# A Lambda-Calculus Foundation for Universal Probabilistic Programming

Johannes Borgström[1], Ugo Dal Lago[2],
Andrew D. Gordon[3,4], and Marcin Szymczak[4]

[1] Uppsala University
[2] University of Bologna
[3] Microsoft Research
[4] University of Edinburgh

**Abstract.** We develop the operational semantics of a probabilistic $\lambda$-calculus with continuous distributions, as a foundation for universal probabilistic programming languages such as Church, Anglican, and Venture. Our first contribution is to adapt the classic operational semantics of $\lambda$-calculus to the continuous case, via creating a measure space on terms and defining step-indexed approximations. We prove equivalence of big-step and small-step formulations of this *distributional semantics*. Our second contribution is to formalize the implementation technique of trace MCMC for our calculus and to show correctness. A key step is defining a *sampling semantics* of a term as a function from a trace of random samples to a value, and showing that the distribution induced by integrating over all traces equals the distributional semantics. Another step is defining sufficient conditions for the distribution induced by trace MCMC to converge to the distributional semantics. To the best of our knowledge, this is the first rigorous correctness proof for trace MCMC for languages in the Church family.

## 1 Semantics for Universal Probabilistic Programming

Church [4] introduced *universal probabilistic programming*, the idea of writing probabilistic models for machine learning in a Turing-complete functional programming language. Church, and its descendants Venture [9], Anglican [11], and Web Church [6] are dialects of Scheme. Another example of universal probabilistic programming is webppl [5], a probabilistic interpretation of JavaScript.

The following is the general form of a probabilistic query in Church:

```
(query (define x1 e1) ... (define xn en) eq (condition ec))
```

The result of the query is a representation of the distribution given by the probabilistic expression $e_q$, given variables $x_i$ defined by potentially probabilistic expressions $e_i$, conditioned by the binary predicate $e_c$.

For example, the following query defines a probability `p` at random, defines a function to flip a coin with bias `p`, conditions the model on single observations of `0` and `1`, and returns a representation of the distribution of `p`. We use calls `(rnd)` to sample a probability from the uniform distribution on the unit interval.

```
(query
  (define p (rnd))
  (define flip (lambda (x) (< (rnd) p)))
  p
  (condition
    (and (eq (flip) 0) (eq (flip) 1)))))
```

The first problem we address in this work is to provide an operational semantics for universal probabilistic programming languages. Our example illustrates the common situation in machine learning that models are based on continuous distributions (such as `(rnd)`), but previous work on operational semantics for probabilistic $\lambda$-calculi are based on discrete distributions.

We introduce a call-by-value $\lambda$-calculus with primitives for random draws from various continuous distributions (such as `(rnd)`), and exceptions to represent conditioning. We describe a measure space of $\lambda$-terms and let $\mathscr{D}$ range over *value distributions*, that is, measures on values of the $\lambda$-calculus. We define step-indexed operational semantics, in both small-step ($M \to_n \mathscr{D}$) and big-step ($M \Downarrow_n \mathscr{D}$) styles, which we prove equivalent, and enable us to define the *distributional semantics* $\mathscr{D} = [\![M]\!]$ for each closed term $M$.

## 2 Semantics and Correctness of Trace MCMC

The original implementation of Church introduced the implementation technique *trace MCMC* [4]. A closed $\lambda$-term $M$ has a *trace* $s$ if there is a run of $M$ making a finite sequence of random choices $s$, which yields a result $V(s)$ functionally dependent on $s$. In our example, each trace has form $s = [p, q_1, q_2]$ where $p$ is the bias of the coin, and each binary flip $b_i$ is true if and only if $q_i < p$.

We consider trace MCMC as an instance of the Metropolis-Hastings (MH) algorithm. Given a closed term $M$, trace MCMC generates a Markov chain of traces, with a stationary distribution on traces that induces a distribution over values corresponding to the semantics of $M$. The algorithm is parametric in a target distribution $\mathscr{D}$ and a proposal kernel $Q$, that takes any trace $s$ of $M$ to a probability distribution over traces, corresponding to a perturbation of $s$.

We formalize the algorithm rigorously, defining the target distribution on program traces and the proposal kernel as Lebesgue integrals of corresponding density functions. We prove these functions measurable with respect to the $\sigma$-algebra on program traces—a step usually omitted in similar developments.

To define the target distribution, we give a deterministic *sampling semantics* for our calculus based on the explicit consumption of a program trace $s$ of random draws and production of an explicit weight $w$ for each outcome. We formulate the sampling semantics in big-step style ($M \Downarrow^s_w V$) and small-step style ($(M, w, s) \to (M', w', s')$) and prove equivalence. Moreover, we prove that the value distributions induced by the sampling and distributional semantics are indeed the same. We exploit this equivalence to show that, subject to sufficient aperiodicity and irreducibility conditions on the transition kernel induced by $Q$ and the acceptance ratio, the distribution on values induced by trace MCMC on $M$ converges to the distributional semantics $\mathscr{D} = [\![M]\!]$.

## 3 Related Work (Partial)

To the best of our knowledge, the only previous theoretical justification for trace MCMC is the recent work by Hur and others [7], who show correctness of trace MCMC for the imperative probabilistic language R2 [10]. Their result does not apply to higher-order languages such as Church.

While giving a domain theory for probabilistic $\lambda$-calculi is known to be hard [8], there have been recent advances on, e.g., probabilistic coherent spaces [1, 3] and also game semantics [2], which in some cases are not only adequate, but also fully abstract. We don't see strong obstacles in applying all these to our calculus, but this remains outside the scope of this contribution.

The full version of this paper, in preparation, will include a full bibliography.

## References


[1] Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, 209(6):966–991, 2011.

[2] Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[3] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 309–320, 2014.

[4] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In David A. McAllester and Petri Myllymäki, editors, *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, pages 220–229. AUAI Press, 2008.

[5] Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014.

[6] Noah D. Goodman and J. B. Tenenbaum. Probabilistic Models of Cognition. http://probmods.org, 2014.

[7] Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, Selva Samuel, and Deepak Vijaykeerthy. Implementing a correct sampler for imperative probabilistic programs.

[8] C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 186–195, 1989.

[9] Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

[10] Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. R2: an efficient MCMC sampler for probabilistic programs. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2476–2482. AAAI Press, 2014.

[11] David Tolpin, Jan-Willem van de Meent, and Frank Wood. Probabilistic programming in anglican. In Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III*, volume 9286 of *Lecture Notes in Computer Science*, pages 308–311. Springer, 2015.