Higher order programming with probabilistic effects: A model of stochastic memoization and name generation

Younesse Kaddar Sam Staton
Department of Computer Science, University of Oxford, UK

This abstract is about probabilistic programming, which is a method of programming statistical and machine learning models. By combining higher-order functions, we can specify increasingly complex models. This abstract focuses on *stochastic memoization*, a higher-order method that is simple and useful in practice, but semantically elusive, particularly regarding dataflow transformations.

Deterministic versus stochastic memoization. Deterministic memoization, for a function f, is the process of storing the result when f is applied to an argument, so that we can reuse it later when the same call is made [Mic68]. Since we only need to compute results once, this leads to a speed-up of the program, but it does not change its semantics. However, in the presence of probabilistic effects, memoizing is no longer just an optimization technique. It does change the semantics [Roy+08; Sta21; Woo+09], enabling us to define infinite random sequences, which are of paramount importance in probability and statistics, as we now explain.

Typed stochastic memoization with monads. Suppose we have a monad for probabilistic effects, Prob. *Stochastic memoization* is a higher-order function with probabilistic effects, of type $\mathsf{mem} :: (\mathsf{a} \to \mathsf{Prob}\ \mathsf{b}) \to \mathsf{Prob}\ (\mathsf{a} \to \mathsf{b})$. For example, suppose we start with a function which randomly picks a number in $[0, 1]$ every time it is called, $\mathbf{const}\ (\mathsf{uniform}\ 0\ \ 1)\ :: \ \mathbb{R} \to \mathsf{Prob}\,\mathbb{R}$. Then the memoized function randomly assigns a number to every input number, $\mathsf{mem}\ (\mathbf{const}\ (\mathsf{uniform}\ 0\ \ 1))\ :: \ \mathsf{Prob}\ (\mathbb{R} \to \mathbb{R})$. This is sometimes called white noise, and it is an important first example; we give a more substantial example based on random graphs in Section 1. In practice, memoization is crucial in Church [Goo+08] and WebPPL [GS14], and in this typed form in our Haskell library LazyPPL [Sta+]

Data flow properties. Stochastic memoization is easy to implement using state. (When a is enumerable, in LazyPPL we can also use laziness and tries, following [Hin00]; some languages also include state, e.g. [Kis]). Unlike a fully stateful effect, however, stochastic memoization is still compatible with commutativity / data flow program transformations:

$$\mathsf{x} \leftarrow \mathsf{t}\ ;\ \mathsf{y} \leftarrow \mathsf{u} \quad = \quad \mathsf{y} \leftarrow \mathsf{u}\ ;\ \mathsf{x} \leftarrow \mathsf{t} \qquad \text{where } \mathsf{x} \notin \mathsf{freevars}(\mathsf{u}),\ \mathsf{y} \notin \mathsf{freevars}(\mathsf{t}) \tag{1}$$

These transformations are very useful in program optimization and inference algorithms. On the foundational side, data flow is a fundamental concept that corresponds to monoidal categories. The challenge is to validate these transformations.

Challenges for probability theory. On the semantic side, these transformations are not trivial. Informally, stochastic memoization appears related to Kolmogorov's extension theorem, which relates probability measures on infinite product spaces to probability measures on the projections from the product. Arguably, then, stochastic memoization validates dataflow transformations because it is not *intrinsically* stateful, rather, it is linked to a fundamental part of pure probability theory. However, traditional probability theory does not actually support higher-order functions [Aum61], and higher-order probability is a burgeoning field [CJ19; Fri20; Heu+17; Koc11; Ste21]. Existing models of higher-order probability do not support stochastic memoization, and this is the contribution of our work: a first model of stochastic memoization with a non-enumerable type and validating the dataflow transformations (1).

1 Examples of programming with random graphs via stochastic memoization

Stochastic memoization is especially important for programming statistical models with random relational structures, *e.g.* graph social networks, world wide web, biochemical pathways, etc. Consider, for example, a Haskell typeclass $\mathsf{RandomGraph}$ corresponding to an abstract type whose interface allows us to generate a new random graph (with

Submitted to:
HOPE 2022

newGraph) from a seed g, draw vertices at random (with newVertex), and inspect the presence of edges between vertices (with isEdge). Every measurable function $g\colon [0,1]^2 \to [0,1]$ (called a *graphon*, *e.g.* [OR13]) can be used as a seed, leading to the following implementation in our library LazyPPL [Sta+]:

```
class RandomGraph g where
  type Graph g
  data Vertex g
  newGraph:: g → Prob (Graph g)
  newVertex:: Graph g → Prob (Vertex g)
  isEdge :: Graph g → Vertex g → Vertex g → Bool
newtype Graphon = G ((ℝ, ℝ) → ℝ)
```

```
instance RandomGraph Graphon where
  type Graph Graphon = (ℝ, ℝ) → Bool
  data Vertex Graphon = V ℝ
  -- return a randomly sampled function '(ℝ, ℝ) → Bool'
  newGraph (G graphon) = mem $ bernoulli . graphon
  newVertex _ = V <$> uniform
  isEdge graph (V x) (V y) = graph (x, y)
```

assuming we have the second-order memoization function mem :: (a → Prob b) → Prob (a → b). The idea is that, once an edge between x and y has been sampled (with probability graphon (x, y)), its presence (or absence) remains unchanged in the rest of the program execution, hence the need to memoize the result. For example, newGraph (G (**const** 0.5)) generates the Erdős-Rényi random graph [ER59].

Although we can define some instances of RandomGraph without using mem, such as geometric random graphs, it turns out that all instances are contextually equivalent to Graphon instances. Here the data flow property (1) corresponds to the statistical 'exchangeability' of vertices (*e.g.* [AFR16; Sta20; Sta+17]). Beyond exchangeable random graphs, memoization is important for more complex / deep exchangeable random datatypes (e.g. [Jun+20; Sta+17]).

2 Semantic models for stochastic memoization

Stochastic functions are usually interpreted as probabilistic kernels $f\colon X \to PY$, where P is a probability monad on a suitable category [Fri20; Koc11]. A semantic model for stochastic memoization should then admit a cartesian closed structure (to model higher-order functions) and a morphism $\mathrm{mem}_{X,Y}\colon (PY)^X \longrightarrow P(Y^X)$. For every f written as a lambda-abstraction $\lambda x.\, u\colon X \to PY$, $\mathrm{mem}_{X,Y}(f)$ will be denoted by $\lambda_{\text{מ}} x.\, u$, so that we require equations such as:

$$\begin{array}{l} \mathsf{f} \leftarrow \lambda_{\text{מ}} x.\, u \\ \mathsf{f}\ \ \mathsf{n} \end{array} \quad \overset{\text{one sample}}{=} \quad u[\mathsf{n}/x] \qquad\qquad \begin{array}{l} \mathsf{f} \leftarrow \lambda_{\text{מ}} x.\, u \\ \mathsf{v}_1 \leftarrow \mathsf{f}\ \mathsf{n} \\ \mathsf{w} \leftarrow \mathsf{f}\ \mathsf{m} \\ \mathsf{v}_2 \leftarrow \mathsf{f}\ \mathsf{n} \\ \mathbf{return}\ (\mathsf{v}_1,\ \mathsf{w},\ \mathsf{v}_2) \end{array} \quad \overset{\text{several samples}}{=} \quad \begin{array}{l} \mathsf{v} \leftarrow u[\mathsf{n}/x] \\ \mathsf{w} \leftarrow u[\mathsf{m}/x] \\ \mathbf{return}\ (\mathsf{v},\ \mathsf{w},\ \mathsf{v}) \end{array} \tag{2}$$

To prove that equations (2) are consistent with the dataflow property (1), we give a denotational model. For simplicity, we focus on Boolean-valued functions over a non-enumerable type of atoms [Pit13]. Our model is based on functors over finite bipartite graphs (bigraphs, for short), see [KS22] for details. At a lower level, we have the following interface:

```
new_atom :: 𝔸 -- Atoms (randomly generated fresh names)
new_function :: 𝔽 -- Function labels : type to be thought of as 𝔸 → Bool
(@) :: (𝔽, 𝔸) → Bool -- Application operator making every function memoized: type of a bipartite graph
```

where every function from a set of atoms $\mathbb{A}$ to **Bool** is viewed as an inhabitant of type $\mathbb{F}$ (thought of as $\mathbb{A} \to$ **Bool**). Applying a function to an argument and memoizing the result amounts to the explicit 'apply' operator (@) :: $\mathbb{F} \times \mathbb{A} \to$ **Bool**. But requiring that the results be memoized is precisely saying that @ ought to be seen as the 'edge' relation of a bigraph with set $\mathbb{F}$ of left nodes and $\mathbb{A}$ of right nodes, the edges of which are such that their presence (or absence) remains unchanged after being sampled, like isEdge in RandomGraph. Inspired by the local state monad [PP02] (which was defined on the functor category $[\mathbf{Inj}, \mathbf{Set}]$, where **Inj** is the category of finite sets and injections), we model probabilistic and name generation effects by a new monad T on $[\mathbf{BiGrph}_{emb}, \mathbf{Set}]$, where $\mathbf{BiGrph}_{emb}$ is the category of finite bigraphs and embeddings. We then use it to give a categorical semantics to our language, and we show that:

Theorem 2.1. *The monad T validates (2). Moreover, it is strong commutative and affine (i.e. an abstract model of probability [Koc11]) and therefore satisfies the dataflow property (1).*

References

[AFR16] Nathanael L Ackerman, Cameron E Freer and Daniel M Roy. "EXCHANGEABLE RANDOM PRIMITIVES". In: PPS 2016. St. Petersburg, Florida, United States, 2016, page 4.

[Aum61] Robert J. Aumann. "Borel Structures for Function Spaces". In: *Illinois Journal of Mathematics* 5.4 (Dec. 1, 1961), pages 614–630. DOI: 10.1215/ijm/1255631584. URL: https://doi.org/10.1215/ijm/1255631584.

[CJ19] Kenta Cho and Bart Jacobs. "Disintegration and Bayesian Inversion via String Diagrams". In: *Mathematical Structures in Computer Science* 29.7 (Aug. 2019), pages 938–971. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/s0960129518000488. arXiv: 1709.00322. URL: http://arxiv.org/abs/1709.00322 (visited on 01/25/2021).

[ER59] P. Erdös and A. Rényi. "On Random Graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), page 290.

[Fri20] Tobias Fritz. "A Synthetic Approach to Markov Kernels, Conditional Independence and Theorems on Sufficient Statistics". In: *Advances in Mathematics* 370 (Aug. 2020), page 107239. ISSN: 00018708. DOI: 10.1016/j.aim.2020.107239. arXiv: 1908.07021. URL: http://arxiv.org/abs/1908.07021 (visited on 01/31/2021).

[GS14] Noah D Goodman and Andreas Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. 2014. URL: http://dippl.org/ (visited on 08/17/2021).

[Goo+08] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz and Joshua B. Tenenbaum. "Church: A Language for Generative Models". In: *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*. UAI'08. Arlington, Virginia, USA: AUAI Press, July 9, 2008, pages 220–229. ISBN: 978-0-9749039-4-1.

[Heu+17] Chris Heunen, Ohad Kammar, Sam Staton and Hongseok Yang. "A Convenient Category for Higher-Order Probability Theory". In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (June 2017), pages 1–12. DOI: 10.1109/lics.2017.8005137. arXiv: 1701.02547. URL: http://arxiv.org/abs/1701.02547 (visited on 12/11/2020).

[Hin00] Ralf Hinze. "Generalizing Generalized Tries". In: *Journal of Functional Programming* 10 (July 2000), pages 327–351. DOI: 10.1017/S0956796800003713.

[Jun+20] Paul Jung, Jiho Lee, Sam Staton and Hongseok Yang. "A Generalization of Hierarchical Exchangeability on Trees to Directed Acyclic Graphs". In: *Annales Henri Lebesgue* 4 (July 24, 2020), pages 325–368. DOI: 10.5802/ahl.74. arXiv: 1812.06282. URL: http://arxiv.org/abs/1812.06282 (visited on 11/13/2020).

[KS22] Younesse Kaddar and Sam Staton. "A Model of Stochastic Memoization and Name Generation". University of Oxford, 2022. URL: https://younesse.net/assets/stochmem.pdf.

[Kis] Oleg Kiselyov. *Logic Programming in HANSEI*. URL: https://okmij.org/ftp/kakuritu/logic-programming.html (visited on 06/09/2022).

[Koc11] Anders Kock. "Commutative Monads as a Theory of Distributions". In: *Theory and Applications of Categories* 26 (Aug. 30, 2011). arXiv: 1108.5952. URL: http://arxiv.org/abs/1108.5952 (visited on 02/07/2021).

[Mic68] Donald Michie. ""Memo" Functions and Machine Learning". In: *Nature* 218.5138 (5138 Apr. 1968), pages 306–306. ISSN: 1476-4687. DOI: 10.1038/218306c0. URL: https://www.nature.com/articles/218306c0 (visited on 06/09/2022).

[OR13] Peter Orbanz and Daniel M. Roy. "Bayesian Models of Graphs, Arrays and Other Exchangeable Random Structures". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (Dec. 2013). DOI: 10.1109/TPAMI.2014.2334607. arXiv: 1312.7857. URL: http://arxiv.org/abs/1312.7857 (visited on 12/03/2021).

[Pit13] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science 57. Cambridge ; New York: Cambridge University Press, 2013. 276 pages. ISBN: 978-1-107-01778-8.

[PP02] Gordon Plotkin and John Power. “Notions of Computation Determine Monads”. In: *Foundations of Software Science and Computation Structures*. Edited by Mogens Nielsen and Uffe Engberg. Redacted by Gerhard Goos, Juris Hartmanis and Jan van Leeuwen. Volume 2303. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pages 342–356. ISBN: 978-3-540-43366-8 978-3-540-45931-6. DOI: 10.1007/3-540-45931-6_24. URL: http://link.springer.com/10.1007/3-540-45931-6_24 (visited on 05/24/2021).

[Roy+08] D. Roy, Vikash K. Mansinghka, Noah D. Goodman and J. Tenenbaum. “A Stochastic Programming Perspective on Nonparametric Bayes”. In: 2008. URL: https://www.semanticscholar.org/paper/A-stochastic-programming-perspective-on-Bayes-Roy-Mansinghka/946ed04f705a2a28539a8af1f5e9ccdedd7fabc2 (visited on 01/21/2022).

[Sta20] Sam Staton. “Categorical Models of Probability with Symmetries”. Categorical Probability and Statistics. June 2020. URL: http://perimeterinstitute.ca/personal/tfritz/2019/cps_workshop/slides/staton.pdf.

[Sta21] Sam Staton. “Some Formal Structures in Probability”. 2021. URL: http://www.cs.ox.ac.uk/people/samuel.staton/2021fscd.pdf.

[Sta+] Sam Staton, Hugo Paquet, Swaraj Dash and Younesse Kaddar. *LazyPPL*. URL: https://lazyppl.bitbucket.io/ (visited on 04/04/2022).

[Sta+17] Sam Staton, Hongseok Yang, Nathanael Ackerman, Cameron Freer and Daniel M Roy. “Exchangeable Random Processes and Data Abstraction”. In: PPS Workshop. Paris, France, 2017, page 4. URL: http://www.cs.ox.ac.uk/people/hongseok.yang/paper/pps17a.pdf.

[Ste21] Dario Maximilian Stein. “Structural Foundations for Probabilistic Programming Languages”. University of Oxford, 2021. 221 pages.

[Woo+09] Frank Wood, Cédric Archambeau, Jan Gasthaus, Lancelot James and Yee Whye Teh. “A Stochastic Memoizer for Sequence Data”. In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML ’09*. The 26th Annual International Conference. Montreal, Quebec, Canada: ACM Press, 2009, pages 1–8. ISBN: 978-1-60558-516-1. DOI: 10/fg8z4q. URL: http://portal.acm.org/citation.cfm?doid=1553374.1553518 (visited on 01/21/2022).