Some Formal Structures in Probability

Sam Staton
University of Oxford, UK

Abstract

This invited talk will discuss how developments in the Formal Structures for Computation and Deduction can also suggest new directions for the foundations of probability theory. I plan to focus on two aspects: abstraction, and laziness. I plan to highlight two challenges: higher-order random functions, and stochastic memoization.

2012 ACM Subject Classification Theory of computation → Program semantics; Mathematics of computing → Nonparametric statistics

Keywords and phrases Probabilistic programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2021.4

Category Invited Talk

Funding *Sam Staton*: Research supported by a Royal Society University Research Fellowship and the ERC BLAST grant.

Summary

Probabilistic programming is a popular tool for statistics and machine learning. The idea is to describe a probabilistic model as a program with random choices. The program might be a simulation of some system, such as a physics model, a model of viral spread, or a model of electoral behaviour. We can now carry out statistical inference over the system by running a Monte Carlo simulation – running the simulation 100,000's of times. The key observation of probabilistic programming is that we can actually run this same probabilistic program with different advanced simulation methods, instead of a naive Monte Carlo simulation, such as a Hamiltonian Monte Carlo simulation or Variational Inference, without changing the program. See [20] for an overview.

Part of the *practical* appeal of probabilistic programming is this separation between probabilistic models and inference algorithms. But this also has a *foundational* appeal: if we can understand probabilistic models as programs, then the foundations of probability and statistics can be discussed in terms of program semantics. This might take the lead of denotational semantics, by interpreting programs in terms of traditional measure theory (e.g. [17, 18]). But there is also a chance of new foundational perspectives on probability by following other semantic methods, such as equational reasoning, rewriting, or categorical axiomatics (see also [1, 4]).

This programming-based foundation for probability is attractive because there are some intuitively simple probabilistic scenarios which have an easy programming implementation but for which a plain measure-theoretic interpretation seems impossible. I now highlight some issues in abstraction and laziness.

Abstraction. Abstraction is a crucial concept in probability: statistics arise by abstracting away information. At a higher level, we have argued that de Finetti's theorem, a fundamental theorem in probability, can be understood in terms of abstract data types [15], and so too generalizations [8, 16].

Function types are a key abstraction in programming theory, but are less well understood in probability. For example, write $\Pr(X)$ for the space of probability distributions on X, and consider the functional

$$\mathsf{piecewise} : \Pr(\mathbb{R} \to \mathbb{R}) \to \Pr(\mathbb{R} \to \mathbb{R})$$

which converts a random function to a random piecewise version of it (see Figure 1). It is easy to define in a few lines of code: $\mathsf{piecewise}(f)$ will draw a random partition of the x-axis, draw random functions from f for each part, and splice them together. But although statistics and probability make plenty of use of random piecewise linear functions, random piecewise constant functions, and so on, the piecewise functional itself has no direct interpretation in traditional measure theory. This has led to some recent semantic developments (e.g. [6, 3, 13, 2, 7, 14]).

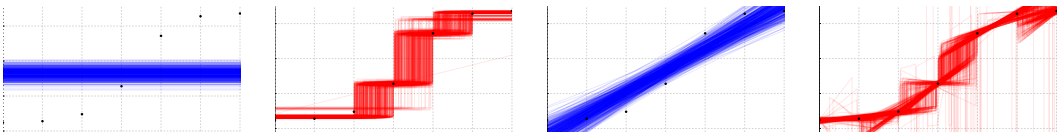

Figure 1 Bayesian regression for the data set indicated by black dots. The regression was done using constant, piecewise constant, linear, and piecewise linear functions respectively. The piecewise functional was used to program the random piecewise functions.

Laziness. Laziness in programming is a counterpart to the notion of "process" which is fundamental in probability. This has long been understood [5, 9, 10], and I recently explored more aspects of laziness in the prototype LazyPPL [19]. For example, a "stick breaking" process randomly divides the unit interval into an infinite number of parts, each part representing a different cluster of some data. If this is computed lazily, it always terminates, because the data is finite (Figure 2(a)).

One outstanding problem is a semantic interpretation of stochastic memoization. In the non-probabilistic setting, memoization is a program optimization, where we are lazy about re-evaluating a function at a given argument, by caching or tabling. But in the probabilistic setting it gives new semantic possibilities. Stochastic memoization is a functional

$$\mathsf{memoize} : (X \to \Pr(Y)) \to \Pr(X \to Y)$$

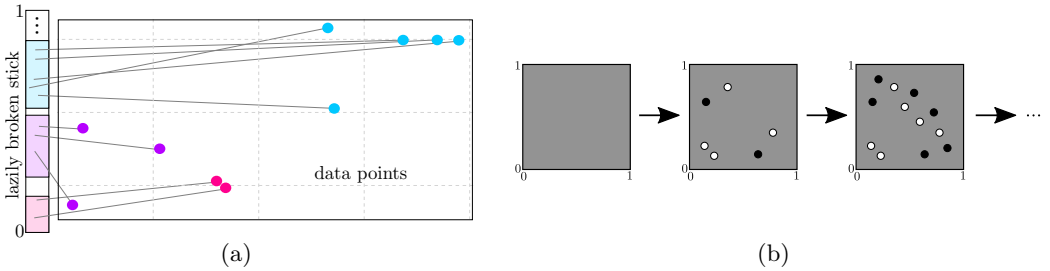

Figure 2 (a) Stick-breaking: To group the data points (right) into an unknown number of clusters, we randomly divide the unit interval "stick" into an infinite partition (left), and then assign a cluster to each data point by randomly picking a number in $[0, 1]$ for each point (lines from the data points to the stick). In practice, this is done lazily. (b) Lazily building the adjacency matrix of an uncountable random graph, as a memoized random function $[0, 1]^2 \to \mathsf{bool}$.

which converts a family $f : X \to \Pr(Y)$ of probability distributions into a distribution on functions $X \to Y$, by sampling $f(x)$ once for every x. When X is infinite, this is impossible to do eagerly, but it is no problem lazily. For example, consider a function $g : [0,1]^2 \to \Pr(\mathsf{bool})$ where $g(x,y)$ is the Bernoulli distribution (a coin flip); then $\mathsf{memoize}(g) : \Pr([0,1]^2 \to \mathsf{bool})$ is the random adjacency matrix of a random uncountable graph (Figure 2(b)). More generally, g is a "graphon" (e.g. [12]). This also generalizes clustering by stick-breaking, because clusters can be regarded as connected components of graphs.

This memoize functional is easy to implement. It appears in several languages [5, 11, 19], and is practically useful in random graphs, probabilistic logic [11], clustering [5, §2.1], and natural language modelling [21]. But there remains a big open problem:

▶ **Open problem.** *To find a denotational model for a language with stochastic memoization.*

I will discuss some recent progress on this problem, based on ongoing work with Swaraj Dash, Younesse Kaddar, Hugo Paquet, and others.

References

1 Kenta Cho and B. Jacobs. Disintegration and Bayesian inversion via string diagrams. *Math. Struct. Comput. Sci.*, 29:938–971, 2019.
2 Fredrik Dahlqvist and Dexter Kozen. Semantics of higher-order probabilistic programs with conditioning. In *Proc. POPL 2020*, 2020.
3 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. In *Proc. POPL 2018*, 2018.
4 T. Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Adv. Math.*, 370, 2020.
5 N. D. Goodman, V. K. Mansinghka, et al. Church: a language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2008.
6 Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *Proc. LICS 2017*, 2017.
7 Xiaodong Jia, Bert Lindenhovius, Michael W. Mislove, and Vladimir Zamdzhiev. Commutative monads for probabilistic programming languages. In *Proc. LICS 2021*, 2021.
8 P. Jung, J. Lee, S. Staton, and H. Yang. A generalization of hierarchical exchangeability on trees to directed acyclic graphs. *Annales Henri Lebesgue*, 4, 2021.
9 Oleg Kiselyov and Chung-chieh Shan. Embedded probabilistic programming. In *Proc. DSL 2009*, 2009.
10 Daphne Koller, David McAllester, and Avi Pfeffer. Effective Bayesian inference for stochastic programs. In *Proc. AAAI 1997*, 1997.
11 Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In *Introduction to Statistical Relational Learning*. MIT Press, 2007.
12 Peter Orbanz and Daniel M. Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(2):437–461, 2015.
13 Hugo Paquet and Glynn Winskel. Continuous probability distributions in concurrent games. In *Proc. MFPS 2018*, pages 321–344, 2018.
14 Marcin Sabok, Sam Staton, Dario Stein, and Michael Wolman. Probabilistic programming semantics for name generation. In *Proc. POPL 2021*, 2021.
15 S. Staton, D. Stein, H. Yang, L. Ackerman, C. E. Freer, and D. M. Roy. The Beta-Bernoulli process and algebraic effects. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2018.
16 S. Staton, H. Yang, N. L.. Ackerman, C. Freer, and D. Roy. Exchangeable random process and data abstraction. In *PPS 2017*, 2017.

17 Sam Staton. Commutative semantics for probabilistic programming. In *Proc. ESOP 2017*, 2017.

18 Sam Staton. Probabilistic programs as measures. In *Foundations of Probabilistic Programming*. CUP, 2020.

19 Sam Staton. LazyPPL, 2021. URL: `https://bitbucket.org/samstaton/lazyppl/src/`.

20 Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming, 2018. `arXiv:1809.10756`.

21 Frank D. Wood, Cédric Archambeau, Jan Gasthaus, Lancelot James, and Yee Whye Teh. A stochastic memoizer for sequence data. In *Proc. ICML 2009*, 2009.