Statistical programming with categorical measure theory and LazyPPL

Swaraj Dash Younesse Kaddar Hugo Paquet Sam Staton
Department of Computer Science, University of Oxford, UK

Introduction. Categorical probability gives a categorical analysis of the fundamental principles and structures of classical probability theory. In this extended abstract and tool demonstration, we show that developments in categorical probability are relevant to practical Bayesian statistics and probabilistic programming. We present LazyPPL (Lazy Probabilistic Programming Library), a new library for statistical programming, which inherits key features of the theory, and facilitates the exchange of ideas between practical Bayesian modelling and categorical probability.

1 Categorical probability theory versus probabilistic programming

Various authors ([6, 17, 9]) have proposed semi-cartesian monoidal categories (monoidal categories whose unit object is terminal) as a good setting for categorical probability theory. The idea is that objects represent spaces, and morphisms represent parameterized probability distributions. The tensor models product spaces, used to reason about independence.

We connect this with a successful line of work using probabilistic programming for practical statistical modelling (e.g. [3, 18, 11, 24, 1]). An informal principle is that probabilistic programming can be seen as an internal language for categorical probability [23, 22], or as a 'practical' synthetic measure theory.

If morphisms correspond to probabilistic programs, the interchange law for monoidal categories means that we can re-order program lines without affecting the computational behaviour. Semi-cartesian structure means that, additionally, we can safely ignore any program instruction whose result is not used. In particular, it is safe to delay the execution of a program instruction until just before it is needed. This type of behaviour is called *lazy* (e.g. [13, Ch. 36]). In summary, we have the following dictionary.

Traditional probability	*Categorical probability*	*Probabilistic programming in LazyPPL*
Spaces	Objects	Types
Parameterized distributions	Morphisms	Programs
Fubini's theorem	Monoidal interchange law	Reordering lines of programs
Normalized probabilities (sum to 1)	Semi-cartesian / terminal unit	Laziness

2 Overview of probabilistic programming in LazyPPL

The basic idea of LazyPPL is that types represent spaces, and probabilistic programs represent measures on those spaces. We take advantage of the type infrastructure in Haskell to explore a range of spaces, using basic type constructors. We have the usual types $\mathbb{R}$, `Bool` of real numbers and booleans. Then we can form finite product types, e.g. `(`$\mathbb{R}$`, `$\mathbb{R}$`)`. For every type `a`, there is a type (`Prob a`) of probability distributions on `a`, and a type (`Meas a`) of *unnormalized* measures on `a`. We obtain *random* distributions as (`Prob (Prob a)`), and so on. For all types `a` and `b` there is a function type (`a` $\to$ `b`), so we get parameterized distributions, e.g. ($\mathbb{R}$ $\to$ `Prob` $\mathbb{R}$), and random functions, e.g. (`Prob (a` $\to$ `b)`).

In categorical probability, the focus is typically on the monoidal category of probability kernels. These are the programs of type (`a` $\to$ `Prob b`) or (`a` $\to$ `Meas b`), depending on whether we are working with normalized distributions (as in Markov categories, [9]) or unnormalized distributions (as in CD-categories [6], or GS-categories [7], or 'synthetic measure theory' [17, 20]). Composition of these morphisms is dealt with by the fact that `Prob` and `Meas` are monads in Haskell, and Haskell has very convenient syntax for Kleisli composition.

Submitted to:
ACT 2022

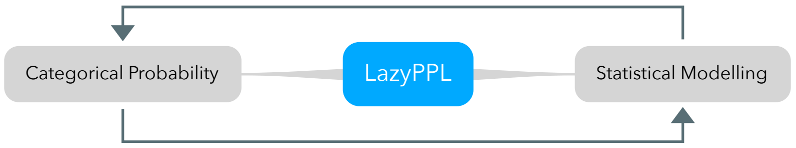
We can then write statistical models in the style of categorical probability, but formally in Haskell syntax, using special operations provided by LazyPPL. For example, (`uniform :: Prob` $\mathbb{R}$) represents the uniform distribution on $[0,1]$.

We view unnormalized measures as probability distributions *weighted* by a likelihood function. These are built out of operations (`sample :: Prob a` $\rightarrow$ `Meas a`), for sampling from a normalized distribution, and (`score ::` $\mathbb{R}$ $\rightarrow$ `Meas ()`), for reweighting the measure. The crux of Monte Carlo simulation for Bayesian inference is in sampling from an unnormalized measure (e.g. [10]). In LazyPPL we provide three alternative routines to do this (`mh`, `mh1`, `lwis`).

3 Laziness, terminal units and non-parametric statistics

There are several other implementations of categorical aspects of probability, including MonadBayes [19] and EfProb [5]; moving beyond probability we also mention [12, 4, 2, 14]. The **key novelty** of LazyPPL is to take full advantage of laziness in program evaluation, which amounts to a monoidal category having a terminal unit. This allows us to immediately compose complex distributions from non-parametric statistics, which refer to sample spaces that are infinite or unbounded.

For example, in Figure 1 we display several instances of Bayesian regression in LazyPPL. The first is parametric linear regression. The second is piecewise linear regression: here the positions of jump points come from an infinite dimensional Poisson process; this is easily written in two lines of code, and composed with linear regression in three lines of code. Although the actual dimension is infinite, the inference concludes and we get the graphs. This is where laziness comes in: the execution engine only looks at the dimensions that are needed to plot the graph, and the bounds on the viewport cause many of the dimensions to be implicitly discarded, essentially using the terminal unit. All of this is automatic and taken care of by the library. For the code and further examples, including non-parametric clustering and non-parametric feature extraction, see `https://lazyppl.bitbucket.io/`. Our implementation requires several innovations, including a lazy probability monad that is inspired by quasi-Borel spaces [15] and the Para construction [21], and a new Metropolis-Hastings kernel. See [8] for details. In summary, LazyPPL facilitates connections between categorical probability and practical statistics, as follows:



New directions for categorical probability: LazyPPL naturally suggests objects that are easy to program with but which are motivating challenges for traditional/categorical probability, such as good function spaces and commutative monads of measures.

New lessons for theory and practice of statistics: Laziness as a means of automatic truncation in non-parametric statistics; Compositional model building allows new abstractions (e.g. [16]).

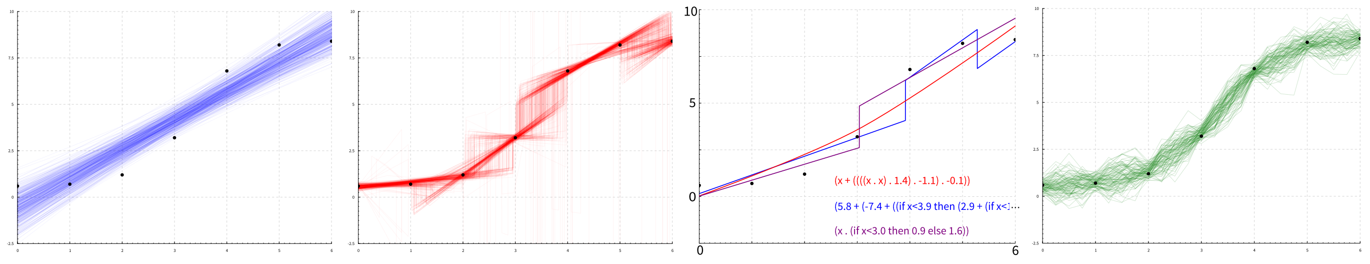


Figure 1: Bayesian regression in LazyPPL for the data set indicated by the dots. Samples from the posteriors starting from different priors on the function space: (a) linear, (b) piecewise linear, (c) program induction, and (d) Wiener process.

References

[1] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

[2] J. Bolt, J. Hedges, and P. Zahn. Bayesian open games.

[3] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[4] Catlab.jl. A framework for applied category theory in the julia language. `https://github.com/AlgebraicJulia/Catlab.jl`.

[5] K. Cho and B. Jacobs. The EfProb library for probabilistic calculations. In *Proc. CALCO 2017*, 2017.

[6] K. Cho and B. Jacobs. Disintegration and Bayesian inversion via string diagrams. *Math. Struct. Comput. Sci.*, 29:938–971, 2019.

[7] A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7(4):299–331, 1999.

[8] S. Dash, Y. Kaddar, H. Paquet, and S. Staton. LazyPPL: Monads and laziness for Bayesian modelling. Preprint available at `https://www.cs.ox.ac.uk/people/hugo.paquet/lazyppl.pdf`.

[9] T. Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Adv. Math.*, 370, 2020.

[10] C. Geyer. Introduction to Markov Chain Monte Carlo. In *Handbook of Markov Chain Monte Carlo*. Chapman Hall/CRC, 2011.

[11] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. 2008.

[12] M. Halter, E. Patterson, A. Baas, and J. Fairbanks. Compositional Scientific Computing with Catlab and SemanticModels.

[13] R. Harper. *Practical foundations for programming languages*. Cambridge University Press, 2016.

[14] J. Hedges. Getting started with open games. `https://github.com/jules-hedges/open-games-hs`.

[15] C. Heunen, O. Kammar, S. Staton, and H. Yang. A convenient category for higher-order probability theory. In *Proc. LICS 2017*, 2017.

[16] 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.

[17] A. Kock. Commutative monads as a theory of distributions. *Theory and Applications of Categories*, 26(4), 2012.

[18] D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*, 28(25):3049–3067, 2009.

[19] A. Ścibior, O. Kammar, and Z. Ghahramani. Functional programming for modular Bayesian inference. In *Proc. ICFP 2018*, 2018.

[20] A. Ścibior, O. Kammar, M. Vákár, and S. Staton. Denotational validation of higher-order Bayesian inference. *Proceedings of POPL*, 2018.

[21] D. Shiebler. Categorical stochastic processes and likelihood. In *Proc. ACT 2020*, 2020.

[22] S. Staton. Probabilistic programs as measures. In *Foundations of Probabilistic Programming*. CUP, 2020.

[23] D. M. Stein. Structural Foundations for Probabilistic Programming Languages, 2021. DPhil thesis.

[24] D. Tolpin, H. Yang, J. W. van de Meent, and F. Wood. Design and implementation of probabilistic programming language Anglican. 2016.