Some Semantic Issues in Probabilistic Programming Languages

Hongseok Yang
School of Computing, KAIST, South Korea
hongseok.yang@kaist.ac.kr

Abstract

This is a slightly extended abstract of my talk at FSCD'19 about probabilistic programming and a few semantic issues on it. The main purpose of this abstract is to provide keywords and references on the work mentioned in my talk, and help interested audience to do follow-up study.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Program semantics; Theory of computation → Denotational semantics; Mathematics of computing → Bayesian nonparametric models; Mathematics of computing → Bayesian computation

Keywords and phrases Probabilistic Programming, Denotational Semantics, Non-differentiable Models, Bayesian Nonparametrics, Exchangeability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.4

Category Invited Talk

1 Introduction

Probabilistic programming [11, 9, 35, 8] refers to the idea of developing a programming language for writing and reasoning about probabilistic models from machine learning and statistics. Such a language comes with the implementation of several generic inference algorithms that answer various queries about the models written in the language, such as posterior inference and marginalisation. By providing these algorithms, a probabilistic programming language enables data scientists to focus on designing good models based on their domain knowledge, instead of building effective inference engines for their models, a task that typically requires expertise in machine learning, statistics and systems. Even experts in machine learning and statistics may benefit from such a probabilistic programming system because using the system they can easily explore highly advanced models. Several probabilistic programming languages have been built. Good examples are Stan [8], PyMC [23], Church [9], Venture [19], Anglican [35, 30], Turing [7], Pyro [3], Edward [32, 31], ProbTorch [26] and Hakaru [20].

In the past five years, with colleagues from programming languages, machine learning and probability theory, I have worked on developing the semantic foundations, efficient inference algorithms, and static program analysis for such probabilistic programming languages, especially those that support expressive language features such as higher-order functions, continuous distributions and general recursion. At FSCD'19, I plan to talk about some of these projects related to semantics and the lessons that I learnt.

This document is a companion to my FSCD'19 talk. Its primary goal is to provide keywords and references to the work mentioned in the talk, and help interested people start their follow-up study. If the reader looks for systematic introduction to probabilistic programming, I recommend to look at books [10, 34, 6] and teaching materials on probabilistic programming instead.

Listing 1 Anglican program with undefined posterior. (`normal_pdf x` 0 1) computes the density of the standard normal distribution at x. The program defines a model with two random variables x and y, the former being sampled from the standard normal distribution and the latter from the exponential distribution. The random variable y is observed to have the value 0, and the program expresses the posterior of x under this observation.

```
(let [x      (sample (normal 0 1))
      x_pdf  (normal_pdf x 0 1)
      y      (observe (exponential (/ 1 x_prob)) 0)]
  x)
```

2 Some research questions

A large part of my research has been about building a solid theoretical foundation for probabilistic programming languages. Doing so is particularly needed for such languages, because programs in those languages are run by inference engines but these inference engines only approximate the ideal mathematical semantics of these programs, namely, their posterior distributions. Even worse, sometimes probabilistic programs do not have posterior distributions at all, and I do not know of any inference engines that can detect it. Listing 1 shows one such program in Anglican, whose posterior distribution (or more precisely posterior density) is undefined.[1] Also, I found this foundation building intellectually rewarding, because, as I will explain shortly, it made me think about unexpected connections among multiple disciplines and revisit old concepts in programming languages, such as data abstraction, from a new perspective.

Here are three specific research questions about the foundation for probabilistic programming that intrigued me and my colleagues.

Q1: How to define a good denotational semantics for higher-order probabilistic programming languages with continuous distributions and general recursions?

A standard tool for defining a continuous probability distribution rigorously is measure theory. But it turns out that measure theory is not good enough for defining the denotational semantics of higher-order probabilistic programming languages, such as Church, Venture and Anglican, because measure theory does not support higher-order functions well. The category of measurable spaces is not Cartesian closed, because the set of measurable functions $[\mathbb{R} \to_m \mathbb{R}]$ cannot be turned into a measurable space that makes the following evaluation map measurable [1]:

$$ev : [\mathbb{R} \to_m \mathbb{R}] \times \mathbb{R} \to \mathbb{R}, \qquad ev(f, r) = f(r).$$

I have been involved in the joint efforts to address this semantic issue [29, 12, 25]. Using tools from category theory, we developed a theory of quasi-Borel spaces [12], which extends measure theory, and used our theory to define the denotational semantics of higher-order probabilistic programming languages and to prove the correctness of inference algorithms for

[1] Let $p_n(x, 0, 1)$ be the density at x of the normal distribution with mean 0 and standard deviation 1, and $p_\lambda(y)$ be the density of the exponential distribution with rate λ. The prior of the program in Listing 1 is $p_n(x, 0, 1)$, and the likelihood is the density of the $1/p_n(x, 0, 1)$-rate exponential distribution at $y = 0$, which is $p_{1/p_n(x,0,1)}(y = 0) = 1/p_n(x, 0, 1)$. Thus, the joint density is 1. The marginal likelihood is ∞. Thus, the posterior density $p(x|y = 0)$ is undefined.

Listing 2 Anglican program with non-differentiable density. The program denotes a model with three random variables x_1, x_2, y, all three being drawn from the normal distribution with different parameters. It expresses the posterior distribution of x_1 under the condition that y has the value 4.

```
(let [x1 (sample (normal 0 1))
      x2 (sample (normal (* x1 x1) 1))
      x3 (if (> x2 0) x1 x2)
      y  (observe (normal x3 1) 4)]
  x1)
```

such languages [25]. Recently, Vákár, Kammar and Staton built a domain theory on top of quasi-Borel spaces, and showed how to handle term and type recursions in denotational semantics in the presence of continuous probability distributions [33].

An interesting future direction is to generalise well-known results from probability theory using the theory of quasi-Borel spaces. Our initial investigation with de Finetti's theorem for exchangeable random sequences shows a promise [12].

Q2: Can a probabilistic program denote a distribution with a density that is not differentiable at some non-measure-zero set?

This question assumes a typical setting that gradient-based inference algorithms operate. In the setting, all sampling statements in probabilistic programs use distributions on $\mathbb{R}^n$ for some n that have densities with respect to the Lebesgue measure, and those probabilistic programs mean distributions on traces of sampled values during execution. For instance, the posterior distribution of the program in Listing 2 has the following density $f : \mathbb{R}^2 \to [0, \infty)$ with respect to the Lebesgue measure on $\mathbb{R}^2$: for $x_1, x_2 \in \mathbb{R}$,

$$f(x_1, x_2) = p_n(x_1, 0, 1) \cdot p_n(x_2, x_1^2, 1) \cdot (\mathbb{1}_{[x_2>0]} \cdot p_n(4, x_1, 1) + \mathbb{1}_{[x_2 \leq 0]} \cdot p_n(4, x_2, 1))$$

where $p_n(x, m, \sigma)$ is the density at x of the normal distribution with mean m and standard deviation σ and $\mathbb{1}_{[\varphi]}$ is the indicator function returning 1 if φ holds and 0 otherwise.

The negative answer to the question is needed in order for these gradient-based inference algorithms to work correctly [21, 18]. Intuitively, it ensures that the algorithms never attempt to compute gradients at non-differentiable points, and the effects of these non-differentiable points to the algorithms can be estimated algorithmically.

Currently we have only a partial answer to the question. We proved that for a first-order probabilistic programming language without loops, if a program in the language uses only analytic operations as its primitive operations, the set of its non-differentiable points has measure zero [36]. The question is open for a language that supports higher-order functions, includes loops, or permits non-analytic primitive operations.

Q3: What is a good theory of data abstraction for probabilistic programming languages, which in particular can let us analyse modules from Bayesian nonparametrics?

Sophisticated probabilistic models from Bayesian nonparametrics are implemented as modules in some probabilistic programming languages, such as Church and Anglican [24, 30]. This question asks for extending the theory of data abstraction to account for such modules. Ideally, the theory should provide guidance about how to implement such modules, and help programmers understand the consequences of using these modules.

I became interested in the question because of an intriguing feature of these modules. They are often implemented using impure features, such as mutable state, but they still satisfy a type of equations that typically hold for pure modules, such as commutativity of module operations. It turns out that this phenomenon is not an accident; the probabilistic models that the modules denote satisfy symmetry properties such as exchangeability and contractibility [22, 16], and the equations for the modules come from these properties [28, 27].

Answering the question amounts to connecting the theory of data abstraction in programming languages to the study on these symmetry properties in probability theory. So far we found a connection for the Beta-Bernoulli process, one of the simplest models [27], and inspired by this connection, we defined a new type of symmetry properties for probabilistic models and proved a representation theorem for them [15]. These results are far from answering the question posed, and I expect (and hope) that more deep results are waiting to be discovered.

The three questions are chosen mainly based on my personal taste. If the reader wants to gain a broad view on what semantics researchers care about regarding probabilistic programming languages, I recommend to read the following papers [17, 14, 13, 4, 2, 5].

3 Final remark

Probabilistic programming is an exciting topic that raises several fresh theoretical and practical questions in programming languages, statistics, machine learning and probability theory. I hope that my FCSD'19 talk and (highly incomplete and biased) list of research questions in the previous section helps the reader partially understand why I and my colleagues are excited about the topic. This document conveys the view of one programming-language researcher on probabilistic programming. To understand how machine learning researchers think about probabilistic programming, I recommend to watch the video recording of Josh Tenenbaum's ICML'18 invited talk, and read the introduction of the book on probabilistic programming [34]; the introduction is written mostly by Frank Wood.

References

1 R. J. Aumann. Borel structures for function spaces. *Illinois Journal of Mathematics*, 5:614–630, 1961.

2 Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio V. Russo. Deriving Probability Density Functions from Probabilistic Functional Programs. *Logical Methods in Computer Science*, 13(2), 2017.

3 Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 20:28:1–28:6, 2019.

4 J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. van Gael. Measure Transformer Semantics for Bayesian Machine Learning. *LMCS*, 9(3):11, 2013.

5 Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. In *ICFP*, pages 33–46, 2016.

6 Cameron Davidson-Pilon. *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*. Addison-Wesley Professional, 2015.

7 Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: Composable inference for probabilistic programming. In *AISTATS*, pages 1682–1690, 2018.

8 Andrew Gelman, Daniel Lee, and Jiqiang Guo. Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization. *Journal of Educational and Behavioral Statistics*, 40(5):530–543, 2015.

9 Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A Language for Generative Models. In *UAI*, pages 220–229, 2008.

10 Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2019-4-11.

11 Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014*, pages 167–181, 2014.

12 Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *LICS*, pages 1–12, 2017.

13 Daniel Huang and Greg Morrisett. An Application of Computable Distributions to the Semantics of Probabilistic Programming Languages. In *ESOP*, pages 337–363, 2016.

14 Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. A Provably Correct Sampler for Probabilistic Programs. In *FSTTCS*, pages 475–488, 2015.

15 Paul Jung, Jiho Lee, Sam Staton, and Hongseok Yang. A Generalization of Hierarchical Exchangeability on Trees to Directed Acyclic Graphs. *arXiv preprint*, 2018. `arXiv:1812.06282`.

16 Olav Kallenberg. *Probabilistic Symmetries and Invariance Principles.* Springer, 2005.

17 D. Kozen. Semantics of Probablistic Programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

18 Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization Gradient for Non-differentiable Models. In *NeurIPS*, pages 5558–5568, 2018.

19 Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint*, 2014. `arXiv:1404.0099`.

20 Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *Proceedings of the 13th International Symposium on Functional and Logic Programming, FLOPS 2016*, pages 62–79, 2016.

21 Akihiko Nishimura, David Dunson, and Jianfeng Lu. Discontinuous Hamiltonian Monte Carlo for Sampling Discrete Parameters. *arXiv preprint*, 2017. `arXiv:1705.08510`.

22 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.

23 Anand Patil, David Huard, and Christopher J Fonnesbeck. PyMC: Bayesian Stochastic Modelling in Python. *Journal of Statistical Software*, 35(4):1, 2010.

24 Daniel M. Roy, Vikash Mansinghka, Noah Goodman, and Joshua Tenenbaum. A stochastic programming perspective on nonparametric Bayes. In *ICML Workshop on Nonparametric Bayesian*, 2008.

25 Adam Scibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. Denotational validation of higher-order Bayesian inference. *PACMPL*, 2(POPL):60:1–60:29, 2018.

26 N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *NIPS*, pages 5927–5937, 2017.

27 Sam Staton, Dario Stein, Hongseok Yang, Nathanael L. Ackerman, Cameron Freer, and Daniel M Roy. The Beta-Bernoulli Process and Algebraic Effects. In *ICALP*, 2018.

28 Sam Staton, Hongseok Yang, Nathanael L. Ackerman, Cameron Freer, and Daniel M Roy. Exchangeable random process and data abstraction. In *Workshop on Probabilistic Programming Semantics, PPS 2017*, 2017.

29 Sam Staton, Hongseok Yang, Chris Heunen, Ohad Kammar, and Frank Wood. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *LICS*, pages 525–534, 2016.

30 David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank D. Wood. Design and Implementation of Probabilistic Programming Language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages, IFL 2016*, pages 6:1–6:12, 2016.

31 Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, and Alexey Radul. Simple, Distributed, and Accelerated Probabilistic Programming. In *NeurIPS*, pages 7609–7620, 2018.

32 Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja R. Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *CoRR*, abs/1610.09787, 2016.

33 Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.

34 Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *arXiv preprint*, 2018. `arXiv:1809.10756`.

35 Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. In *AISTATS*, pages 1024–1032, 2014.

36 Yuan Zhou, Bradley Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. A Low-Level Probabilistic Programming Language for Non-Differentiable Models. In *AISTATS*, 2019.