Probabilistic programs, computability, and de Finetti measures

Daniel M. Roy
Massachusetts Institute of Technology
droy@csail.mit.edu

Cameron E. Freer
Massachusetts Institute of Technology
freer@math.mit.edu

The complexity of probabilistic models, especially those involving recursion, has far exceeded the representational capacity of graphical models. Functional programming languages with probabilistic choice operators have recently been proposed as universal representations for statistical modeling (e.g., IBAL [Pfe01], $\lambda_\circ$ [PPT08], Church [GMR+08]). The conditional independence structure of a probabilistic program is not, in general, representable by a graphical model. Rather, it is dynamic and is given by the random control and data flow of the program. These functional probabilistic languages are allied with imperative probabilistic languages (e.g., Infer.NET) and a similar tradition of augmenting logical representations with probabilistic quantifiers (e.g., BLOG [MMR+05], iBLOG [MMG08], PRISM [SK97], BLP [KR07], SLP [Mug96], Markov Logic [RD06], Independent Choice Logic [Poo08]).

In Church, a probabilistic dialect of Scheme/LISP, probability distributions are represented as procedures which generate samples. However, while sampling defines the semantics, implementations need not be sampling-based. Posterior analysis is implemented by conditioning on a program's output and querying the values of variables internal to the program (e.g., a variable which represents the mixture component for a particular data point in a mixture model).

The MIT-Church implementation [GMR+08] provides two general inference algorithms: an exact but arbitrarily slow rejection sampler for generating perfect samples, and an approximate Metropolis-Hastings algorithm which performs a random walk over the space of possible computational histories of the constrained program. In contrast, [Rad07] gives a deterministic inference algorithm for probabilistic Scheme.

Although the semantics of probabilistic programs have been studied extensively in theoretical computer science in the context of randomized algorithms (e.g., [Koz81] and [JP89]), this application of probabilistic programs to universal statistical modeling has a different character which has raised a number of interesting theoretical questions (e.g., [RP02], [PPT08], and [GMR+08]). Some preliminary connections between exchangeability, nonparametrics, and computability were described in [RMGT08]. Here we describe a recent theoretical result [FR09] on computability, exchangeability and de Finetti measures, and highlight its consequences for the semantics of probabilistic programs and for statistical inference.

1 The de Finetti Theorem

Let $X = \{X_i\}_{i\geq 1}$ be an infinite sequence of real random variables. An infinite sequence X is *exchangeable* if, for any finite set $\{k_1, \ldots, k_j\}$ of distinct indices, $(X_{k_1}, \ldots, X_{k_j}) \stackrel{d}{=} (X_1, \ldots, X_j)$, where $\stackrel{d}{=}$ denotes equality in distribution. We denote the indicator function of a set B by $\mathbf{1}_B$.

Theorem (de Finetti [Kal05, Chap. 1.1]). *Let $\{X_i\}_{i\geq 1}$ be a exchangeable sequence of real-valued random variables. There is a random probability measure ν on $\mathbf{R}$ such that $\{X_i\}_{i\geq 1}$ is conditionally i.i.d. with respect to ν. Moreover, ν is given, almost surely, by $\nu(B) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_B(X_i)$, where B ranges over the Borel subsets of $\mathbf{R}$.* □

Following [Ald85], we call ν the *directing random measure* and call its distribution the *de Finetti measure*.

A Bayesian perspective suggests the following interpretation: exchangeable sequences arise from independent observations from a latent random distribution. Posterior analysis requires a prior distribution on this unknown distribution. This has been justified by the existence of de Finetti measures. A natural question to ask is whether *computable* exchangeable sequences arise from independent observations from *computable* random distributions. Using notions of computability established in computable analysis [Sch07], it was recently shown [FR09] that this is always the case.

As an example, consider the Beta(a, b)-Bernoulli pro-

cess and the Pólya urn scheme written in Church. We will define the procedure `sample-coin` such that calling `sample-coin` returns a new procedure which itself returns random binary values. The probabilistic program

```
(define my-coin (sample-coin))
(my-coin)
(my-coin)
(my-coin)
...
```

defines a random binary sequence. Consider the following two implementations of `sample-coin` (and recall that the `(λ () ...)` special form creates a procedure of no arguments):

(i)
```
(define (sample-coin)
  (let ([coin-weight (beta a b)])
    (λ () (bernoulli coin-weight)) ) )
```

(ii)
```
(define (sample-coin)
  (let ([heads a]
        [total (+ a b)] )
    (λ () (let ([x (bernoulli heads/total)])
            (set! heads (+ heads x))
            (set! total (+ total 1))
            x ) ) ) )
```

In case *(i)*, evaluating `(my-coin)` returns a 1 with probability `coin-weight` and a 0 otherwise, where the shared `coin-weight` parameter is itself drawn from a Beta(a, b) distribution on $[0, 1]$. Note that the sequence of values obtained by evaluating `(my-coin)` is exchangeable but not i.i.d. (e.g., an initial sequence of ten 1's leads one to predict that the next draw is more likely to be 1 than 0). However, conditioned on the `coin-weight` (a hidden variable within the opaque procedure `my-coin`) the sequence is i.i.d.

The code in *(ii)* implements the Pólya urn scheme (see [dF75, Chap. 11.4]), and the sequence of return values is exchangeable because `x` depends only on the number of heads and tails, and not on the order of the previous samples. Furthermore, the Pólya urn scheme is known to produce the Beta-Bernoulli process.

Because the sequence induced by *(ii)* is exchangeable, de Finetti's theorem implies that it has the same distribution as the product of repeated draws from some random distribution. In fact, by the above characterization of the Beta-Bernoulli process, *(i)* and *(ii)* are equivalent as distributions over sequences.

However, there is an important difference between these two implementations: *(i)* denotes the de Finetti measure, while *(ii)* does not, as samples from it do not denote fixed distributions.

The state of a procedure `my-coin` sampled using *(ii)* changes after each iteration, as the sufficient statistics are updated (using `set!`). Therefore, each element of the sequence is generated from a different distribution. Even though the sequence of calls to such a `my-coin` has the same *marginal* distribution as those given by repeated calls to a `my-coin` sampled using *(i)*, a procedure `my-coin` sampled using *(ii)* denotes a probability kernel which depends on the state.

In contrast, a `my-coin` sampled using *(i)* does not modify itself via mutation (`set!`); the value of `coin-weight` does not change after it is randomly initialized and therefore `my-coin` denotes a fixed distribution — a particular Bernoulli. Its marginal distribution is a random Bernoulli, precisely the *directing random measure* of the Beta-Bernoulli process. The *de Finetti measure* is defined to be the distribution of this directing random measure, and so *(i)* denotes the de Finetti measure.

Mathematically, the relationship between *(i)* and *(ii)* is that *(ii)* is obtained from *(i)* by *marginalization* (in particular, integrating over the directing random measure). Thus, *exchangeable mutation* in probabilistic programs arises from *integrating out* variables, thereby inducing non-local dependencies.

2 Computable de Finetti Theorem

In the case of the exchangeable sequence given by the (manifestly) computable Pólya urn scheme, the directing random measure ν is a random Bernoulli whose parameter is drawn from a Beta distribution. The distribution of the latter is also computable. In general, it can be shown that computable exchangeable sequences always induce computable de Finetti measures. Furthermore, given code generating an exchangeable sequence (e.g., given *(ii)*), we can automatically generate code for the de Finetti measure (i.e., give a procedure of the form *(i)* which does not use mutation):

Theorem (Computable de Finetti, [FR09])**.** *Let X be a real-valued exchangeable sequence and let μ be the distribution of its directing random measure ν. Then X is computable iff μ is computable. Moreover, μ is uniformly computable in X, and conversely.* □

In many cases, exchangeable sequences are expressed as samples from a sequence of conditional distributions $P(X_{k+1} \mid X_1, \ldots, X_k)$. The computable de Finetti result implies that if the conditional distributions are computable (loosely, *sampleable*), then so is the de Finetti measure. Like the Pólya urn example, given a probabilistic program implementing the Chinese restaurant process, the random directing measure given by Sethuraman's [Set94] stick-breaking con-

struction of the Dirichlet process could be automatically recovered. These different representations may have vastly different time, space and entropy complexity; mutation becomes especially relevant when implementing parallel inference algorithms that span large networks where communication costs are high.

The use of mutation also has important theoretical consequences for the semantics of probabilistic languages. Procedures which do not use mutation denote *probabilistic functions* (i.e., probability kernels) which do not depend on the state. Procedures which use mutation will, in general, denote probabilistic functions which do depend on the state. However, if the sequence of repeated calls to such a procedure is exchangeable, the classical de Finetti theorem implies that repeated calls to the procedure have the same distribution as that of some probabilistic function (not depending on state), even though the denotational semantics are different.

In particular, calls to a procedure which uses exchangeable mutation are draws from *some* distribution. The computable de Finetti theorem implies that this distribution is itself computable, and provides a means of effectively recovering it. Thus one can move freely between representations with and without mutation, by transforming procedures which use exchangeable mutation into their de Finetti representations, which induce the same marginal distribution and do not use mutation. An open problem is to determine the complexity of this transformation between representations.

References

[Ald85] David J. Aldous, *Exchangeability and related topics*, École d'été de probabilités de Saint-Flour, XIII—1983, Lecture Notes in Math., vol. 1117, Springer, Berlin, 1985, pp. 1–198.

[dF75] Bruno de Finetti, *Theory of probability. Vol. 2*, John Wiley & Sons Ltd., London, 1975.

[FR09] Cameron E. Freer and Daniel M. Roy, *Computable exchangeable sequences have computable de Finetti measures*, Mathematical Theory and Computational Practice: Fifth Conference on Computability in Europe, CiE 2009 (Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, eds.), Lecture Notes in Computer Science, Springer, Berlin, 2009.

[GMR+08] Noah Goodman, Vikash Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua Tenenbaum, *Church: a language for generative models with non-parametric memoization and approximate inference*, Uncertainty in Artificial Intelligence, 2008.

[JP89] C. Jones and G. Plotkin, *A probabilistic powerdomain of evaluations*, Proc. of the Fourth Ann. Symp. on Logic in Comp. Sci., IEEE Press, 1989, pp. 186–195.

[Kal05] Olav Kallenberg, *Probabilistic symmetries and invariance principles*, Springer, New York, 2005.

[Koz81] Dexter Kozen, *Semantics of probabilistic programs*, J. Comp. System Sci. **22** (1981), no. 3, 328–350.

[KR07] K. Kersting and L. De Raedt, *Bayesian logic programming: Theory and tool*, An Introduction to Statistical Relational Learning (L. Getoor and B. Taskar, eds.), MIT Press, 2007.

[MMG08] D. McAllester, B. Milch, and N. D. Goodman, *Random-world semantics and syntactic independence for expressive languages*, Tech. Report MIT-CSAIL-TR-2008-025, Massachusetts Institute of Technology, 2008.

[MMR+05] B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, and A. Kolobov, *BLOG: Probabilistic models with unknown objects*, Proc. of Int. Joint Conf. on Artificial Intelligence, 2005.

[Mug96] S. Muggleton, *Stochastic logic programs*, Advances in Inductive Logic Programming (L. de Raedt, ed.), IOS Press, 1996, pp. 254–264.

[Pfe01] Avi Pfeffer, *IBAL: A probabilistic rational programming language*, Proc. of the Int. Joint Conf. on Artificial Intelligence, 2001, pp. 733–740.

[Poo08] David Poole, *The independent choice logic and beyond*, Probabilistic Inductive Logic Programming, 2008, pp. 222–243.

[PPT08] Sungwoo Park, Frank Pfenning, and Sebastian Thrun, *A probabilistic language based on sampling functions*, ACM Trans. Program. Lang. Syst. **31** (2008), no. 1, 1–46.

[Rad07] Alexey Radul, *Report on the probabilistic language Scheme*, Tech. Report MIT-CSAIL-TR-2007-059, Massachusetts Institute of Technology, 2007.

[RD06] M. Richardson and P. Domingos, *Markov logic networks*, Machine Learning **62** (2006), no. 1, 107–136.

[RMGT08] Daniel M. Roy, Vikash Mansinghka, Noah Goodman, and Josh Tenenbaum, *A stochastic programming perspective on nonparametric Bayes*, Nonparametric Bayesian Workshop, Int. Conf. on Machine Learning, 2008.

[RP02] Norman Ramsey and Avi Pfeffer, *Stochastic lambda calculus and monads of probability distributions*, Proc. of the 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (2002), 154–165.

[Sch07] Matthias Schröder, *Admissible representations of probability measures*, Electron. Notes Theor. Comput. Sci. **167** (2007), 61–78.

[Set94] J. Sethuraman, *A Constructive definition of Dirichlet priors*, Statistica Sinica **4** (1994), 639–650.

[SK97] T. Sato and Y. Kameya, *PRISM: A symbolic-statistical modeling language*, Proc. of Int. Joint Conf. on Artificial Intelligence, 1997.