EXCHANGEABLE RANDOM PRIMITIVES

NATHANAEL L. ACKERMAN[1], CAMERON E. FREER[2], AND DANIEL M. ROY[3]

ABSTRACT. Exchangeable random primitives (XRPs) are a crucial, but undocumented, class of constructs in the probabilistic programming language Church. We give a rigorous definition for the samplers of XRPs, and give a representation theorem characterizing their essential form and conditional independence structure. We place semantic questions surrounding exchangeable random primitives in the context of a line of research in programming languages involving purity and referential transparency.

1. Introduction

In the view of the authors, one of the key contributions of the probabilistic programming language Church is a class of constructs, called *exchangeable random primitives* (XRPs), that can be used to extend the language with stochastic primitives whose implementations use stateful computation. Surprisingly, XRPs are not mentioned in the paper introducing the language [GMRBT08] and are buried in the original implementation, dubbed MIT-Church.

Despite being reimplemented and generalized in several languages descending from Church, XRPs are poorly documented and, as a result, poorly understood. XRPs, as defined in MIT-Church, are composed of three parts: a *sampler*, an *unsampler*, and a *scorer*. Informally, a sampler takes as inputs one or more parameters, and returns an output, which may have been generated stochastically. Given any sequence of calls to the sampler, the scorer must produce a "log probability density" for all the input-output pairs, unless they have been "forgotten" (unsampled) by the unsampler. Finally, the input–output pairs must be "exchangeable". The statement closest to, but still not quite, a formal definition appears in Mansinghka, Selsam, and Perov [MSP14]:

> "The formal requirement is that the cumulative log probability density of any sequence of input-output pairs is invariant under permutation."

The three parts of an XRP may maintain shared state in order to meet the specification. See also the proposal that exchangeability provides a notion of purity for probabilistic programs in [RMGT08] and [Man09, §2.2.1], and the descriptions of XRPs in [Wu13, §4]. As described in [RMGT08], it is natural to use XRPs to represent various objects occurring in nonparametric Bayesian statistics. Support for other particular XRPs such as memoization and gensym appears in some dialects of Church, and also in Hansei [KS09]. Venture [MSP14] introduces *stochastic procedures* which include XRPs as a special case.

We give a rigorous definition for a sampler, leaving the definition of an unsampler and scorer to future work. Even focusing on the sampler, there is interesting mathematical structure to characterize and some clarity can be brought to the relationship between XRPs and exchangeable random structures. We also give a precise representation theorem that explains the way in which XRPs behave like ordinary procedures in Church.

While we believe the joint characterization of a sampler and unsampler is straightforward, there appears to be some intrinsic difficulties with the idea that a scorer reports a log probability density. In particular, on the natural product space of input–output pairs, one can define samplers such that there is no dominating product measure, and so the obvious notion of a probability density need not

[1] HARVARD UNIVERSITY; [2] GAMALON LABS; [3] UNIVERSITY OF TORONTO.
E-mail addresses: nate@math.harvard.edu, cameron.freer@gamalon.com, droy@utstat.toronto.edu.

```
for i in range(10):
    Sum[i] = 1.0; Total[i] = i+1

def next_draw(i):
    global Sum, Total
    y = bernoulli(Sum[i]/Total[i])
    Sum[i] += y; Total[i] += 1
    return y
```

FIGURE 1. Example of a random procedure whose outputs are exchangeable.

even exist. On the other hand, it holds trivially that the constant function 1 is a probability density with respect to the probability distribution of the input–output pairs themselves, yet every existing implementation of an XRP-based language fails to produce correct behavior if this scoring function is used. It is an open problem to resolve this issue, e.g., by giving rigorous sufficient conditions on the scorer in order to produce correct behavior in a reference implementation.

Functional programming languages [Lon99] and notions of purity [Sab98] have been explored primarily in the deterministic setting, with some extensions to nondeterministic programming languages in [SS92] and [FKS11]. XRPs might provide a basis for extending concepts such as referential transparency, definiteness, extensionality, and unfoldability [SS90] to the probabilistic setting. In particular, because the order of evaluation of a sequence of calls to an XRP does not affect the resulting joint distribution, XRPs constitute a more general class than the naive extension of functions to the probabilistic setting given by *random* probability kernels (whose repeated evaluations yield *conditionally* i.i.d. sequences), while still preserving some analogues of definiteness and extensionality.

2. EXAMPLE OF AN EXCHANGEABLE RANDOM PROCESSES USING STATE

Consider the example in Fig. 1. This Python code block describes an array of Pólya urn processes indexed by $0, \dots, 9$. In this example, the state space S consists of possible values of the variables `Sum` and `Total`, and the parameter space X is $\{0, \dots, 9\}$.

This example uses state in a nontrivial way, but there is always an alternative way of expressing the same distribution on return values without making use of state. (For this example, this furthermore has a nice form given by the beta-bernoulli process.)

3. FORMALIZING XRPS

Definition 3.1. Let X, Y, and S be spaces. A *stateful kernel from X to Y with state in S* is a family $P = (P_{x,s})_{(x,s)\in X\times S}$ of probability measures on $Y \times S$.

Let X, Y, and S be spaces, and let P be a stateful kernel from X to Y with state in S. For every initial state $s_0 \in S$ and sequence of inputs $x = (x_1, \dots, x_k) \in X^*$, let $Q_{s_0,x}$ be the law of the Markov chain $(s_n, y_n : n = 1, \dots, k)$ given by Let $Q'_{s_0,x}$ be the marginal law of the sequence $(y_1, \dots, y_k)$. The following definition captures the property that repeated sampling from a stateful kernel does not depend on the order of the evaluation.

Definition 3.2. A stateful sampler P is *functional* when, for every $k \in \mathbb{N}$, $s_0 \in S$, $x = (x_1, \dots, x_k) \in X^k$, and permutation σ of $[k] := \{1, \dots, k\}$, we have $Q'_{s_0,x} \circ (\bar{\sigma})^{-1} = Q'_{s_0,x\circ\sigma}$, i.e., the $\bar{\sigma}$-pushforward of the $Q_{s_0,x}$-distribution of the sequence $(y_1, \dots, y_k)$ is the same as the $Q_{s_0,x\circ\sigma}$-distribution of the sequence $(y_1, \dots, y_k)$.

Remark 3.3. When X is a one-point space, then P is functional if and only if $(y_1, \dots, y_k)$ is exchangeable in the ordinary sense of a finite sequence of random variables. ◁

Remark 3.4. When S is a one-point space, then P is functional if and only if for every sequence $x = (x_1, \dots, x_k) \in X^k$, the sequence $y = (y_1, \dots, y_k)$ consists of independent random variables such that for every constant subsequence of x, the corresponding subsequence of y is i.i.d. $\triangleleft$

The following is a standard result in probability theory.

Theorem 3.5 (Doob)**.** *Let $f : [0,1] \times X \to \mathcal{M}_1(Y)$. Then there exists a function $\mathrm{samp}(f) : [0,1] \times X \times [0,1] \to Y$ such that, for all $\alpha \in [0,1]$, we have $\mathrm{samp}(f)(\alpha, x, U) \sim f(\alpha, x)$ whenever U is uniformly distributed.*

Using this, we can now characterize functional, stateful kernels in terms of sampling.

Theorem 3.6. *Let X be countable and let P be a functional, stateful kernel from X to Y with state in S. Then there exists a function $f : [0,1] \times X \to \mathcal{M}_1(Y)$ such that, for every $x = (x_1, \dots, x_k) \in X^*$, the sequence*

$$(\mathrm{samp}(f)(U, x_1, U_1), \dots, \mathrm{samp}(f)(U, x_k, U_k)) \tag{1}$$

has distribution $Q'_{\varepsilon,x}$ when $U, U_1, \dots, U_k$ are i.i.d. uniform random variables.

There is an alternative view on this result, which we now state in terms of probability kernels.

Theorem 3.7. *Let X be countable and let P be a functional, stateful kernel from X to Y with state in S. Then there exists a function $f : [0,1] \times X \to \mathcal{M}_1(Y)$ such that, for every $x = (x_1, \dots, x_k) \in X^*$, the sequence of random variables $y_1, \dots, y_k$, defined to be conditionally independent given a uniform random variable U and distributed according to*

$$y_j \mid U \sim f(U, x_j), \quad \text{for } j \in [k], \tag{2}$$

has distribution $Q'_{\varepsilon,x}$.

Remark 3.8. For countable X (and some unstated but reasonable regularity on Y), the proof can be reduced to an application of de Finetti's theorem. To prove the theorem for a continuum-size space X, some care would be required. It seems that one would have to require that X be separable, and that the distribution $Q'_{\varepsilon,x}$ depend at least in a measurable way on x. $\triangleleft$

Remark 3.9. By de Finetti's theorem, every exchangeable sequence is a mixture of i.i.d. sequences. Furthermore, a computable extension of de Finetti's theorem (see [FR10] and [FR12]) shows that the representations in Theorems 3.6 and 3.7 can be computed when $X = \{1\}$ and $Y = \mathbb{R}$ (which can be straightforwardly extended to the case where X is countable and Y is a computable Polish space). $\triangleleft$

Acknowledgments

The author would like to thank Vikash Mansinghka for helpful discussions, and Matt Barr for comments on a draft. DMR was partially supported by graduate fellowships from the National Science Foundation and MIT Lincoln Laboratory, by a Newton International Fellowship, Emmanuel Research Fellowship, and by an NSERC Discovery Grant and Connaught Award. This material is based upon work supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA) under Contracts No. FA8750-14-C-0001 and FA8750-14-2-0004, the Army Research Office (ARO) grant W911NF-13-1-0212, the Office of Naval Research (ONR) grant N00014-13-1-0333, the National Science Foundation (NSF) grants DMS-0901020 and DMS-0800198, and grants from the John Templeton Foundation and Google. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force, Army, Navy, or DARPA, or the John Templeton Foundation.

References

[FKS11] S. Fischer, O. Kiselyov, and C. Shan. *Purely functional lazy nondeterministic programming*. J. Funct. Program. 21.4–5 (2011), pp. 413–465.

[FR10] C. E. Freer and D. M. Roy. *Posterior distributions are computable from predictive distributions*. In: Proc. 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010). Vol. 9. J. Machine Learning Research W&CP. 2010.

[FR12] C. E. Freer and D. M. Roy. *Computable de Finetti measures*. Ann. Pure Appl. Logic 163.5 (2012), pp. 530–546.

[GMRBT08] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. *Church: A language for generative models*. In: Proc. 24th Conf. on Uncertainty in Artificial Intelligence (UAI 2008). Corvalis, Oregon: AUAI Press, 2008, pp. 220–229.

[KS09] O. Kiselyov and C. Shan. *Embedded Probabilistic Programming*. In: Proc. of Domain-Specific Languages, IFIP TC 2 Working Conf (DSL 2009). Ed. by W. M. Taha. Vol. 5658. Lecture Notes in Comput. Sci. Springer, 2009, pp. 360–384.

[Lon99] J. Longley. *When is a Functional Program Not a Functional Program?* SIGPLAN Not. 34.9 (Sept. 1999), pp. 1–7.

[Man09] V. K. Mansinghka. “Natively Probabilistic Computing”. PhD thesis. Massachusetts Institute of Technology, 2009.

[MSP14] V. Mansinghka, D. Selsam, and Y. Perov. *Venture: a higher-order probabilistic programming platform with programmable inference*. ArXiv e-print 1404.0099 (Mar. 2014).

[RMGT08] D. M. Roy, V. K. Mansinghka, N. D. Goodman, and J. B. Tenenbaum. *A stochastic programming perspective on nonparametric Bayes*. Nonparametric Bayesian Workshop, Int. Conf. on Machine Learning. 2008.

[Sab98] A. Sabry. *What is a purely functional language?* J. Funct. Program. 8.1 (1998), pp. 1–22.

[SS90] H. Søndergaard and P. Sestoft. *Referential Transparency, Definiteness and Unfoldability*. Acta Informatica 27.6 (1990), pp. 505–517.

[SS92] H. Søndergaard and P. Sestoft. *Non-determinism in functional languages*. Comput. J. 35.5 (1992), pp. 514–523.

[Wu13] J. Wu. “Reduced Traces and JITing in Church”. MA thesis. Massachusetts Institute of Technology, 2013.