_n^{*}\})
        \{\bar{w}_n^{(l)}, \bar{x}_n^{(l)}\} ← \mathcal{T} ∪ \{\bar{w}_n^{*}, \bar{x}_n^{*}\}
    end if
end for
end for
\end{algorithm}
```
Example 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

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.

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

HMM - Single-site MH

HMM - PMCMC

HMM - Single-site MH

HMM - PMCMC

forward-backward
Compiled PMCMC – Probabilistic-C

```c
#include "probabilistic.h"
#define K 3
#define N 11

/* Markov transition matrix */
static double T[K][K] = { { 0.1, 0.5, 0.4 },
                        { 0.2, 0.2, 0.6 },
                        { 0.15, 0.15, 0.7 } };

/* Observed data */
static double data[N] = { nan, .9, .8, .7, 0, -.025,
                       -5, -2, -.1, 0, 0.13 };

/* Prior distribution on initial state */
static double initial_state[K] = { 1.0/3, 1.0/3, 1.0/3 };

/* Per-state mean of Gaussian emission distribution */
static double state_mean[K] = { -1, 1, 0 };

/* Generative program for a HMM */
int main(int argc, char **argv) {
    int states[N];
    for (int n=0; n<N; i++) {
        states[n] = (n==0) ? discrete_rng(initial_state, K)
                       : discrete_rng(T[states[n-1]], K);
        if (n > 0) {
            observe(normal_lnp(data[n], state_mean[states[n]], 1));
        }
        printf("state[%d],%d\n", n, states[n]);
    }
    return 0;
}
```

Paige et al “A Compilation Target for Probabilistic Programming Languages.” in submission, 2014
Compiled PMCMC $\approx 100 \times$ Speedup

- HMM 10-states, 50 observations
- CRP 10 observation mixture of 1-D Gaussian

Compiled MH - https://github.com/dritchie/probabilistic-js
Compiled PMCMC Algorithm Performance

- What’s the right inference algorithm for the model?
- What’s the right performance measure?
Time to produce 10,000 samples running probabilistic-C HMM code on multi-core EC2 instances with identical processor type while varying number of particles (bars). Both more cores and more particles eventually degrade performance suggesting possible operating system optimizations.
Jack-Up Units

Keppel FELS

Maersk

Keppel FELS

Slide from Houlsby
Jack-up operations

Float to site
Lower legs
Light ship load
Preload
Dump preload
Climb to air-gap and operate
Storm

sketches after Poulos (1988)
Slide from Houlsby
Forward Spudcan Simulation / Inference

- Deterministic simulation
  - ~750 lines of C code
- Stochastic simulation
  - ~900 lines of C code
- Inverse simulation
  - +15 lines of C code
- ~1000 samples / second
Conclusion

- Forward inference methods for probabilistic programming
  - Easy to understand and implement
    - Requires only POSIX abstraction
  - Automatic parallel scalability
  - No new language skills required
    - Little new programming in some cases

- Thanks to the team

collaborators

van de Meent
Perov
Paige

Mansinghka

and sponsors
DP Mixture Code

[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
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

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

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 [3] for sampling from a Gaussian [7]. 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 [6] 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.

5.5 Line Permutation

Syntactically and semantically `observe` and `predict`'s are mutually exchangeable (so too are `assume`'s up to...
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
; Branching Code

(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)
```

---

5.3 Branching

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.

5.4 Marsaglia

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 \[3\] for sampling from a Gaussian \[7\]. 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 \[6\] 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

5.5 Line Permutation

Syntactically and semantically `observe` and `predict`'s are mutually exchangeable (so too are `assume`'s up to
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.

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

Inference optimization by program reordering

The DP mixture program corresponds to a clustering with unknown mean and variance problem modeled 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

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

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 [3] for sampling from a Gaussian [7]. 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 [6] 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.
Another Opportunity

Parallelism

(a) HMM

(b) DP Mixture

Figure 3 shows how varying the number of particles in the PMCMC inference engine affects performance. Perfor-
Expressivity Case Study: Probabilistic Program Synthesis

Sample program from meta-generative model

Run program(s) with summary statistic observations

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
Synthesis of Probabilistic Programs

Sampled samplers (i.e. probabilistic generative programs)

```scheme
[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))))]
```
Samples from sampled samplers