Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets

NATE ACKERMAN, Harvard University, USA
CAMERON E. FREER, Massachusetts Institute of Technology, USA
YOUNESSE KADDAR, University of Oxford, UK
JACEK KARWOWSKI, University of Oxford, UK
SEAN MOSS, University of Birmingham, UK
DANIEL ROY, University of Toronto, Canada
SAM STATON, University of Oxford, UK
HONGSEOK YANG, KAIST, South Korea

We study semantic models of probabilistic programming languages over graphs, and establish a connection to graphons from graph theory and combinatorics. We show that every well-behaved equational theory for our graph probabilistic programming language corresponds to a graphon, and conversely, every graphon arises in this way.

We provide three constructions for showing that every graphon arises from an equational theory. The first is an abstract construction, using Markov categories and monoidal indeterminates. The second and third are more concrete. The second is in terms of traditional measure theoretic probability, which covers 'black-and-white' graphons. The third is in terms of probability monads on the nominal sets of Gabbay and Pitts. Specifically, we use a variation of nominal sets induced by the theory of graphs, which covers Erdős-Rényi graphons. In this way, we build new models of graph probabilistic programming from graphons.

CCS Concepts: • **Theory of computation** → **Semantics and reasoning**; **Probabilistic computation**.

Additional Key Words and Phrases: probability monads, exchangeable processes, graphons, nominal sets, Markov categories, probabilistic programming

1 INTRODUCTION

This paper is about the semantic structures underlying probabilistic programming with random graphs. Random graphs have applications in statistical modelling across biology, chemistry, epidemiology, and so on, as well as theoretical interest in graph theory and combinatorics (e.g. [Bornholdt and Schuster 2002]). Probabilistic programming, i.e. programming for statistical modelling [van de

---

Authors' addresses: Nate Ackerman, Harvard University, USA, nate@aleph0.net; Cameron E. Freer, Massachusetts Institute of Technology, USA, freer@mit.edu; Younesse Kaddar, University of Oxford, UK, younesse.kaddar@chch.ox.ac.uk; Jacek Karwowski, University of Oxford, UK, jacek.karwowski@cs.ox.ac.uk; Sean Moss, University of Birmingham, UK, s.k.moss@bham.ac.uk; Daniel Roy, University of Toronto, Canada, daniel.roy@utoronto.ca; Sam Staton, University of Oxford, UK, sam.staton@cs.ox.ac.uk; Hongseok Yang, KAIST, School of Computing, South Korea, hongseok00@gmail.com.

Meent et al. 2018], is useful for building the statistical models for the applications. Moreover, as we show (Theorem 23 and Corollary 26), the semantic aspects of programming languages for random graphs correspond to graphons [Lovász 2012], a core structure in graph theory and combinatorics.

To set the scene more precisely, we recall the setting of probabilistic programming with real-valued distributions, and contrast it with the setting with graphs. Many probabilistic programming languages provide a type of real numbers (real) and distributions such as the normal distribution

$$\text{normal} : \text{real} * \text{real} \rightarrow \text{real} \tag{1}$$

together with arithmetic operations such as

$$(+) : \text{real} * \text{real} \rightarrow \text{real}. \tag{2}$$

Even if we encounter an unfamiliar distribution over (real) in a library, we have a rough idea of how to explain what it could be, in terms of probability densities and measures.

In this paper, we consider the setting of probabilistic programming with graphs, where the probabilistic programming language or library provides a type (vertex) and some distribution

$$\text{new} : \text{unit} \rightarrow \text{vertex} \tag{3}$$

together with a test

$$\text{edge} : \text{vertex} * \text{vertex} \rightarrow \text{bool}. \tag{4}$$

Our goal is to analyze the interface (vertex, new, edge) for graphs semantically, and answer, for instance, what they could be and what they could do. We give one example analysis in Section 1.1 first, and the general one later in Theorem 23 and Corollary 26, which says that to give an implementation of (vertex, new, edge), satisfying the laws of probabilistic programming, is to give a graphon. In doing so, we connect the theory of probabilistic programming with graph theory and combinatorics.

Probabilistic programming is generally used for statistical inference, in which we describe a generative model by writing a program using primitives such as (1)–(4) above, and then infer a distribution on certain parameters, given particular observed data. This paper is focused on the generative model aspect, and not inference (although for simple examples, generic inference methods apply immediately, see §1.5).

1.1 Example of an Implementation of a Random Graph: Geometric Random Graphs

To illustrate the interface (vertex, new, edge) of (3)–(4), we consider for illustration a random geometric graph (e.g. [Bubeck et al. 2016; Penrose 2003]) where the vertices are points on the surface of the unit sphere, chosen uniformly at random, and where there is an edge between two vertices if the angle between them is less than some fixed θ. This random graph might be used, for instance, to model the connections between people on the earth.

For example, a simple statistical inference problem might start from the observed connectivity in Figure 1(a). We might ask for the distribution on θ given that this graph arose from the spherical random geometric graph. One sample from this posterior distribution on random geometric graphs with $\theta = \pi/3$ is shown in Figure 1(b). Another, unconditioned sample from the random geometric graph with $\theta = \pi/6$ is shown in Figure 1(c).

We can regard this example as an implementation of the interface (vertex, new, edge) as follows: we implement (vertex) as the surface of the sphere (e.g. implemented as Euclidean coordinates).

- new() : vertex randomly picks a new vertex as a point on the sphere uniformly at random. Figure 1(c) shows the progress after calling new() 15 times.
- edge : vertex * vertex → bool checks whether there is an edge between two vertices; this amounts to checking whether the angle between two points is less than θ.
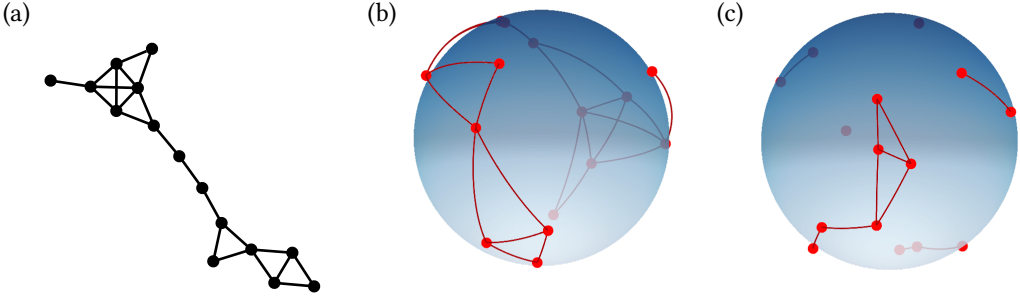
Fig. 1. (a) A graph; (b) an inferred geometric realization of it ($\theta \approx \pi/3$); (c) a generated sample for $\theta = \pi/6$.

For a simple example, we can write a program over the interface to calculate the probability of three random vertices forming a triangle: the program

$$
\begin{aligned}
&\mathsf{let}\ a\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{let}\ b\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{let}\ c\ =\ \mathsf{new}()\ \mathsf{in} \\
&\mathsf{edge}(a, b)\ \&\ \mathsf{edge}(b, c)\ \&\ \mathsf{edge}(a, c)\ :\ \mathsf{bool}
\end{aligned}
\tag{5}
$$

randomly returns true or false; the probability of true is the probability of a triangle.

This implementation using the sphere is only one way to implement (vertex, new, edge). There are implementations using higher-dimensional spheres, or other geometric objects. We can also consider random equivalence relations as graphs, i.e. disjoint unions of complete graphs, or random bipartite graphs, which are triangle-free. We can consider the Erdős–Rényi random graph, where the chance of an edge between two vertices is independent of the other edges, and has a fixed probability. These are all different implementations of the same abstract interface, (vertex, new, edge), and programs such as (5) make sense for all of them. The point of this paper is to characterize all these implementations, as graphons.

1.2 Implementations Regarded as Equational Theories

The key method of this paper is to treat implementations of the interface (vertex, new, edge) extensionally, as equational theories. That is, rather than looking at specific implementation details, we look at the equations between programs that a user of the implementation would rely on. (This is analogous to the idea in model theory of studying first-order theories rather than specific models; similar ideas arise in the algebraic theory of computational effects [Plotkin and Power 2002].) For example, if an implementation always provides a bipartite random graph, we have the equation

$$\text{Program (5)} \quad \equiv \quad \text{false} \qquad \text{between programs,}$$

because a triangle is never generated. This equation does not hold in the example of Figure 1(b–c), since triangles are possible.

We focus on a class of equational theories that are well behaved, as follows. First, we suppose that they contain basic laws for probabilistic programming (eqns. (7) – (11), §2.2). This basic structure already appears broadly in different guises, including in Moggi's monadic metalanguage [Moggi 1989], in linear logic [Ehrhard and Tasson 2019], and in synthetic probability theory [Fritz 2020]. Second, we suppose that the equational theories are equipped with a 'Bernoulli base', which means that although we do not specify an implementation for the type (vertex), each closed program of type (bool) is equated with some ordinary Bernoulli distribution, in such a way as to satisfy the classical laws of traditional finite probability theory (§ 2.4). Finally, we suppose that the edge relation is symmetric (the graphs are undirected) and that it doesn't change when the same question

is asked multiple times ('deterministic'), e.g.

$$\text{let } a \; = \; \text{new}() \text{ in let } b \; = \; \text{new}() \text{ in edge}(a,b) \; \& \; \neg\text{edge}(a,b) \quad \equiv \quad \text{false.} \tag{6}$$

A *graphon* is a symmetric measurable function $[0,1]^2 \to [0,1]$. We show that every equational theory for the interface (vertex, new, edge) gives rise to a graphon (Theorem 23), and conversely that every graphon arises in this way (Corollary 26).

We emphasize that this abstract treatment of implementations, in terms of equational theories, is very open-ended, and permits a diverse range of implementation methods. Indeed, we show in Section 5 that any implementation using traditional measure-theoretic methods will only produce black-and-white graphons, so this abstract treatment is crucial.

1.3 From Equational Theories to Graphons

In Section 3, we show how an equational theory over programs in the interface (vertex, new, edge) gives rise to a graphon. The key first step is that graphons (modulo equivalence) can be characterized in terms of sequences of finite random graphs that satisfy three conditions: exchangeability, consistency, and locality.

To define a graphon, we show how to define programs that describe finite random graphs, by using new and edge to build boolean-valued $n \times n$ adjacency matrices, for all n (shown in (18)). Assuming that the equational theory of programs is Bernoulli-based, these programs can be interpreted as probability distributions on the finite spaces of adjacency matrices which, we show, are finite random graphs.

It remains to show that the induced sequence of random graphs satisfies the three conditions for graphons (exchangeability, consistency, and locality). These can be formulated as equational properties, and so they can be verified by using the equational reasoning in the equational theory. This is Theorem 23. A key part of the proof is the observation that exchangeability for graphons connects to commutativity of let (9): we can permute the order in which vertices are instantiated without changing the distributions.

1.4 From Graphons to Equational Theories

We also show the converse: every graphon arises from a good equational theory for the interface (vertex, new, edge). We look at this from three angles: first, we prove this in the general case using an abstract method, and then, we use concrete methods for two special cases.

Fixing a graphon, we build an equational theory by following a categorical viewpoint. A good equational theory for probabilistic programming amounts to a 'distributive Markov category', which is a monoidal category with coproducts that is well-suited to probability (§2.2 and [Fritz 2020]). The idea that distributive categories are a good way to analyze abstract interfaces goes back at least to [Walters 1989], which used distributive categories to study interfaces for stacks and storage. We can thus use now-standard abstract methods for building monoidal and distributive categories to build an equational theory for the programming language.

We proceed in two steps. We first use methods such as [Hermida and Tennent 2012; Hu and Tholen 1995] to build an abstract distributive Markov category that supports the interface (vertex, new, edge) in a generic way. This equational theory is generic and not Bernoulli-based: although it satisfies the equational laws of probabilistic programming, there is no given connection to traditional probability. The second step is to show that (a) it is possible to quotient this generic category to get Bernoulli-based equational theories; (b) the choices of quotient are actually in bijective correspondence with graphons. Thus, we can build an equational theory from which any given graphon arises, via (18): this is Corollary 26. (The framework of Bernoulli-based Markov categories is new here, and the techniques of [Hermida and Tennent 2012; Hu and Tholen 1995]

have not previously been applied in categorical probability, so a challenge for future work is to investigate these ideas in other aspects of categorical probability.)

Although this is a general method, it is an abstract method involving quotient constructions. The ideal form of denotational semantics is to explain what programs are by regarding them as functions between certain kinds of spaces. Although Corollary 26 demonstrates that every graphon arises from an equational theory, the type (vertex) is interpreted as an object of an abstract category, and programs are equivalence classes of abstract morphisms. In the remainder of the paper, we give two situations where we can interpret (vertex) as a genuine concrete space, and programs are functions or distributions on spaces. Such an interpretation immediately yields an equational theory, where two programs are equal if they have the same interpretation.

- *Section 5*: For 'black-and-white graphons', we present measure-theoretic models of the interface, based on a standard measure-theoretic interpretation of probabilistic programming (e.g. [Kozen 1981]). We interpret (vertex) as a measurable space, and (new) as a probability measure on it, and (edge) in terms of a measurable predicate. Then, the composition of programs is defined in terms of probability kernels and Lebesgue integration. This kind of model exactly captures the black-and-white graphons (Prop. 29).
- *Section 6*: For 'Erdős–Rényi' graphons, which are constantly gray, and not black-and-white, we present a model based on Rado-nominal sets (§6.1). These are a variant of nominal sets ([Gabbay and Pitts 1999; Pitts 2013]) where the atoms are vertices of the Rado graph (following [Bojańczyk et al. 2014]). We consider a new notion of 'internal probability measure' in this setting, and use this to give a compositional semantics that gives rise to the Erdős–Rényi graphons (Corollary 45).

Together, these more concrete sections then provide further intuition for the correspondence between equational theories and graphons.

1.5 Connection to Practice

We conclude this introduction with remarks on the connection to practical modelling. In practice, the graph interface might form part of a generative model, on which we perform inference. The structure is clearest in a typed language, and one example is the LazyPPL library for Haskell [Dash et al. 2023]. (Similar examples are implemented in [Goodman and Tenenbaum 2023, Ch. 12], albeit untyped.) Then our interface is captured by a Haskell type class:[1]

```
class RandomGraph p vertex | p → vertex where
  new :: p → Prob vertex
  edge :: p → vertex → vertex → Bool
```

Here p is a parameter type, and we write Prob for a probability monad. A spherical implementation of the interface (following §1.1) is parameterized by the dimension d and the distance θ, as follows:

```
data SphGrph = SG Int Double -- parameters for a sphere graph
data SphVertex = SV [Double] -- vertices are Euclidean coordinates
instance RandomGraph SphGrph SphVertex where
  new :: SphGrph → Prob SphVertex
  new (SG d theta) = ... -- sample a random unit d-vector uniformly
  edge :: SphGrph → SphVertex → SphVertex → Bool
  edge (SG d theta) v w = ... -- check whether arccos(v.w) < theta
```

We can use this as a building block for more complex models. For a simple example, we generated Figure 1(b) by using the generic Metropolis-Hastings inference of the LazyPPL library to infer θ given

---

[1]See https://lazyppl-team.github.io/GraphDemo.html for full details in literate Haskell.

a particular graph (Fig. 1(a)). We have also implemented other random graphs; our implementation of the Erdős–Rényi graph uses stochastic memoization [Kaddar and Staton 2023; Roy et al. 2008].

Summary and context. As we have discussed, our main result is that equational theories for the programming interface (§1.1) give rise to graphons (§1.3) and every graphon arises in this way (§1.4).

These results open up new ways to study random graphs, by using programming semantics. On the other hand, our results here put the abstractions of practical probabilistic programming on a solid theoretical foundation (see also §7).

2 PROGRAMMING INTERFACES FOR RANDOM GRAPHS: EQUATIONAL THEORIES AND MARKOV CATEGORIES

In Section 1.1, we considered probabilistic programming over a graph interface. To make this formal, we now recall syntax, types, and equational reasoning for simple probabilistic programming languages. We begin with a general syntax (§2.1), which can accommodate various interfaces in the form of type and term constants, including the interface for graphs (Ex. 1(3)).

We study different instantiations of the probabilistic programming language in terms of the equational theories that they satisfy. We consider two equivalent ways of understanding equational theories: as distributive Markov categories (§2.2) and in terms of affine monads (§2.3). Markov categories are a categorical formulation of probability theory (e.g. [Fritz 2020]), and affine monads arise in the categorical analysis of probability (e.g. [Fritz et al. 2023; Jacobs 2018; Kock 2012]) as well as in the semantics for probabilistic programming (e.g. [Azevedo de Amorim 2023; Dahlqvist et al. 2018; Dash et al. 2023]). We make a connection with traditional probability via the notion of Bernoulli base (§2.4).

Much of this section will be unsurprising to experts: the main purpose is to collect definitions and results. The definition of *distributive* Markov category appears to be novel, and so we go over that definition and correspondence with monads (Propositions 8 and 13). In Section 2.5, we give a construction for quotienting a distributive Markov category, which we will need in Section 4. We include the result in the section because it may be of independent interest.

2.1 Syntax for a Generic Probabilistic Programming Language

Our generic probabilistic programming language is, very roughly, an idealized, typed fragment of a typical language like Church [Goodman et al. 2008]. We start with a simple programming language (following [Ehrhard and Tasson 2019; Staton 2017; Stein 2021] but also [Moggi 1989]) with at least the following product and sum type constructors:

$$A, A_1, A_2, B ::= \mathsf{unit} \mid 0 \mid A_1 * A_2 \mid A_1 + A_2 \mid \ldots$$

and terms, including the typical constructors and destructors but also explicit sequencing (let in)

$$t, t_1, t_2, u ::= x \mid () \mid (t_1, t_2) \mid \pi_1\, t \mid \pi_2\, t \mid \mathsf{in}_1\, t \mid \mathsf{in}_2\, t$$
$$\mid \mathsf{let}\, x = t_1 \,\mathsf{in}\, t_2 \mid \mathsf{case}\, t \,\mathsf{of}\, \{\} \mid \mathsf{case}\, t \,\mathsf{of}\, \{\mathsf{in}_1(x_1) \Rightarrow u_1; \mathsf{in}_2(x_2) \Rightarrow u_2\} \mid \ldots$$

We consider the standard typing rules (where $i \in \{1, 2\}$):

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 * A_2} \qquad \frac{\Gamma \vdash t : A_1 * A_2}{\Gamma \vdash \pi_i\, t : A_i} \qquad \frac{\Gamma \vdash t : A_i}{\Gamma \vdash \mathsf{in}_i\, t : A_1 + A_2}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma, x : A \vdash u : B}{\Gamma \vdash \mathsf{let}\, x = t \,\mathsf{in}\, u : B} \qquad \frac{\Gamma \vdash t : 0}{\Gamma \vdash \mathsf{case}\, t \,\mathsf{of}\, \{\} : B} \qquad \frac{\Gamma \vdash t : A_1 + A_2 \quad \left(\Gamma, x_i : A_i \vdash u_i : B\right)_{i \in \{1,2\}}}{\Gamma \vdash \mathsf{case}\, t \,\mathsf{of}\, \{\mathsf{in}_1(x_1) \Rightarrow u_1; \mathsf{in}_2(x_2) \Rightarrow u_2\} : B}$$

(Here, a context Γ is a sequence of assignments of types A to variables x.)

In what follows, we use shorthands such as bool = unit + unit, and if-then-else instead of case.

This language is intended to be a generic probabilistic programming language, but so far there is nothing specifically probabilistic about this syntax. Different probabilistic programming languages support distributions over different kinds of structures. Thus, our language is extended according to an 'interface' by specifying type constants and typed term constants $f : A \rightarrow B$. For each term constant $f : A \rightarrow B$, we include a new typing rule,

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash f(t) : B}$$

EXAMPLE 1. *We consider the following examples of interfaces.*

(1) *For probabilistic programming over finite domains, we may have term constants such as* $\mathrm{bernoulli}_{0.5} : \mathrm{unit} \rightarrow \mathrm{bool}$, *intuitively a fair coin toss.*
(2) *For probabilistic programming over real numbers, we may have a type constant* real *and term constants such as* normal : real $*$ real $\rightarrow$ real, *intuitively a parameterized normal distribution, and arithmetic operations such as* (+) : real $*$ real $\rightarrow$ real.
(3) *The main interface of this paper is for random graphs: this has a type constant* vertex *and term constants* new : unit $\rightarrow$ vertex *and* edge : vertex $*$ vertex $\rightarrow$ bool.

(We have kept this language as simple as possible, to focus on the interesting aspects. A practical probabilistic programming language will include other features, which are largely orthogonal, and indeed within our implementation in Haskell (§1.5), programming features like higher order functions and recursion are present and useful. See also the discussion in §2.3.4.)

2.2 Equational Theories and Markov Categories

Section 2.1 introduced a syntax for various probabilistic programming interfaces. The idea is that this is a generic language which applies to different interfaces with different distributions that are implemented in different ways. Rather than considering various ad hoc operational semantics, we study the instances of interfaces by the program equations that they support.

Regardless of the specifics of a particular implementation, we expect basic equational reasoning principles for probabilistic programming to hold, such as the following laws:

$$(\text{let } y=(\text{let } x=t \text{ in } u) \text{ in } t') \equiv (\text{let } x=t \text{ in let } y=u \text{ in } t') \quad (\text{where } x \notin \mathrm{fv}(t')) \tag{7}$$

$$(t, u) \equiv (\text{let } x \ = \ t \text{ in let } y \ = \ u \text{ in } (x, y)) \tag{8}$$

$$(\text{let } x=t \text{ in let } x'=t' \text{ in } u) \equiv (\text{let } x'=t' \text{ in let } x=t \text{ in } u) \quad (\text{where } x \notin \mathrm{fv}(t') \text{ and } x' \notin \mathrm{fv}(t)) \tag{9}$$

$$(\text{let } x \ = \ t' \text{ in } t) \equiv t \quad (\text{where } x \notin \mathrm{fv}(t)) \tag{10}$$

The following law does not always hold, but does hold when v is 'deterministic'.

$$(\text{let } x \ = \ v \text{ in } t) \equiv t[v/x] \tag{11}$$

Equations (9) and (10) say that parts of programs can be re-ordered and discarded, as long as the dataflow is respected. This is a feature of probabilistic programming. For example, coins do not remember the order nor how many times they have been tossed. But these equations would typically not hold in a language with state.

The cleanest way to study equational theories of programs is via a categorical semantics, and for Markov categories have arisen as a canonical setting for categorical probability. Informally, a category is a structure for composition, and this matches the composition structure of let in our language. We also have monoidal structure which allows for the type constructor $A \times B$ and for the compound contexts Γ, comonoid structure which allows duplication of variables, and distributive coproduct structure which allows for the sum types.

Definition 2. *A symmetric monoidal category* $(C, \otimes, I)$ *is a category* C *equipped with a functor* $\otimes : C \times C \to C$ *and an object* I *together with associativity, unit and symmetry structure ([Mac Lane 1998, XI.1.]). A* Markov category *([Fritz 2020]) is a symmetric monoidal category in which*

- *the monoidal unit* I *is a terminal object* $(I = 1)$, *and*
- *every object* X *is equipped with a comonoid* $\Delta_X : X \to X \otimes X$, *compatible with the tensor product* $(\Delta_{X \otimes Y} = (X \otimes \mathrm{swp} \otimes Y) \cdot (\Delta_X \otimes \Delta_Y)$, *where* swp *is the swap map of* C).

A morphism $f : X \to Y$ *in a Markov category is* deterministic *if it commutes with the comonoids:* $(f \otimes f) \cdot \Delta_X = \Delta_Y \cdot f$.

A distributive symmetric monoidal category *(e.g. [Jay 1993; Walters 1989]) is a symmetric monoidal category equipped with chosen finite coproducts such that the canonical maps* $X \otimes Z + Y \otimes Z \to (X + Y) \otimes Z$ *and* $0 \to 0 \otimes Z$ *are isomorphisms. A* distributive Markov category *is a Markov category whose underlying monoidal category is also distributive and whose chosen coproduct injections* $X \to X + Y \leftarrow Y$ *are deterministic. A* distributive category *[Carboni et al. 1993; Cockett 1993] is a distributive Markov category where all morphisms are deterministic.*

A (strict) distributive Markov functor *is a functor* $F : C \to D$ *between distributive Markov categories which strictly preserves the chosen symmetric monoidal, coproduct, and comonoid structures.*

In this paper we mainly focus on functors between distributive Markov categories that strictly preserve the relevant structure, so we elide 'strict'. (Nonetheless, non-strict functors are important, e.g. [Fritz 2020, §10.2] and Prop. 13.)

We interpret the language of Section 2.1 in a distributive Markov category C by interpreting types A and type contexts Γ as objects $[\![A]\!]$ and $[\![\Gamma]\!]$, and typed terms $\Gamma \vdash t : A$ as morphisms $[\![\Gamma]\!] \to [\![A]\!]$. (See e.g. [Pitts 2001] for a general discussion of terms as morphisms.)

In more detail, to give such an interpretation, type constants must first be given chosen interpretations as objects of C. We can then interpret types and contexts using the monoidal and coproduct structure of C. Following this, term constants $f : A \to B$ must be given chosen interpretations as morphisms $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ in C. The interpretation of other terms is made by induction on the structure of typing derivations in a standard manner, using the structure of the distributive Markov category (e.g. [Benton et al. 1992], [Stein 2021, §7.2]). For example,

$$[\![\Gamma, x : A, \Gamma' \vdash x : A]\!] = [\![\Gamma, x : A, \Gamma']\!] \cong [\![\Gamma]\!] \otimes [\![A]\!] \otimes [\![\Gamma']\!] \xrightarrow{!\otimes[\![A]\!]\otimes!} 1 \otimes [\![A]\!] \otimes 1 \cong [\![A]\!]$$

$$[\![\Gamma \vdash \mathrm{let}\, x = t\, \mathrm{in}\, u : B]\!] = [\![\Gamma]\!] \xrightarrow{\Delta_{[\![\Gamma]\!]}} [\![\Gamma]\!] \otimes [\![\Gamma]\!] \xrightarrow{[\![\Gamma]\!]\otimes[\![t]\!]} [\![\Gamma]\!] \otimes [\![A]\!] = [\![\Gamma, x : A]\!] \xrightarrow{[\![u]\!]} [\![B]\!]$$

$$[\![\Gamma \vdash \mathrm{case}\, t\, \mathrm{of}\, \{\mathrm{in}_1(x_1) \Rightarrow u_1; \mathrm{in}_2(x_2) \Rightarrow u_2\} : B]\!] =$$

$$[\![\Gamma]\!] \xrightarrow{\Delta_{[\![\Gamma]\!]}} [\![\Gamma]\!] \otimes [\![\Gamma]\!] \xrightarrow{[\![\Gamma]\!]\otimes[\![t]\!]} [\![\Gamma]\!] \otimes [\![A_1 + A_2]\!] \cong [\![\Gamma, x : A_1]\!] + [\![\Gamma, x : A_2]\!] \xrightarrow{\langle[\![u_1]\!],[\![u_2]\!]\rangle} [\![B]\!]$$

$$[\![\Gamma \vdash f(t) : B]\!] = [\![\Gamma]\!] \xrightarrow{[\![t]\!]} [\![A]\!] \xrightarrow{[\![f]\!]} [\![B]\!]$$

An interpretation in a Markov category induces an equational theory between programs: let $\Gamma \vdash t = u : A$ if $[\![t]\!] = [\![u]\!]$.

Proposition 3 (e.g. [Stein 2021], §7.1). *The equational theory induced by the interpretation in a distributive Markov category, with given interpretations of type and term constants, always includes the equations* (7)–(10), *and also* (11) *whenever* $[\![v]\!]$ *is a deterministic morphism.*

Example 4. *The category* (FinSet, $\times$, 1) *of finite sets is a distributive Markov category. As in any category with products, each object has a unique comonoid structure, and all morphisms are deterministic. This is a good Markov category for interpreting the plain language with no type or term constants. For example,* $[\![\mathrm{bool}]\!]$ *is a set with two elements.*

EXAMPLE 5. *The category* **FinStoch** *has natural numbers as objects and the morphisms are stochastic matrices. In more detail, a morphism $m \to n$ is a matrix in $(\mathbb{R}_{\geq 0})^{m \times n}$ such that each row sums to 1. Composition is by matrix multiplication. The monoidal structure is given on objects by multiplication of numbers, and on morphisms by Kronecker product of matrices. By choosing an enumeration of each finite set, we get a functor* **FinSet** $\to$ **FinStoch** *that converts a function to the corresponding (0/1)-valued matrix. So every object of* **FinStoch** *can be regarded with the comonoid structure from* **FinSet**. *The deterministic morphisms in* **FinStoch** *are exactly the morphisms from* **FinSet** *[Fritz 2020, 10.3].*

This is a good Markov category for interpreting the language with Bernoulli distributions (Ex. 1(1)). We interpret the fair coin as the 1×2 matrix $(0.5, 0.5)$.

We can also give some interpretations for the graph interface (Ex. 1(3)) in **FinStoch**. *For instance, consider random graphs made of two disjoint complete subgraphs, as is typical in a clustering model. We can interpret this by putting $[\![\text{vertex}]\!] = 2$, $[\![\text{edge}]\!] = (\begin{smallmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{smallmatrix})^\top$, and $[\![\text{new}]\!] = (0.5, 0.5)$.*

We look at other examples of distributive Markov categories and interpretations of these interfaces in Sections 2.3.2 and 2.3.3, and then in Sections 4–6.

2.3 Equational Theories and Affine Monads

2.3.1 *Distributive Markov Categories from Affine Monads.* One way to generate equational theories via Markov categories is by considering certain kinds of monads, following Moggi [Moggi 1989].

DEFINITION 6. *A* strong monad *on a category* **A** *with finite products is given by*

- *for each object X, an object $T(X)$;*
- *for each object X, a morphism $\eta_X : X \to T(X)$;*
- *for objects Z, X, Y, a family of functions natural in Z*

$$(\ggg) : \mathbf{A}(Z, T(X)) \times \mathbf{A}(Z \times X, T(Y)) \to \mathbf{A}(Z, T(Y))$$

such that $\ggg$ is associative with unit η.

(There are various different formulations of this structure. When **A** is cartesian closed, as in Defs. 9 and 41, then the bind ($\ggg$) is represented by a morphism ($\ggg$) : $T(X) \times (X \Rightarrow T(Y)) \to T(Y)$, by the Yoneda lemma.)

DEFINITION 7 ([JACOBS 1994; KOCK 1970; LINDNER 1979]). *Given a strong monad T, we say that two morphisms $f : X_1 \to T(X_2)$, $g : X_1 \to T(X_3)$* commute *if*

$$f \ggg ((g \circ \pi_1) \ggg (\eta \circ \langle \pi_2 \circ \pi_1, \pi_2 \rangle)) = g \ggg ((f \circ \pi_1) \ggg (\eta \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle)) : X_1 \to T(X_2 \times X_3).$$

A strong monad is commutative *if all morphisms commute. It is* affine *if $T(1) \to 1$ is an isomorphism.*

The Kleisli category $\mathrm{Kl}(T)$ of a strong monad T has the same objects as **A**, but the morphisms are different: $\mathrm{Kl}(T)(A, B) = \mathbf{A}(A, T(B))$. There is a functor $J : \mathbf{A} \to \mathrm{Kl}(T)$, given on morphisms by composing with η (e.g. [Mac Lane 1998, §VI.5], [Moggi 1989]).

PROPOSITION 8. *Let T be a strong monad on a category* **A**. *If T is commutative and affine and* **A** *has finite products, then the Kleisli category $\mathrm{Kl}(T)$ has a canonical structure of a Markov category. Furthermore, if* **A** *is distributive, then $\mathrm{Kl}(T)$ can be regarded as a distributive Markov category.*

PROOF NOTES. The Markov structure follows [Fritz 2020, §3]. Since T is commutative, the product structure of **A** extends to a symmetric monoidal structure on $\mathrm{Kl}(T)$. Since $T(1) = 1$, the monoidal unit (1) is terminal in $\mathrm{Kl}(T)$. Every object in **A** has a comonoid structure, and this is extended to $\mathrm{Kl}(T)$ via J. The morphisms in the image of J are deterministic, although this need not be a full characterization of determinism.

For the distributive structure, recall that J preserves coproducts and indeed it has a right adjoint. Hence, the coproduct injections will be deterministic.   □

We can thus interpret the language of Section 2.1 using any strong monad, interpreting the types A as objects $[\![A]\!]$ of $\mathbf{A}$, and a term $\Gamma \vdash t : A$ as a morphism $[\![t]\!] : [\![\Gamma]\!] \rightarrow T([\![A]\!])$. This interpretation matches Moggi's interpretation of the language of Section 2.1 in a strong monad.

2.3.2 Example Affine Monad: Distribution Monad.

DEFINITION 9 (E.G. [JACOBS 2016], §4.1). *The distribution monad $\mathcal{D}$ on* **Set** *is defined as follows:*
- *On objects: each set X is mapped to the set of all finitely-supported discrete probability measures on X, that is, all functions $p : X \rightarrow \mathbb{R}$ that are non-zero for only finitely many elements and satisfy $\sum_{x \in X} p(x) = 1$.*
- *The unit $\eta_X : X \rightarrow \mathcal{D}(X)$ maps $x \in X$ to the indicator function $\lambda y.\, [y = x]$, i.e. the Dirac distribution δ_x.*
- *The bind function ($\gg\!\!=$) is defined as follows:*

$$(f \gg\!\!= g)(z)(y) = \sum_{x \in X} f(z)(x) \cdot g(z, x)(y)$$

By the standard construction for strong monads, each morphism $f : X \rightarrow Y$ gets mapped to $\mathcal{D}f : \mathcal{D}X \rightarrow \mathcal{D}Y$, that is, the pushforward in this case: $\mathcal{D}f(p)(y) = \sum_{x \in f^{-1}(y)} p(x)$.

Consider the language with no type constants, and just the term constant $\mathrm{bernoulli}_{0.5}$ (Ex. 1(1)). This can be interpreted in the distribution monad. Since every type A is interpreted as a finite set $[\![A]\!]$, and every context Γ as a finite set $[\![\Gamma]\!]$, a term $\Gamma \vdash t : A$ is interpreted as a function $[\![\Gamma]\!] \rightarrow \mathcal{D}[\![A]\!]$. To give a Kleisli morphism between finite sets is to give a stochastic matrix, and so the induced equational theory is the same as the interpretation in **FinStoch** (Ex. 5).

2.3.3 Example Affine Monad: Giry Monad. We recall some rudiments of measure-theoretic probability.

DEFINITION 10. *A σ-algebra on a set is a non-empty collection of subsets that contains the empty set and is closed under countable unions and complements. A measurable space is a pair (X, Σ) of a set and a σ-algebra on it. A measurable function $(X, \Sigma_X) \rightarrow (Y, \Sigma_Y)$ is a function $f : X \rightarrow Y$ such that $f^{-1}(U) \in \Sigma_X$ for all $U \in \Sigma_Y$.*

A probability measure on a measurable space (X, Σ) is a function $\mu : \Sigma \rightarrow [0, 1]$ that has total mass 1 ($\mu(X) = 1$) and that is σ-additive: $\mu(\biguplus_{i=1}^{\infty} U_i) = \sum_{i=1}^{\infty} \mu(U_i)$ for any sequence of disjoint U_i.

Examples of measurable spaces include: the finite sets X equipped with their powerset σ-algebras; the unit interval $[0, 1]$ equipped with its Borel σ-algebra, which is the least σ-algebra containing the open sets. Examples of probability measures include: discrete probability measures (Def. 9); the uniform measure on $[0, 1]$; the Dirac distribution $\delta_x(U) = [x \in U]$.

The product of two measurable spaces $(X, \Sigma_X) \times (Y, \Sigma_Y) = (X \times Y, \Sigma_X \otimes \Sigma_Y)$ comprises the product of sets with the least σ-algebra making the projections $X \leftarrow X \times Y \rightarrow Y$ measurable. The category of measurable spaces and measurable functions is a distributive category.

A *probability kernel* between measurable spaces (X, Σ_X) and (Y, Σ_Y) is a function $k : X \times \Sigma_Y \rightarrow [0, 1]$ that is measurable in the first argument and that is σ-additive and has mass 1 in the second argument.

To compose probability kernels, we briefly recall Lebesgue integration. Consider a measurable space (X, Σ_X), a measure $\mu : \Sigma_X \rightarrow [0, 1]$, and a measurable function $f : X \rightarrow [0, 1]$. If f is a simple function, i.e. $f(x) = \sum_{i=1}^{m} r_i \cdot [x \in U_i]$ for some $m, r_i \in [0, 1]$, and $U_i \in \Sigma_X$, the Lebesgue integral $\int f \, d\mu = \int f(x) \, \mu(dx) \in [0, 1]$ is defined to be $\sum_{i=1}^{m} r_i \times \mu(U_i)$. If f is not a simple function, there exists a sequence of increasing simple functions $f_1, f_2, \ldots : X \rightarrow [0, 1]$ such that $\sup_k f_k(x) = f(x)$

(for example, by taking $f_k(x) \stackrel{\text{def}}{=} \lfloor 10^k f(x) \rfloor / 10^k$). In that case, the integral is defined to be the limit of the integrals of the f_k's (which exists by monotone convergence).

Probability kernels can be equivalently formulated as morphisms $X \to \mathcal{G}(Y)$, where $\mathcal{G}$ is the Giry monad:

DEFINITION 11 ([GIRY 1980]). *The Giry monad $\mathcal{G}$ is a strong monad on the category* **Meas** *of measurable spaces given by*

- $\mathcal{G}(X)$ *is the set of probability measures on X, with the least σ-algebra making $\int f \, \mathrm{d}(-) : \mathcal{G}(X) \to [0, 1]$ measurable for all measurable $f : X \to [0, 1]$;*
- *the unit η maps x to the Dirac distribution δ_x;*
- *the bind is given by composing kernels:*

$$(k \ggeq l)(z, U) = \int l((z, x), U) \, k(z, \mathrm{d}x). \tag{12}$$

PROPOSITION 12. *The monad $\mathcal{G}$ is commutative and affine.*

PROOF NOTES. Commutativity boils down to Fubini's theorem for reordering integrals and affineness is marginalization (since probability measures have mass 1). See also [Jacobs 2018]. □

Consider the real-numbers language (Ex. 1(2)). Let $\llbracket \text{real} \rrbracket = \mathbb{R}$, with the Borel sets, and interpret normal as the normal probability measure on $\mathbb{R}$. The basic arithmetic operations are all measurable.

Among the following three programs

$$\text{let } x = \text{normal}(0, 1) \text{ in } x + x \tag{13}$$

$$\text{let } x = \text{normal}(0, 1) \text{ in let } y = \text{normal}(0, 1) \text{ in } x + y \tag{14}$$

$$\text{normal}(0, 1) + \text{normal}(0, 1) \tag{15}$$

the programs (14) and (15) denote the same normal distribution with variance 2, whereas (13) denotes a distribution with variance 4. Notice that we cannot use (11) to equate all the programs, because $\llbracket \text{normal} \rrbracket$ is not deterministic.

We can also interpret the Bernoulli language (Ex. 1(1)) in the Giry monad; this interpretation gives the same equational theory as the interpretation in **FinStoch** and in the distribution monad in Section 2.3.2.

We can also give some interpretations for the graph interface (Ex. 1(3)) in the Giry monad. For an informal example, consider the geometric example from Section 1.1, let $\llbracket \text{vertex} \rrbracket = S_2$ (the sphere), and define $\llbracket \text{new} \rrbracket$ to be the uniform distribution on the sphere. (See also Section 5.2.)

2.3.4 Affine Monads from Distributive Markov Categories. The following result, a converse to Proposition 8, demonstrates that the new notion of distributive Markov category (Def. 2) is a canonical one, and emphasizes the close relationship between semantics with distributive Markov categories and semantics with commutative affine monads.

PROPOSITION 13. *Let* **C** *be a small distributive Markov category. Then, there is a distributive category* **A** *with a commutative affine monad T on it and a full and faithful functor* **C** $\to \mathrm{Kl}(T)$ *that preserves symmetric monoidal structure, comonoids, and sums.*

PROOF NOTES. Our proof is essentially a recasting of [Power 2006b, §7] to this different situation, as follows.

Let $\mathbf{C}_{\text{det}}$ be the wide subcategory of **C** comprising the deterministic morphisms, and write $J : \mathbf{C}_{\text{det}} \to \mathbf{C}$ for the identity-on-objects inclusion functor. Note that $\mathbf{C}_{\text{det}}$ is a distributive category. We would like to exhibit **C** as the Kleisli category for a monad on $\mathbf{C}_{\text{det}}$, but this might not be possible:

intuitively, $\mathbf{C}_{\mathrm{det}}$ might be too small for the monad to exist. Instead, we first embed $\mathbf{C}_{\mathrm{det}}$ in a larger category $\mathbf{A}$ and construct a monad on $\mathbf{A}$.

The main construction in our proof is the idea that if $\mathbf{X}$ is a small distributive monoidal category, then the category $\mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$ of finite-product-preserving functors is such that

- $\mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$ is cocomplete and moreover total ([Street and Walters 1978]) as a category;
- $\mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$ admits a distributive monoidal structure;
- the Yoneda embedding $\mathbf{X} \to [\mathbf{X}^{\mathrm{op}}, \mathbf{Set}]$, which is full and faithful, factors through $\mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$, and this embedding $\mathbf{X} \to \mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$ preserves finite sums and is strongly monoidal;
- the Yoneda embedding exhibits $\mathrm{FP}(\mathbf{X}^{\mathrm{op}}, \mathbf{Set})$ as a free colimit completion of $\mathbf{X}$ as a monoidal category that already has finite coproducts.

So we let $\mathbf{A} = \mathrm{FP}(\mathbf{C}_{\mathrm{det}}^{\mathrm{op}}, \mathbf{Set})$ comprise the finite-product-preserving functors $\mathbf{C}_{\mathrm{det}}^{\mathrm{op}} \to \mathbf{Set}$. This is a distributive category. To get a monad on $\mathbf{A}$, we note that since $\mathrm{FP}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$ has finite coproducts and $\mathbf{C}_{\mathrm{det}} \to \mathbf{C} \to \mathrm{FP}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$ preserves finite coproducts and is monoidal, the monoidal structure induces a canonical colimit-preserving monoidal functor $J_! : \mathrm{FP}(\mathbf{C}_{\mathrm{det}}^{\mathrm{op}}, \mathbf{Set}) \to \mathrm{FP}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$. Any colimit-preserving functor $J_!$ out of a total category has a right adjoint J^*, and hence a monoidal monad $(J^* J_!)$ is induced on $\mathbf{A}$.

It remains for us to check that the embedding $\mathbf{C} \to \mathrm{FP}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$ factors through the comparison functor $\mathrm{Kl}(J^* J_!) \to \mathrm{FP}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$, which follows from the fact that $J : \mathbf{C}_{\mathrm{det}} \to \mathbf{C}$ is identity on objects. □

As an aside, we note that, although our simple language in Section 2.1 did not include higher-order functions, the category $\mathbf{A}$ constructed in the proof of Proposition 13 is cartesian closed, and since the embedding is full and faithful, this shows that higher-order functions would be a conservative extension of our language. Indeed, this kind of conservativity result was part of the motivation of [Power 2006b]. For the same reason, inductive types (lists, and so on) would also be a conservative extension. We leave conservativity with other language features for future work. Recursion in probabilistic programming is still under investigation [Ehrhard et al. 2018; Goubault-Larrecq et al. 2021; Jia et al. 2021; Matache et al. 2022; Vákár et al. 2019]; there is also the question of conservativity with respect to combining Markov categories, e.g. combining real number distributions ((1)–(2)) with graph programming ((3)–(4)).

2.4 Bernoulli Bases, Numerals and Observation

Although an interface may have different type constants, it will always have the 'numeral' types, sometimes called 'finite' types:

$$0 \quad \mathrm{unit} \quad \mathrm{bool} = \mathrm{unit} + \mathrm{unit} \quad \mathrm{unit} + \mathrm{unit} + \mathrm{unit} \quad \ldots$$

For probabilistic programming languages, there is a clear expectation of what will happen when we run a program of type bool: it will randomly produce either true or false, each with some probability. Similarly for other numeral types. For type constants, we might not have evident notions of observation or expected outcomes. But for numeral types, it should be routine. We now make this precise via the notion of Bernoulli base.

On the semantic side, distributive Markov categories will always have 'numeral' objects

$$0 \quad 1 \quad 2 \overset{\mathrm{def}}{=} 1 + 1 \quad 3 \overset{\mathrm{def}}{=} 1 + 1 + 1 \quad \ldots$$

For any type A formed without type constants, and any Markov category, we have that $[\![A]\!] \cong n$ for some numeral object. Any equational theory for the programming language induces in particular an equational theory for the sub-language without any type constants.

PROPOSITION 14. *For any distributive Markov category* C*, let* $C_\mathbb{N}$ *be the category whose objects are natural numbers, and where the morphisms are the morphisms in* C *between the corresponding numeral objects. This is again a distributive Markov category.*

EXAMPLE 15.   (1) $\mathbf{FinSet}_\mathbb{N} = \mathbf{Set}_\mathbb{N}$ *is equivalent to* **FinSet** *as a category.*
(2) *For the finite distributions and the Giry monad (§2.3.2–2.3.3),* $\mathrm{Kl}(\mathcal{D})_\mathbb{N} \simeq \mathrm{Kl}(\mathcal{G})_\mathbb{N} \simeq \mathbf{FinStoch}$.

Recall that a functor is *faithful* if it is injective on hom-sets.

DEFINITION 16. *A* Bernoulli base *for a distributive Markov category* C *is a faithful distributive Markov functor* $\Psi : C_\mathbb{N} \rightarrowtail \mathbf{FinStoch}$.

Thus, for any distributive Markov category with a Bernoulli base, for any closed term $\vdash t : A$ of numeral type ($[\![A]\!] = n$), we can regard its interpretation $[\![t]\!] : 1 \to n$ as nothing but a probability distribution $\Psi([\![t]\!])$ on n outcomes. This is the case even if t uses term constants and has intermediate subterms using type constants.

EXAMPLE 17. *All the examples seen so far can be given Bernoulli bases. In fact, for* **FinStoch***,* $\mathrm{Kl}(\mathcal{D})$ *and* $\mathrm{Kl}(\mathcal{G})$*, the functor* $\Psi : C_\mathbb{N} \rightarrowtail \mathbf{FinStoch}$ *is an isomorphism of distributive Markov categories.*

When Ψ is an isomorphism of categories, that means that *all* the finite probabilities are present in C. This is slightly stronger than we need in general. For instance, when C = **FinSet**, there is a unique Bernoulli base $\Psi : \mathbf{FinSet}_\mathbb{N} \rightarrowtail \mathbf{FinStoch}$, taking a function to a 0/1-valued matrix, but it is not full. We could also consider variations on **FinStoch**. For example, consider the subcategory **Fin$\mathbb{Q}$Stoch** of **FinStoch** where the matrices are rational-valued; this has a Bernoulli base that is not an isomorphism.

2.5 Quotients of Distributive Markov Categories

We provide a new, general method for constructing a Bernoulli-based Markov category out of a distributive Markov category. Our construction is a categorical formulation of the notion of contextual equivalence.

Recall that, in general, contextual equivalence for a programming language starts with a notion of basic observation for closed programs at ground types. We then say that programs $\Gamma \vdash t, u : A$ at other types are *contextually equivalent* if for every context C with $\vdash C[t], C[u] : n$, for some ground type n, we have that $C[t]$ and $C[u]$ satisfy the same observations. In the categorical setting, the notion of observation is given by a distributive Markov functor $C_\mathbb{N} \to \mathbf{FinStoch}$, and the notion of context C is replaced by suitable morphisms (h, k below). We now introduce a quotient construction that will be key in showing that every graphon arises from a distributive Markov category (Corollary 26), via Theorem 23. We note that this is a general new method for building Markov categories.

PROPOSITION 18. *Let* C *be a distributive Markov category, and let* $\Psi : C_\mathbb{N} \to \mathbf{FinStoch}$ *be a distributive Markov functor. Suppose that for every object* $X \in C$*, either* $X = 0$ *or there exists a morphism* $1 \to X$*. Then, there is a distributive Markov category* $C/_\Psi$ *with a Bernoulli base, equipped with a distributive Markov functor* $C \to C/_\Psi$ *and a factorization of distributive Markov functors* $\Psi = C_\mathbb{N} \to (C/_\Psi)_\mathbb{N} \rightarrowtail \mathbf{FinStoch}$.

PROOF. Define an equivalence relation $\sim$ on each hom-set $C(X, Y)$, by $f \sim g : X \to Y$ if

$$\forall Z, n. \forall h : 1 \to X \otimes Z. \forall k : Y \otimes Z \to n. \quad \Psi(k \cdot (f \otimes Z) \cdot h) = \Psi(k \cdot (g \otimes Z) \cdot h) \text{ in } \mathbf{FinStoch}(1, n).$$

Informally, our equivalence relation considers all ways of generating X's via precomposition (h), all ways for testing Y's via postcomposition (k), and all ways of combining with some ancillary

data (Z). It is essential that we consider all these kinds of composition in order for the quotient category to have the categorical structure.

It is immediate that composition of morphisms respects $\sim$, and hence we have a category: the objects are the same as $\mathbf{C}$, and the morphisms are $\sim$-equivalence classes. This is our category $\mathbf{C}/_{\Psi}$.

It is also immediate that if $f \sim g$ and $f' \sim g'$ then $(f \otimes f') \sim (g \otimes g')$. Thus, $\mathbf{C}/_{\Psi}$ is a monoidal category.

For the coproduct structure, we must show that if $f \sim g : X \to Y$ and $f' \sim g' : X' \to Y'$ then $(f + f') \sim (g + g') : X + X' \to Y + Y'$. We proceed by noting that since we have morphisms $x : 1 \to X$ and $x' : 1 \to X'$, as well as terminal morphisms $X \to 1$ and $X' \to 1$, we have that $X + X'$ is a retract of $X \otimes X' \otimes 2$, with the section and retraction given by:

$$X + X' \xrightarrow{X \otimes x' + x \otimes X} X \otimes X' + X \otimes X' \cong X \otimes X' \otimes 2 \quad X \otimes X' \otimes 2 \cong X \otimes X' + X \otimes X' \xrightarrow{X \otimes ! + ! \otimes X} X + X'$$

Thus, by composing with this retract, it suffices to check that $(f \otimes f' \otimes 2) \sim (g \otimes g' \otimes 2)$, which we have already shown.

The functor $\Psi : \mathbf{C}_{\mathbb{N}} \to \mathbf{FinStoch}$ clearly factors through $(\mathbf{C}/_{\Psi})_{\mathbb{N}}$, but it remains to check that the functor $(\mathbf{C}/_{\Psi})_{\mathbb{N}} \to \mathbf{FinStoch}$ is now faithful (Bernoulli base). So suppose that $\Psi(f) = \Psi(g)$. To show that $f \sim g : 1 \to m$, we consider $h : 1 \to 1 \otimes Z$, and $k : m \otimes Z \to n$. We must show that $\Psi(k \cdot (f \otimes Z) \cdot h) = \Psi(k \cdot (g \otimes Z) \cdot h)$. Since $h = 1 \otimes h'$, for some $h' : 1 \to Z$, we have

$$\Psi(k \cdot (f \otimes Z) \cdot h) = \Psi(k \cdot (m \otimes h') \cdot f) = \Psi(k \cdot (m \otimes h')) \cdot \Psi(f)$$
$$= \Psi(k \cdot (m \otimes h')) \cdot \Psi(g) = \Psi(k \cdot (m \otimes h') \cdot g) = \Psi(k \cdot (g \otimes Z) \cdot h).$$

$\square$

3 FROM PROGRAM EQUATIONS TO GRAPHONS

The graph interface for the probabilistic programming language (Ex. 1(3)) does not have one fixed equational theory. Rather, we want to consider different equational theories for the language, corresponding to different implementations of the interface for the graph (see also §1.2). We now show how the different equational theories for the graph language each give rise to a graphon, by building adjacency matrices for finite graphs (shown in (18)). To do this, we set up the well-behaved equational theories (§2.4), recall the connection between graphons and finite random graphs (§3.1), and then show the main result (§3.2, Theorem 23).

3.1 Graphons as Consistent and Local Random Graph Models

For all $n \geq 1$, let $[n]$ be the set $\{1, \ldots, n\}$. (We sometimes omit the square brackets, when it is clear.) A simple undirected graph g with n nodes can be represented by its adjacency matrix $A_g \in 2^{[n]^2}$ such that $A_g(i, i) = 0$ and $A_g(i, j) = A_g(j, i)$. Henceforth, we will assume that finite graphs are simple and undirected, unless otherwise stated. A random finite graph, then, has a probability distribution in $\mathcal{D}\big(2^{[n]^2}\big)$ that only assigns non-zero probability to adjacency matrices.

DEFINITION 19 (E.G. [Lovász 2012, §11.2.1]). *A random graph model is a sequence of distributions of random finite graphs of the form:*

$$p_1 \in \mathcal{D}\big(2^{[1]^2}\big), \, p_2 \in \mathcal{D}\big(2^{[2]^2}\big), \, \ldots, \, p_n \in \mathcal{D}\big(2^{[n]^2}\big), \ldots$$

We say such a sequence is

- exchangeable *if each of its elements is invariant under permuting nodes: for every n and bijection $\sigma : [n] \to [n]$, we have $\mathcal{D}\big(2^{(\sigma^2)}\big)(p_n) = p_n$ (where $2^{(\sigma^2)} : 2^{[n]^2} \to 2^{[n]^2}$ is the function that permutes the rows and columns according to σ; we are regarding $\mathcal{D}$ as a covariant functor, Def. 9, and $2^{(-)}$ as a contravariant functor);*

- consistent *if the sequence is related by marginals: for every n and for the inclusion function* $\iota \colon [n] \hookrightarrow [n+1]$, $\mathcal{D}\big(2^{(\iota^2)}\big)(p_{n+1}) = p_n$ *(where* $2^{(\iota^2)} : 2^{([n+1]^2)} \to 2^{[n]^2}$ *is the evident projection);*
- local *if the subgraphs are independent: if* $A \subseteq [n]$ *and* $B \subseteq [n]$ *are disjoint, then we have an injective function* $\jmath \colon A^2 + B^2 \hookrightarrow [n]^2$, *and* $\mathcal{D}(2^\jmath)(p_n) \in \mathcal{D}\big(2^{(A^2)} \times 2^{(B^2)}\big)$ *is a product measure* $p_A \otimes p_B$ *(where* $2^\jmath : 2^{[n]^2} \to 2^{(A^2)} \times 2^{(B^2)}$ *is the evident pairing of projections).*

DEFINITION 20 (E.G. [Lovász 2012]). *A* graphon W *is a symmetric measurable function* $W :$ $[0,1]^2 \to [0,1]$.

Given a graphon W, we can generate a finite simple undirected graph g with vertex set $[n]$ by sampling n points $x_1, \ldots, x_n$ uniformly from $[0,1]$ and, then, including the edge (i,j) with probability $W(x_i, x_j)$ for all $1 \le i, j \le n$. This sampling procedure defines a distribution over finite graphs: the probability $p_{W,n}(A_g)$ of the graph $g = ([n], E)$ is:

$$\int_{[0,1]^n} \prod_{(i,j) \in E} W(x_i, x_j) \prod_{(i,j) \notin E} \big(1 - W(x_i, x_j)\big) \, \mathrm{d}(x_1 \ldots x_n) \tag{16}$$

PROPOSITION 21 ([Lovász AND Szegedy 2006], [Lovász 2012, §11.2]). *Every graphon generates an exchangeable, consistent, and local random graph model, by the sampling procedure of* (16). *Conversely, every exchangeable, consistent, and local random graph model is of the form* $p_{W,n}$ *for some graphon* W.

NOTE. There are various methods for constructing W from an exchangeable, consistent and local random graph model, however all are highly non-trivial. A general idea is that W is a kind of limit object. For examples see e.g. [Lovász and Szegedy 2006, §11.3] or [Tao 2013]. Fortunately though, we will not need explicit constructions in this paper.   □

3.2 Theories of Program Equivalence Induce Graphons

In this section we consider the instance of the generic language with the graph interface (Ex. 1(3)):

$$\text{vertex} \qquad \text{new} : \text{unit} \to \text{vertex} \qquad \text{edge} : \text{vertex} * \text{vertex} \to \text{bool}$$

We consider a theory of program equivalence, i.e. a distributive Markov category with a distinguished object $[\![\text{vertex}]\!]$ and morphisms $[\![\text{new}]\!] : 1 \to [\![\text{vertex}]\!]$ and $[\![\text{edge}]\!] : [\![\text{vertex}]\!] \otimes [\![\text{vertex}]\!] \to 1 + 1$. We make two assumptions about the theory:

- The graphs are simple and undirected:

  $$x \colon \text{vertex} \vdash \text{edge}(x, x) \equiv \text{false} \qquad x, y \colon \text{vertex} \vdash \text{edge}(x, y) \equiv \text{edge}(y, x) \tag{17}$$

  and edge is deterministic.
- The theory is Bernoulli based (§2.4).

For each $n \in \mathbb{N}$, we can build a random graph with n vertices as follows. We consider the following program t_n:

$$\vdash \text{let } x_1 = \text{new}() \text{ in} \ldots \text{let } x_n = \text{new}() \text{ in} \begin{pmatrix} \text{edge}(x_1, x_1) & \ldots & \text{edge}(x_1, x_n) \\ \vdots & & \vdots \\ \text{edge}(x_n, x_1) & \ldots & \text{edge}(x_n, x_n) \end{pmatrix} : \text{bool}^{(n^2)} \tag{18}$$

(Here we use syntactic sugar, writing a matrix instead of iteratively using pairs.)

Because the equational theory is Bernoulli-based, the interpretation $[\![t_n]\!]$ induces a probability distribution $\Psi[\![t_n]\!]$ on $2^{(n^2)}$. For clarity, we elide Ψ in what follows, since it is faithful.

PROPOSITION 22. *Each random matrix in* (18) *is a random adjacency matrix, i.e. a random graph.*

PROOF NOTE. This follows from (17).   □

THEOREM 23. *For any Bernoulli-based equational theory, the random graph model* $(\llbracket t_n \rrbracket)_n$ *in* (18) *is exchangeable, consistent, and local. Thus, the equational theory induces a graphon.*

PROOF. We denote the matrix in (18) by $(\mathsf{edge}(x_i, x_j))_{i,j \in [n]}$.

Exchangeability. We show that the distribution $\llbracket t_n \rrbracket$ is invariant under relabeling the nodes. By commutativity of the let construct (9), the program

$$t_n^\sigma \stackrel{\text{def}}{=} \mathsf{let}\ x_{\sigma^{-1}(1)}\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_{\sigma^{-1}(n)}\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]}$$

satisfies $\llbracket t_n^\sigma \rrbracket = \llbracket t_n \rrbracket$. Hence, $\mathcal{D}(2^{\sigma^2})(\llbracket t_n \rrbracket) = \llbracket t_n^\sigma \rrbracket = \llbracket t_n \rrbracket$, for every n and bijection $\sigma \colon [n] \to [n]$.

Consistency. We define a macro subm_I in the graph programming language to extract a submatrix at the index set $I \subseteq [n]$: we have the (definitional) equality

$$\mathsf{subm}_I((a_{i,j})_{i,j \in [n]}) \stackrel{\text{def}}{=} (a_{i,j})_{i,j \in I} \quad \text{for } I \subseteq [n].$$

We need to show that, if we delete the last node from a graph sampled from $\llbracket t_{n+1} \rrbracket$, the resulting graph has distribution $\llbracket t_n \rrbracket$. This amounts to the affineness property (10), as follows. Let $g \sim \llbracket t_{n+1} \rrbracket$ be a random graph, and let $g' \stackrel{\text{def}}{=} g|_{[n]}$ be the graph obtained by deleting the last node from g. Then clearly, the adjacency matrix of g' is the adjacency matrix of g where the last row and column have been removed, *i.e.* g' is sampled from the interpretation of the program:

$$\begin{aligned}
t' &\stackrel{\text{def}}{=} \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{let}\ x_{n+1}\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{subm}_{[n]}\big((\mathsf{edge}(x_i, x_j))_{i,j \in [n+1]}\big) \\
&\equiv \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{let}\ x_{n+1}\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]} \\
&\equiv \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]} \qquad \text{(by (10))} \\
&\equiv t_n.
\end{aligned}$$

Locality. Without loss of generality (by exchangeability and consistency), we need to show that for every random graph $g \sim \llbracket t_n \rrbracket$ and $1 < k < n$, the subgraphs g_{A_k}, g_{B_k} respectively induced by the sets $A_k \stackrel{\text{def}}{=} [k]$ and $B_k \stackrel{\text{def}}{=} \{k+1, \ldots, n\}$ are independent as random variables. Let $\jmath$ be the injection $\jmath \colon A_k^2 + B_k^2 \hookrightarrow n^2$, and $g' \sim \mathcal{D}(2^\jmath)(\llbracket t_n \rrbracket) \in \mathcal{D}(2^{(A_k^2)} \times 2^{(B_k^2)})$. We want to show that g' and $(g_{A_k}, g_{B_k}) \sim \llbracket t_k \rrbracket \otimes \llbracket t_{n-k} \rrbracket$ (by consistency) are equal in distribution. Modulo α-renaming, (g_{A_k}, g_{B_k}) is sampled from the interpretation of the program:

$$\begin{aligned}
t' &\stackrel{\text{def}}{=} \big(\mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_k\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in [k]}, \\
&\qquad \mathsf{let}\ x_{k+1}\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{k+1 \le i, j \le n}\big) \\
&\equiv \mathsf{let}\ u_1\ =\ (\mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_k\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in A_k})\ \mathsf{in} \\
&\qquad \mathsf{let}\ u_2\ =\ (\mathsf{let}\ x_{k+1}\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in B_k})\ \mathsf{in}\ (u_1, u_2) \qquad \text{(by (8))} \\
&\equiv \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_k\ =\ \mathsf{new}()\ \mathsf{in}\ \mathsf{let}\ x_{k+1}\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in} \qquad \text{((7),(9))} \\
&\qquad \mathsf{let}\ u_1\ =\ \mathsf{subm}_{A_k}\big((\mathsf{edge}(x_i, x_j))_{i,j \in [n]}\big)\ \mathsf{in}\ \mathsf{let}\ u_2\ =\ \mathsf{subm}_{B_k}\big((\mathsf{edge}(x_i, x_j))_{i,j \in [n]}\big)\ \mathsf{in}\ (u_1, u_2) \\
&\equiv \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in} \\
&\qquad \mathsf{let}\ t\ =\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]}\ \mathsf{in}\ \mathsf{let}\ u_1\ =\ \mathsf{subm}_{A_k}(t)\ \mathsf{in}\ \mathsf{let}\ u_2\ =\ \mathsf{subm}_{B_k}(t)\ \mathsf{in}\ (u_1, u_2) \quad \text{(by (11))} \\
&\equiv \mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in} \\
&\qquad \mathsf{let}\ t\ =\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]}\ \mathsf{in}\ \big(\mathsf{subm}_{A_k}(t),\ \mathsf{subm}_{B_k}(t)\big) \qquad\qquad\qquad\qquad\qquad \text{(by (8))} \\
&\equiv \mathsf{let}\ t\ =\ \big(\mathsf{let}\ x_1\ =\ \mathsf{new}()\ \mathsf{in} \ldots \mathsf{let}\ x_n\ =\ \mathsf{new}()\ \mathsf{in}\ (\mathsf{edge}(x_i, x_j))_{i,j \in [n]}\big)\ \mathsf{in} \\
&\qquad \big(\mathsf{subm}_{A_k}(t),\ \mathsf{subm}_{B_k}(t)\big) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by (7))}
\end{aligned}$$

and $g' \sim \mathcal{D}(2^J)(\llbracket t_n \rrbracket)$ is indeed sampled from the interpretation of the latter program, which yields the result.   □

4 FROM GRAPHONS TO PROGRAM EQUATIONS

In Section 3, we showed how a distributive Markov category modelling the graph interface (Ex. 1(3)) gives rise to a graphon. In this section, we establish a converse: every graphon arises in this way (Corollary 26). Theorem 25 will establish slightly more: there is a 'generic' distributive Markov category (§4.1) modelling the graph interface whose Bernoulli-based quotients are in precise correspondence with graphons (§4.2). This approach also suggests an operational way of implementing the graph interface for any graphon (§4.3).

4.1 A Generic Distributive Markov Category for the Graph Interface

We construct this generic category in two steps. We first create a distributive Markov category, actually a distributive category, $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$, that supports (vertex, edge). We then add new using the monoidal indeterminates method of [Hermida and Tennent 2012].

4.1.1 Step 1: A Distributive Category with edge. We first define a distributive category that supports (vertex, edge). Let $\mathbf{G}$ be the category of finite graphs and functions that preserve and reflect the edge relation. That is, a morphism $f : g \to g'$ is a function $f : V_g \to V'_g$ such that for all $v, w \in V_g$ we have $E_g(v, w)$ if and only if $E_g(f(v), f(w))$.

Recall (e.g. [Hu and Tholen 1995]) that the free finite coproduct completion of a category $\mathbf{C}$, $\mathsf{Fam}(\mathbf{C})$ is given as follows. The objects of $\mathsf{Fam}(\mathbf{C})$ are sequences $(X_1 \ldots X_n)$ of objects of $\mathbf{C}$, and the morphisms $(X_1 \ldots X_m) \to (Y_1 \ldots Y_n)$ are pairs $(f, \{f_i\}_{i=1}^m)$ of a function $f : m \to n$ and a sequence of morphisms $f_1 : X_1 \to Y_{f(1)}, \ldots, f_m : X_m \to Y_{f(m)}$ in $\mathbf{C}$.

We consider the category $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$. Let $\llbracket \mathrm{vertex} \rrbracket = (1)$, the singleton sequence comprising the one-vertex graph.

> PROPOSITION 24. (1) *The free coproduct completion* $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$ *is a distributive category, with the product* $\llbracket \mathrm{vertex} \rrbracket^n$ *being the sequence of all graphs with n vertices. In particular,* $\llbracket \mathrm{vertex} \rrbracket^2$ *is a sequence with two components, the complete graph and the edgeless graph with two vertices.*
> (2) *Let* edge $: \llbracket \mathrm{vertex} \rrbracket \times \llbracket \mathrm{vertex} \rrbracket \to 1 + 1$ *be the morphism* $(\mathrm{id}, \{!, !\})$, *intuitively returning true for the edge, and false for the edgeless graph. Here the terminal object 1 of* $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$ *is the singleton tuple of the empty graph. This interpretation satisfies* (17).

PROOF NOTES. Item (1) follows from [Hu and Tholen 1995], which shows that limits in $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$ amount to "multi-colimits" in $\mathbf{G}$. For example, the family of all graphs with n vertices is a multi-coproduct of the one-vertex graph in $\mathbf{G}$, hence forms a product in $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$. Item (2) is then a quick calculation. All morphisms in $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})$ are deterministic.   □

4.1.2 Step 2: Adjoining new. In Section 4.1.1, we introduced a distributive category that interprets the interface (vertex, edge). But it does not support new, and indeed there are no morphisms $1 \to \llbracket \mathrm{vertex} \rrbracket$. To additionally interpret (new), we freely adjoin it. We essentially use the 'monoidal indeterminates' method of Hermida and Tennent [Hermida and Tennent 2012] to do this. Their work was motivated by semantics of dynamic memory allocation, but has also been related to quantum phenomena [Andrés-Martínez et al. 2022; Huot and Staton 2018] and to categorical gradient/probabilistic methods [Cruttwell et al. 2021; Fong et al. 2021; Shiebler 2021], where it is known as the 'para construction'. It is connected to earlier methods for the action calculus [Pavlović 1997].

Let **FinSetInj** be the category of finite sets and injections. It is a monoidal category with the disjoint union monoidal structure (e.g. [Fiore 2005; Power 2006a]). Consider the functor

$J : \textbf{FinSetInj}^{\text{op}} \to \textsf{Fam}(\mathbf{G}^{\text{op}})$, with $J(n) = [\![\text{vertex}]\!]^n$, and where the functorial action is by exchange and projection. This is a strong monoidal functor. (Indeed, it is the unique monoidal functor with $J(1) = [\![\text{vertex}]\!]$.)

For any monoidal functor, Hermida and Tennent [Hermida and Tennent 2012] provide monoidal indeterminates by introducing a 'polynomial category', by analogy with a polynomial ring. Unfortunately, a general version for *distributive* monoidal categories is not yet known, so we focus on the specific case of $J : \textbf{FinSetInj}^{\text{op}} \to \textsf{Fam}(\mathbf{G}^{\text{op}})$. We build a new category $\textsf{Fam}(\mathbf{G}^{\text{op}})[v : J\,\textbf{FinSetInj}^{\text{op}}]$, which we abbreviate $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$. It has the same objects as $\textsf{Fam}(\mathbf{G}^{\text{op}})$, but the morphisms $\vec{X} \to \vec{Y}$ are equivalence classes of morphisms

$$[k, f] : [\![\text{vertex}]\!]^k \times \vec{X} \to \vec{Y}$$

in $\textsf{Fam}(\mathbf{G}^{\text{op}})$, modulo reindexing. The reindexing equivalence relation is generated by putting $[k, f] \sim [l, g]$ when there exist injections $\iota_1 \ldots \iota_m : k \to l$ such that

$$g = \left([\![\text{vertex}]\!]^l \times \vec{X} \cong \sum_{j=1}^m [\![\text{vertex}]\!]^l \times X_j \xrightarrow{[\![\text{vertex}]\!]^{(\iota_j)} \times X_j} \sum_{j=1}^m [\![\text{vertex}]\!]^k \times X_j \cong [\![\text{vertex}]\!]^k \times \vec{X} \xrightarrow{f} \vec{Y} \right)$$

where $\vec{X} = (X_1, \ldots, X_m)$. In particular, when $m = 1$, i.e. $\vec{X} = X$ is a singleton sequence, we have

$$\textsf{Fam}(\mathbf{G}^{\text{op}})[v](X, \vec{Y}) \cong \operatorname*{colim}_{k \in \textbf{FinSetInj}} \textsf{Fam}(\mathbf{G}^{\text{op}})([\![\text{vertex}]\!]^k \times X, \vec{Y}). \tag{19}$$

Composition and monoidal structure accumulate in $[\![\text{vertex}]\!]^k$, as usual in the monoidal indeterminates ('para') construction, e.g.

$$\left(\vec{X} \xrightarrow{[k,f]} \vec{Y} \xrightarrow{[l,g]} \vec{Z} \right) = \left(\vec{X} \xrightarrow{[l+k, g \circ ([\![\text{vertex}]\!]^l \times f)]} \vec{Z} \right)$$

and although our equivalence relation is slightly coarser, it still respects the symmetric monoidal category structure, and there is a monoidal functor $\textsf{Fam}(\mathbf{G}^{\text{op}}) \to \textsf{Fam}(\mathbf{G}^{\text{op}})[v]$, regarding each morphism $f : \vec{X} \to \vec{Y}$ in $\textsf{Fam}(\mathbf{G}^{\text{op}})$ as a morphism $[0, f]$ in $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$. But there is also now an adjoined morphism $v = [1, \text{id}] : 1 \to [\![\text{vertex}]\!]$.

This monoidal category $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$ moreover inherits the distributive coproduct structure from $\textsf{Fam}(\mathbf{G}^{\text{op}})$, and the functor $\textsf{Fam}(\mathbf{G}^{\text{op}}) \to \textsf{Fam}(\mathbf{G}^{\text{op}})[v]$ is a distributive Markov functor. To define copairing of $[k, f] : \vec{X} \to \vec{Z}$ and $[l, g] : \vec{Y} \to \vec{Z}$ we use the reindexing equivalence relation to assume $k = l$ and then define the copairing as $\langle [k, f], [k, g] \rangle = [k, \langle f, g \rangle] : \vec{X} + \vec{Y} \to \vec{Z}$.

In summary:

- $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$ is a distributive Markov category.
- $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$ supports the graph interface, via the interpretation of (vertex, edge) in $\textsf{Fam}(\mathbf{G}^{\text{op}})$, but also with the interpretation $[\![\text{new}]\!] = v : 1 \to [\![\text{vertex}]\!]$.

4.2 Bernoulli Bases for Random Graph Models

The following gives a precise characterization of graphons in terms of the numerals of $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]$.

THEOREM 25. *To give a distributive Markov functor* $\textsf{Fam}(\mathbf{G}^{\text{op}})[v]_{\mathbb{N}} \to \textbf{FinStoch}$ *is to give a graphon.*

PROOF OUTLINE. We begin by showing a related characterization: that graphons correspond to certain natural transformations. Observe that any distributive Markov category $\mathbf{C}$ gives rise to a symmetric monoidal functor $\mathbf{C}(1, -) : \textbf{FinSet}_{\mathbb{N}} \to \textbf{Set}$, regarding the numerals of $\textbf{FinSet}_{\mathbb{N}}$ as

objects of C (§2.4). Let $G_k = 2^{k(k-1)/2}$ be the set of k-vertex graphs. We can characterize the natural transformations $\alpha : \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu](1, -) \to \mathbf{FinStoch}(1, -)$ as follows.

$$\mathrm{Nat}(\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu](1, -)\, , \, \mathbf{FinStoch}(1, -))$$

$$\cong \mathrm{Nat}\big(\operatorname*{colim}_{k \in \mathbf{FinSetInj}} \mathbf{FinSet}(G_k, -)\, , \, \mathcal{D}(-)\big) \qquad \text{(Ex. 15(2), Prop. 24(1) and (19))}$$

$$\cong \operatorname*{lim}_{k \in \mathbf{FinSetInj}^{\mathrm{op}}} \mathrm{Nat}(\mathbf{FinSet}(G_k, -)\, , \, \mathcal{D}(-)) \qquad \text{(universal property of colimits)}$$

$$\cong \operatorname*{lim}_{k \in \mathbf{FinSetInj}^{\mathrm{op}}} \mathcal{D}(G_k) \qquad\qquad\qquad \text{(Yoneda lemma)}$$

An element of this limit of sets is by definition a sequence of distributions p_k on G_k that is invariant under reindexing by $\mathbf{FinSetInj}^{\mathrm{op}}$. Since injections are generated by inclusions and permutations, this is then a sequence that is consistent and exchangeable (Def. 19), respectively. Such a natural transformation α is monoidal if and only if the sequence is also *local*. Hence a monoidal natural transformation is the same thing as a random graph model.

In fact, every monoidal natural transformation $\alpha : \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu](1, -) \to \mathbf{FinStoch}(1, -)$ arises uniquely by restricting a distributive Markov functor $F : \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]_{\mathbb{N}} \to \mathbf{FinStoch}$. We now show this, to conclude our proof. Given α, let $F_{m,n} : \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]_{\mathbb{N}}(m, n) \to \mathbf{FinStoch}(m, n)$ be:

$$\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]_{\mathbb{N}}(m, n) \cong \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]_{\mathbb{N}}(1, n)^m \xrightarrow{\alpha_n^m} \mathbf{FinStoch}(1, n)^m \cong \mathbf{FinStoch}(m, n).$$

It is immediate that this F preserves the symmetric monoidal structure and coproduct structure, but not that F is a functor. However, the naturality of α in $\mathbf{FinSet}_{\mathbb{N}}$ gives us that F preserves postcomposition by morphisms of $\mathbf{FinSet}_{\mathbb{N}}$. All of this implies that general categorical composition is preserved as well, since, in any distributive Markov category of the form $\mathbf{C}_{\mathbb{N}}$, for $f : l \to m$ and $g : m \to n$, the composite $g \circ f : l \to n$ is equal to

$$l = l \otimes 1^{\otimes m} \xrightarrow{f \otimes g_1 \otimes \ldots \otimes g_m} m \otimes n^{\otimes m} \xrightarrow{\mathrm{eval}} n$$

where $g_i = g \circ \iota_i$ for $i = 1, \ldots, m$ and eval is just the evaluation map $m \times n^m \to n$ in $\mathbf{FinSet}$. □

COROLLARY 26. *Every graphon arises from a distributive Markov category via the random graph model in (18).*

PROOF SUMMARY. Given a graphon, we consider the distributive Markov functor that corresponds to it, $\Psi : \mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]_{\mathbb{N}} \to \mathbf{FinStoch}$, by Theorem 25. Using the quotient construction of Proposition 18, we get a distributive Markov category with a Bernoulli base. It is straightforward to verify that the random graph model induced by (18) is the original graphon. □

4.3 Remark on Operational Semantics

The interpretation in this section suggests a general purpose operational semantics for closed programs at ground type, $\vdash t : n$, along the following lines:

(1) Calculate the interpretation $[\![t]\!] : 1 \to n$ in $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]$. There are no probabilistic choices in this step, it is a symbolic manipulation, because the morphisms of the Markov category $\mathsf{Fam}(\mathbf{G}^{\mathrm{op}})[\nu]$ are built from tuples of finite graph homomorphisms. In effect, this interpretation pulls all the new's to the front of the term.

(2) Apply the Markov functor $\Psi([\![t]\!])$ to obtain a probability distribution on n, and sample from this distribution to return a result.

5 INTERPRETATION: BLACK-AND-WHITE GRAPHONS VIA MEASURE-THEORETIC PROBABILITY

In Section 4, we gave a general syntactic construction for building an equational theory from a graphon. Since that definition is based on free constructions and quotients, *a priori*, it does not 'explain' what the type vertex stands for. Like contextual equivalence of programs, *a priori*, it does not give useful compositional reasoning methods. To prove two programs are equal, according to the construction of Prop. 18, one needs to quantify over all Z, h, and k, in general.

In this section, we show that one class of graphons, black-and-white graphons (Def. 27), admits a straightforward measure-theoretic semantics, and we can thus use the equational theory induced by this semantics, rather than the method of Section 4. This measure-theoretic semantics is close to previous measure-theoretic work on probabilistic programming languages (e.g. [Kozen 1981; Staton 2017]).

After recapping measure-theoretic probability (§2.3.3), in Section 5.1, we show that every black-and-white graphon arises from a measure-theoretic interpretation (Prop. 28). In Section 5.2, by defining 'measure-theoretic interpretation' more generally, we show that, conversely, this measure-theoretic approach can *only* cater for black-and-white graphons (Prop. 29).

5.1 Black-and-White Graphons from Equational Theories

DEFINITION 27. *[e.g. [Janson 2013]] A graphon* $W : [0,1]^2 \to [0,1]$ *is* black-and-white *if there exists* $E : [0,1]^2 \to \{0,1\}$ *such that* $W(x,y) = E(x,y)$ *for almost all* x, y.

Recall that the Giry monad (Def. 11) gives rise to a Bernoulli-based distributive Markov category (§2.3.3, Ex. 15). For any black-and-white graphon W, we define an interpretation of the graph interface for the probabilistic programming language using $\mathcal{G}$, as follows.

- $[\![\text{vertex}]\!]_W = [0,1]$; $[\![\text{bool}]\!]_W = 2$, the discrete two element space;
- $[\![\text{new}()]\!]_W = \text{Uniform}(0,1)$, the uniform distribution on $[0,1]$;
- $[\![\text{edge}]\!]_W(x,y) = \eta(E(x,y))$.

PROPOSITION 28. *Let* W *be a black-and-white graphon. The equational theory induced by* $[\![-]\!]_W$ *induces the graphon* W *according to the construction in Section 3.2.*

PROOF. Suppose that W corresponds to the sequence of random graphs $p_1, p_2, \ldots$ as in Section 3.1. Consider the term t_n in (18), and directly calculate its interpretation. Then, we get $[\![t_n]\!]_W = p_n$, via (16), as required.

The choice of E does not matter in the interpretation of these terms, because $W = E$ almost everywhere.   □

5.2 All Measure-Theoretic Interpretations are Black-and-White

Although the model in Section 5.1 is fairly canonical, there are sometimes other enlightening interpretations using the Giry monad. These also correspond to black-and-white graphons.

For example, consider the geometric-graph example from Figure 1. We interpret this using the Giry monad, putting

- $[\![\text{vertex}]\!] = S_2$, the sphere; $[\![\text{bool}]\!] = 2$;
- $[\![\text{new}()]\!] = \text{Uniform}(S_2)$, the uniform distribution on the sphere;
- $[\![\text{edge}]\!](x,y) = \eta(d(x,y) < \theta)$, i.e. an edge if their distance is less than θ.

This will again induce a graphon, via (18). We briefly look at theories that arise in this more flexible way:

PROPOSITION 29. *Consider any interpretation of the graph interface in the Giry monad: a measurable space $[\![vertex]\!]$, a measurable set $[\![edge]\!] \subseteq [\![vertex]\!]^2$, and a probability measure $[\![new()]\!]$ on $[\![vertex]\!]$. The induced graphon is black-and-white.*

PROOF NOTES. If $[\![vertex]\!]$ is standard Borel, the randomization lemma [Kallenberg 2010, Lem. 3.22] gives a function $f : [0,1] \rightarrow [\![vertex]\!]$ that pushes the uniform distribution on $[0,1]$ onto the probability measure $[\![new()]\!]$. We define a black-and-white graphon W by $W(x,y) = 1$ if $(f(x), f(y)) \in [\![edge]\!]$, and $W(x,y) = 0$ otherwise. This graphon interpretation $[\![-]\!]_W$ gives the same sequence of graphs in (18), just by reparameterizing the integrals.

If $[\![vertex]\!]$ is not standard Borel, we note that there is an equivalent interpretation where it is, because there exists a measure-preserving map $[\![vertex]\!] \rightarrow \Omega$ to a standard Borel space Ω and a measurable set $E \subseteq \Omega^2$ that pulls back to $[\![edge]\!]$, giving rise to the same graphon (e.g. [Janson 2013, Lemma 7.3]). □

Discussion. Proposition 29 demonstrates that this measure-theoretic interpretation has limitations.

DEFINITION 30. *For $\alpha \in (0,1)$, the* Erdős–Rényi graphon $W_\alpha : [0,1]^2 \rightarrow [0,1]$ *is given by* $W_\alpha(x,y) = \alpha$.

The Erdős-Rényi graphons cannot arise from measure-theoretic interpretations of the graph interface, because they are not black-and-white. In Section 6, we give an alternative interpretation for the Erdős–Rényi graphons.

The reader might be tempted to interpret an Erdős–Rényi graphon by defining

$$[\![edge]\!]_{W_\alpha}(x,y) \quad = \quad \text{bernoulli}(\alpha).$$

However, this interpretation does not provide a model for the basic equations of the language, because this $[\![edge]\!]$ is not deterministic, and derivable equations such as (6) will fail. Intuitively, once an edge has been sampled between two given nodes, its presence (or absence) remains unchanged in the rest of the program, *i.e.* the edge is not resampled again, it is memoized (see also [Kaddar and Staton 2023; Roy et al. 2008]).

Although not all graphons are black-and-white, these are still a widely studied and useful class. They are often called 'random-free'. For example, an alternative characterization is that the random graph model of Prop. 21 has subquadratic entropy function [Janson 2013, §10.6].

6 INTERPRETATION: ERDŐS–RÉNYI GRAPHONS VIA RADO-NOMINAL SETS

In Section 4, we gave a general construction to show that every graphon arises from a Bernoulli-based equational theory. In Section 5, we gave a more concrete interpretation, based on measure-theory, for black-and-white graphons. We now consider the Erdős–Rényi graphons (Def. 30), which are not black-and-white.

Our interpretation is based on Rado-nominal sets. These are also studied elsewhere, but for different purposes (e.g. [Bojańczyk et al. 2014; Bojańczyk and Place 2012; Klin et al. 2016], [Pitts 2013, §1.9]).

Rado-nominal sets (§6.1) are sets that are equipped with an action of the automorphisms of the Rado graph, which is an infinite graph that contains every finite graph. There is a particular Rado-nominal set $\mathbb{V}$ of the vertices of the Rado graph. The type vertex will be interpreted as $\mathbb{V}$; edge is interpreted using the edge relation E on $\mathbb{V}$. The equational theory induced by this interpretation gives rise to the Erdős–Rényi graphons (Def. 30).

Since Rado-nominal sets form a model of ZFA set theory (Prop. 36), we revisit probability theory internal to this setting. We consider internal probability measures on Rado-nominal sets (§6.3), and we show that there are internal probability measures on $\mathbb{V}$ that give rise to Erdős–Rényi graphons

(§6.3). The key starting point here is that, internal to Rado-nominal sets, the only functions $\mathbb{V} \to 2$ are the sets of vertices that are definable in the language of graphs (§6.2).

We organize the probability measures (Def. 37) into a probability monad on Rado-nominal sets (§6.4), analogous to the Giry monad. Fubini does not routinely hold in this setting (§6.4.4), but we use a standard technique to cut down to a commutative affine monad (§6.4.5). This gives rise to a Bernoulli-based equational theory, and in fact, this theory corresponds to Erdős–Rényi graphons (via (18): Corollary 45).

6.1 Definition and First Examples

The Rado graph $(\mathbb{V}, E)$ ([Ackermann 1937; Rado 1964], also known as the 'random graph' [Erdős and Rényi 1959]) is the unique graph, up to isomorphism, with a countably infinite set of vertices that has the extension property: if A, B are disjoint finite subsets of $\mathbb{V}$, then there is a vertex $a \in \mathbb{V} \setminus (A \cup B)$ with an edge to all the vertices in A but none of the vertices in B.

The Rado graph embeds every finite graph, which can be shown by using the extension property inductively.

An automorphism of the Rado graph is a graph isomorphism $\mathbb{V} \to \mathbb{V}$. The automorphisms of the Rado graph relate to isomorphisms between finite graphs, as follows. First, if A is a finite graph regarded as a subset of $\mathbb{V}$, then any automorphism σ induces an isomorphism of finite graphs $A \cong \sigma[A]$. Conversely, if $f \colon A \cong B$ is an isomorphism of finite graphs, and we regard A and B as disjoint subsets of $\mathbb{V}$, then there exists an automorphism σ of $\mathbb{V}$ that restricts to f (i.e. $f = \sigma|_A$).

We write Aut(Rado) for the group of automorphisms of $(\mathbb{V}, E)$. (This has been extensively studied in model theory and descriptive set theory, e.g. [Angel et al. 2014; Kechris et al. 2005].)

DEFINITION 31. *A* Rado-nominal set *is a set X equipped with an action $\bullet \colon \text{Aut(Rado)} \times X \to X$ (i.e. id $\bullet x = x$; $(\sigma_2 \cdot \sigma_1) \bullet x = \sigma_2 \bullet \sigma_1 \bullet x$) such that every element has finite support.*

An element $x \in X$ is defined to have finite support *if there is a finite set $A \subseteq \mathbb{V}$ such that for all automorphisms σ, if σ fixes A (i.e. $\sigma|_A = \text{id}_A$), it also fixes x (i.e. $\sigma \bullet x = x$).*

Equivariant functions between Rado-nominal sets *are functions that preserve the group action (i.e. $f(\sigma \bullet x) = \sigma \bullet (f(x))$).*

PROPOSITION 32 ([PITTS 2013]). *If finite sets $A, B \subseteq \mathbb{V}$ both support x, so does $A \cap B$. Hence every element has a least support.*

EXAMPLE 33. (1) *The set $\mathbb{V}$ of vertices is a Rado-nominal set, with $\sigma \bullet a = \sigma(a)$. The support of vertex a is $\{a\}$.*

(2) *The set $\mathbb{V} \times \mathbb{V}$ of pairs of vertices is a Rado-nominal set, with $\sigma \bullet (a, b) = (\sigma(a), \sigma(b))$. The support of (a, b) is $\{a, b\}$. More generally, a finite product of Rado-nominal sets has a coordinate-wise group action.*

(3) *The edge relation $E \subseteq \mathbb{V} \times \mathbb{V}$ is a Rado-nominal subset (which is formally defined in §6.2) because automorphisms preserve the edge relation.*

(4) *Any set X can be regarded with the discrete action, $\sigma \bullet x = x$, and then every element has empty support. We regard these sets with the discrete action: $1 = \{\star\}$; $2 = \{0, 1\}$; $\mathbb{N}$; and the unit interval $[0, 1]$.*

6.2 Powersets and Definable Sets

For any subset $S \subseteq X$ of a Rado-nominal set, we can define $\sigma \bullet S = \sigma[S] = \{\sigma \bullet x \mid x \in S\}$. We let

$$2^X = \{S \subseteq X \mid S \text{ has finite support}\}. \tag{20}$$

This is a Rado-nominal set.

EXAMPLE 34. *We give some concrete examples of subsets.*

(1) *For vertices b and c in $\mathbb{V}$ with no edge between them, the set $\{a \in \mathbb{V} \mid E(a, b) \wedge E(a, c)\}$ is the set of ways of forming a horn. It has support $\{b, c\}$.*

(2) $\{(b, c) \in \mathbb{V}^2 \mid E(a, b) \wedge E(a, c) \wedge \neg E(b, c)\}$ *is the set of horns with apex a; it has support $\{a\}$.*

(3) $\{(a, b, c) \in \mathbb{V}^3 \mid E(a, b) \wedge E(a, c) \wedge \neg E(b, c)\}$ *is the set of all oriented horns; it has empty support.*

(4) *(Non-example) There is a countable totally disconnected subgraph of $\mathbb{V}$; it does not have finite support as a subset of $\mathbb{V}$.*

In fact, the finitely supported subsets correspond exactly to the definable sets in first-order logic over the theory of graphs. The following results may be folklore.

PROPOSITION 35. *Let $S \subseteq \mathbb{V}^n$, and $A \subseteq \mathbb{V}$ be finite. The following are equivalent:*

- $S = \{(s_1, \ldots s_n) \mid \phi(s_1 \ldots s_n)\}$, *for a first-order formula ϕ over the theory of graphs, with parameters in A;*
- *S has support A.*

PROOF. ($\Rightarrow$) For all isomorphisms $f \colon \mathbb{V} \to \mathbb{V}$ that fix A, and for all elements $a_1 \ldots a_k \in A$ and subsets $S = \{(s_1, \ldots, s_n) \mid \phi(s_1 \ldots s_n, a_1 \ldots a_k)\}$, we have

$$\phi(f(s_1) \ldots f(s_n), a_1 \ldots a_k) \quad = \quad \phi(f(s_1) \ldots f(s_n), f(a_1) \ldots f(a_k)).$$

Furthermore, ϕ is invariant with respect to f. Thus, the image $f(S) \subseteq S$. By a similar argument, we have $f^{-1}(S) \subseteq S$, so that $S \subseteq f(S)$. Thus, $f(S) = S$ ([Marker 2002, Prop. 1.3.5]).

($\Leftarrow$) This is a consequence of the Ryll-Nardzewski theorem for the theory of the Rado graph (which can be shown to be ω-categorical by a back-and-forth argument, using the extension property of the Rado graph). But we give here a more direct proof, assuming $n = 1$ for simplicity. Suppose $A \subseteq \mathbb{V}$ is a finite support for S. Then, for any $v, v' \in \mathbb{V} \backslash A$, if v and v' have the same connectivity to A, then they are either both in or not in S since, by the extension property, we can find an automorphism fixing A and sending v to v'. The set of vertices with the same connectivity to A as v is definable, and there are only $2^{|A|}$ such sets. Hence, $S \backslash A$ is a union of finitely many definable sets, and as $S \cap A$ is definable (being finite), so is $S = (S \backslash A) \cup (S \cap A)$.   □

We note that 2^X in (20) is a canonical notion of internal powerset, from a categorical perspective.

PROPOSITION 36. **RadoNom** *is a Boolean Grothendieck topos, with powerobject 2^X in (20).*

PROOF NOTES. **RadoNom** can be regarded as continuous actions of Aut(Rado), regarded as a topological group with the product topology, and then we invoke standard methods [Johnstone 2002, Ex. A2.1.6]. It is also equivalent to the category of sheaves over finite graphs and embeddings with the atomic topology. See [Caramello 2013, 2014] for general discussion.   □

6.3 Probability Measures on Rado-Nominal Sets

The finitely supported sets $S \subseteq \mathbb{V}$ can be regarded as 'events' to which we would assign a probability. For example, if we already have vertices b and c, we may want to know the chance of picking a vertex that forms a horn, and this would be the probability of the set in Ex. 34(a).

DEFINITION 37. *A sequence $S_1, S_2 \cdots \subseteq X$ is said to be* support-bounded *if there is one finite set $A \subseteq \mathbb{V}$ that supports all the sets S_i.*

A function $\mu \colon 2^X \to [0, 1]$ is (internally) countably additive *if for any support-bounded sequence $S_1, S_2 \cdots \subseteq X$ of disjoint sets,*

$$\mu(\uplus_{i=1}^{\infty} S_i) = \sum_{i=1}^{\infty} \mu(S_i).$$

A probability measure *on a Rado-nominal set X is an equivariant function $\mu \colon 2^X \to [0, 1]$ that is internally countably additive, such that $\mu(X) = 1$.*

We remark that there are two subtleties here. First, we restrict to support-bounded sequences. These are the correctly internalized notion of sequence in Rado-nominal sets, since they correspond precisely to finitely-supported functions $\mathbb{N} \to 2^X$. Second, we consider a Rado-nominal set to be equipped with its internal powerset 2^X, rather than considering sub-σ-algebras.

Measures on the space of vertices. We define an internal probability measure (Def. 37) on the space $\mathbb{V}$ of vertices, which, we will show, corresponds to the Erdős-Rényi graphon. Fix $\alpha \in [0, 1]$, the chance of an edge.

We define the measure ν_α of a definable set $S \in 2^{\mathbb{V}}$ as follows. Suppose that S has support $\{a_1, \ldots, a_n\}$. We choose an enumeration of vertices $(v_1, \ldots, v_{2^n})$ in $\mathbb{V}$ (disjoint from $\{a_1, \ldots, a_n\}$) that covers all the 2^n possible edge relationships that a vertex could have with the a_i's. (For example, v_1 has no edges to any a_i, and v_{2^n} has an edge to every a_i, and the other v_j's have the other possible edge relationships.) Let:

$$\nu_\alpha(S) = \sum_{j=1}^{2^n} [v_j \in S] \prod_{i=1}^{n} \big(\alpha E(v_j, a_i) + (1 - \alpha)(1 - E(v_j, a_i)) \big). \tag{21}$$

PROPOSITION 38. *The assignment given in* (21) *is an internal probability measure (Def. 37) on* $\mathbb{V}$.

PROOF. The function ν_α is well-defined: it does not depend on the choice of v_j's (by Prop. 35), nor on the choice of support (by direct calculation). It is equivariant, since for $\sigma \bullet S$, a valid enumeration of vertices is given by $\sigma \bullet v_1, \ldots \sigma \bullet v_{2^n}$. Also, $\nu(\mathbb{V}) = 1$, since $\mathbb{V}$ has empty support. Internal countable additivity follows from the identity $\big[v_j \in \biguplus_{i=1}^{\infty} S_i \big] = \sum_{i=1}^{\infty} [v_j \in S_i]$. □

Remark. The definitions and results of this section appear to be novel. However, the general idea of considering measures on formulas which are invariant to substitutions that permute the variables goes back to work of Gaifman [Gaifman 1964]. The paper [Ackerman et al. 2016a] characterizes those countably infinite graphs that can arise with probability 1 in that framework; see [Ackerman et al. 2017b] for a discussion of how Gaifman's work connects to Prop. 21.

6.4 Nominal Probability Monads

Since **RadoNom** is a Boolean topos with natural numbers object (Prop. 36), we can interpret measure-theoretic notions in the internal language of the topos, as long as they do not require the axiom of choice. We now spell out the resulting development, without assuming familiarity with topos theory. By doing this, we build new probability monads on **RadoNom**.

6.4.1 Finitely Supported Functions and Measures. Let X and Y be Rado-nominal sets. The set of all functions $X \to Y$ has an action of Aut(Rado), given by $(\sigma \bullet f)(x) = \sigma^{-1} \bullet (f(\sigma \bullet x))$. The function space $[X \Rightarrow Y]$ comprises those functions that have finite support under this action. Categorically, this structure is uniquely determined by the 'currying' bijection, natural in Z:

$$\mathbf{RadoNom}(Z \times X, Y) \cong \mathbf{RadoNom}(Z, X \Rightarrow Y).$$

(For example, the powerobject 2^X (§6.2) can be regarded as $[X \Rightarrow 2]$, if we regard a set as its characteristic function.)

In Def. 37, we focused on equivariant probability measures. We generalize this to finitely supported measures. For example, pick a vertex $a \in \mathbb{V}$. Then, the Dirac measure on $\mathbb{V}$ (i.e. $\delta_a(S) = 1$ if $a \in S$, and $\delta_a(S) = 0$ if $a \notin S$) has support $\{a\}$.

DEFINITION 39. *For a Rado-nominal set* X, *let* $\mathcal{P}(X)$ *comprise the finitely supported functions* $\mu : 2^X \to [0, 1]$ *that are internally countably additive, and satisfy* $\mu(X) = 1$. *This is a Rado-nominal set, as a subset of* $[2^X \Rightarrow [0, 1]]$. *Functions in* $\mathcal{P}(X)$ *are called* finitely supported probability measures.

6.4.2 Internal Integration. We revisit some basic integration theory in this nominal setting. In traditional measure theory, one can define the Lebesgue integral of a *measurable* function $f : X \to [0, 1]$ by $\int f(x)\mu(dx) = \sup \sum_{i=1}^{n} r_i \mu(U_i)$ where the supremum ranges over simple functions $\sum_i r_i[- \in U_i]$ with U_i measurable in X and bounded above by f (§2.3.3). The same construction works in the internal logic of **RadoNom**.

Note that the following does not mention f being measurable: since X is considered to have its internal powerset σ-algebra, finite-supportedness implies 'measurability' here.

PROPOSITION 40. *Let $\mu \in \mathcal{P}(X)$ be a finitely supported probability measure on X. For any finitely supported function $f : X \to [0, 1]$, the internally-constructed Lebesgue integral $\int f(x)\,\mu(dx) \in [0, 1]$ exists. Moreover, integration is an equivariant map*

$$\int : \mathcal{P}(X) \times [X \Rightarrow [0, 1]] \to [0, 1]$$

which preserves suprema of internally countable monotone sequences in its second argument.

PROOF. If $U_1, \ldots, U_n \subseteq X$ are finitely supported, $r_1, \ldots, r_n \in [0, 1]$, and $\sum_i r_i[- \in U_i] \leq f$, then by ordinary additivity of μ, we have $\sum r_i \mu(U_i) \in [0, 1]$. By ordinary real analysis, the supremum of all such values exists and is in $[0, 1]$. For equivariance, recall that $[0, 1]$ is equipped with the trivial action of Aut(Rado). Use the fact that $\sum_i r_i[- \in U_i] \leq f$ if and only if $\sum_i r_i[- \in \sigma \bullet U_i] \leq \sigma \bullet f$. The last claim is the monotone convergence theorem internalized to **RadoNom**.   □

6.4.3 Kernels and a Monad. We can regard a 'probability kernel' as a finitely supported function $k : X \to \mathcal{P}(Y)$. Equivalently, k is a finitely supported function $k : X \times 2^Y \to [0, 1]$ that is countably additive and has mass 1 in its second argument.

(In traditional measure theory, one would explicitly ask that k is measurable in its first argument, but as we observed, finite-supportedness already implies it.)

As usual, probability kernels compose, and this allows us to regard them as Kleisli morphisms for a monad (Def. 6), defined as follows.

DEFINITION 41. *We define the strong monad $\mathcal{P}$ on **RadoNom** as follows.*

- *For a Rado-nominal set X, let $\mathcal{P}(X)$ comprise the finitely supported probability measures (Def. 39).*
- *The unit of the monad $\eta_X : X \to \mathcal{P}(X)$ is the Dirac measure, $\eta_X(x)(S) = [x \in S]$.*
- *The bind $(\ggeq) : \mathcal{P}(X) \times (X \Rightarrow \mathcal{P}(Y)) \to \mathcal{P}(Y)$ is given by*

$$(\mu \ggeq k)(S) = \int_X k(x, S)\,\mu(dx).$$

We note that this is similar to the 'expectations monad' [Jacobs and Mandemaker 2012, Thm. 4].

6.4.4 Commuting Integrals (Fubini). For measures $\mu_1 \in \mathcal{P}(X)$ and $\mu_2 \in \mathcal{P}(Y)$, the monad structure allows us to define a product measure

$$\mu_1 \otimes \mu_2 = \big(\mu_1 \ggeq (\lambda x.\, \mu_2 \ggeq \lambda y.\, \eta(x, y))\big)$$

$$\int f(x, y)\,(\mu_1 \otimes \mu_2)(d(x, y)) = \int \int f(x, y)\,\mu_2(dy)\,\mu_1(dx). \tag{22}$$

Although this iterated integration is reminiscent of the traditional approach, in general we cannot reorder integrals ('Fubini does not hold'). For example, given two measures ν_α and ν_β for $\alpha \neq \beta$

and f being the characteristic function of the set $\{(x, y) : E(x, y)\} \subseteq \mathbb{V}^2$, we have

$$\int \int [E(x, y)] \, \nu_\alpha(\mathrm{d}y) \, \nu_\beta(\mathrm{d}x) = \int \alpha \, \nu_\beta(\mathrm{d}x) = \alpha$$
$$\neq \beta = \int \beta \, \nu_\alpha(\mathrm{d}y) = \int \int [E(x, y)] \, \nu_\beta(\mathrm{d}x) \, \nu_\alpha(\mathrm{d}y). \tag{23}$$

However, it does hold when we consider only copies of the same measure.

PROPOSITION 42. *For $\nu_\alpha \in \mathcal{P}(\mathbb{V})$ as in (21), ν_α commutes with ν_α. That is, for any finitely supported $f : \mathbb{V} \times \mathbb{V} \to [0, 1]$,*

$$\int \int f(x, y) \, \nu_\alpha(\mathrm{d}y) \, \nu_\alpha(\mathrm{d}x) = \int \int f(x, y) \, \nu_\alpha(\mathrm{d}x) \, \nu_\alpha(\mathrm{d}y).$$

PROOF NOTES. By Prop. 35 and 40, it suffices to check on the indicator functions of definable subsets of $\mathbb{V}^2$. The indicators of sets $\{(x, y) \mid \Phi(x, y)\}$ where $\Phi(x, y)$ is a disjunction of $x = y$, $x = a$, or $y = a$ for some $a \in \mathbb{V}$ are seen to have integral 0 on both sides. The remaining possibilities can be reduced to the case where $\Phi_{A,\phi,\psi,\epsilon}(x, y)$ is $(x, y \notin A) \wedge (x \neq y) \wedge (E(x, y) \leftrightarrow \epsilon) \wedge \bigwedge_{a \in A}(E(a, x) \leftrightarrow \phi_a) \wedge (E(a, y) \leftrightarrow \psi_a)$ where $A \subseteq \mathbb{V}$ is a finite set, $\epsilon \in \{\bot, \top\}$, and $\phi, \psi \in \{\bot, \top\}^A$. This formula corresponds to choosing a two-vertex extension of the finite graph spanned by $A \subseteq \mathbb{V}$. Intuitively, the two double integrals correspond to the two alternative two-step computations of the conditional probability of extending the graph A to this extension according to which of the two vertices is sampled first, and indeed both evaluate to $\alpha^k(1 - \alpha)^{2|A|+1-k}$ where $k = [\epsilon] + \sum_{a \in A}([\phi_a] + [\psi_a])$. □

Remark. In traditional measure theory, iterated integrals are defined using product σ-algebras. Here we have not constructed product σ-algebras, but rather always take the internal powerset as the σ-algebra. This allows us to view all the definable sets as measurable on $\mathbb{V}^n$ (Prop. 35), which is very useful. We remark that alternative product spaces also arise in non-standard approaches to graphons (see [Tao 2013, §6] for an overview), and also in quasi-Borel spaces [Heunen et al. 2017] for different reasons.

6.4.5 A Commutative Monad. We now use Prop. 42 to build a commutative affine submonad $\mathcal{P}_\alpha$ of the monad $\mathcal{P}$, which we will use to model the graph interface for the probabilistic programming language. With Prop. 36, we use the following general result.

PROPOSITION 43. *Let $\mathcal{T}$ be a strong monad on a Grothendieck topos. Consider a family of morphisms $\{f_i : X_i \to \mathcal{T}(Y_i)\}_{i \in I}$.*

- *There is a least strong submonad $\mathcal{T}_f \subseteq \mathcal{T}$ through which all f_i factor.*
- *If the morphisms f_i all commute with each other, then $\mathcal{T}_f$ is a commutative monad (Def. 7).*

PROOF NOTES. Our argument is close to [Kammar and McDermott 2018, §2.3] and also [Kammar 2014, Thms. 7.5 & 12.8].

We let $\mathcal{T}_f$ be the least subfunctor of $\mathcal{T}$ that contains the images of the f_i's and η, and is closed under the image of monadic bind ($\gg=$). To show that this exists, we proceed as follows. First, fix a regular cardinal $\lambda > I$ such that Y_i's are all λ-presentable, such that the topos is locally λ-presentable (e.g. [Adámek and Rosický 1994]). Consider the poset $\mathrm{Sub}_\lambda(\mathcal{T})$ of λ-accessible subfunctors of $\mathcal{T}$. The cardinality bound λ ensures it is small. Ordered by pointwise inclusion, this is a complete lattice: the non-empty meets are immediate, and the empty meet requires us to consider the λ-accessible coreflection of $\mathcal{T}$.

We defined $\mathcal{T}_f$ by a monotone property which we can regard as a monotone operator on this complete lattice $\mathrm{Sub}_\lambda(\mathcal{T})$, and so the least λ-accessible subfunctor exists. This is $\mathcal{T}_f$. Concretely, it

is a least upper bound of an ordinal indexed chain. The chain starts with the functor

$$F_0(Z) = \bigcup_{i \in I, g: Y_i \to Z} \text{image}(\mathcal{T}(g) \circ f_i) \quad \subseteq \mathcal{T}(Z)$$

which is λ-accessible because the Y_i's are λ-presentable. The chain iteratively closes under the image of monadic bind, until we reach a subfunctor that is a sub*monad* of $\mathcal{T}$.

To see that $\mathcal{T}_f$ is commutative, we appeal to (transfinite) induction. Say that a subfunctor F of $\mathcal{T}$ is *commutative* if all morphisms that factor through F commute (Def. 7), and then note that the property of being commutative is preserved along the ordinal indexed chain. □

With this in mind, fixing a measure ν_α as in (21), we form the least submonad $\mathcal{P}_\alpha$ of $\mathcal{P}$ induced by the morphisms

$$\nu_\alpha : 1 \to \mathcal{P}(\mathbb{V}) \qquad \text{bernoulli} : [0, 1] \to \mathcal{P}(2) \tag{24}$$

where $\text{bernoulli}(r) = r \cdot \eta(0) + (1 - r) \cdot \eta(1)$.

COROLLARY 44. *The least submonad $\mathcal{P}_\alpha$ of the probability monad $\mathcal{P}$ induced by the morphisms in (24) is a commutative affine monad (Def. 7).*

PROOF NOTES. It is easy to show that bernoulli commutes with every morphism $X \to \mathcal{P}(Y)$. Moreover, ν_α commutes with itself (Prop. 42). Finally, $\mathcal{P}_\alpha$ is affine since $\mathcal{P}$ is. □

6.5 Summary and Interpretation

Fix $\alpha \in [0, 1]$. We induce an internal measure ν_α on the vertices of the Rado graph as explained in (21); and build a commutative submonad $\mathcal{P}_\alpha$ of $\mathcal{P}$. We can then interpret the graph probabilistic programming language. We interpret types as Rado-nominal sets:

$$[\![\text{bool}]\!] = 2 \qquad [\![\text{vertex}]\!] = \mathbb{V} \qquad [\![\text{unit}]\!] = 1 \qquad [\![A_1 * A_2]\!] = [\![A_1]\!] \times [\![A_2]\!]. \tag{25}$$

We interpret typed programs $\Gamma \vdash t : A$ as Kleisli morphisms

$$[\![\Gamma]\!] \to \mathcal{P}_\alpha([\![A]\!])$$

i.e. internal probability kernels $[\![\Gamma]\!] \times 2^{[\![A]\!]} \to [0, 1]$. Sequencing (let) is interpreted using the monad structure, with $[\![\text{new}]\!] : 1 \to \mathcal{P}_\alpha(\mathbb{V})$ and $[\![\text{edge}]\!] : \mathbb{V} \times \mathbb{V} \to \mathcal{P}_\alpha(2)$ as

$$[\![\text{new}()]\!] = \nu_\alpha \qquad [\![\text{edge}]\!](v, w) = \eta(E(v, w)) \tag{26}$$

COROLLARY 45. *Consider the interpretation in Rado-nominal sets ((25)–(26)). If we form the sequence of random graphs in (18), then these correspond to the Erdős-Rényi graphon.*

PROOF NOTES. The semantics interprets ground types as finite sets with discrete Aut(Rado) action – in which case internal probability kernels correspond to stochastic matrices, agreeing with **FinStoch**. Thus, the theory is Bernoulli-based. To see that the graphon arises, consider for instance when $n = 2$, we have:

$$[\![t_2]\!](\star) = \int \begin{pmatrix} [E(x_1, x_1)], & [E(x_1, x_2)] \\ [E(x_2, x_1)], & [E(x_2, x_2)] \end{pmatrix} (\nu_\alpha \otimes \nu_\alpha)(\text{d}(x_1, x_2))$$

for t_2 as in (18), and therefore

$$[\![t_2]\!] = \begin{pmatrix} \delta_0, & \text{bernoulli}(\alpha) \\ \text{bernoulli}(\alpha), & \delta_0 \end{pmatrix} : \mathcal{P}(2^4)$$

For general n, this corresponds to the random graph model $p_{W_\alpha, n}$ for the Erdős-Rényi graphon W_α. □

7 CONCLUSION

Summary. We have shown that equational theories for the graph interface to the probabilistic programming language (Ex. 1) give rise to graphons (Theorem 23). Conversely, every graphon arises in this way. We showed this generally using an abstract construction based on Markov categories (Corollary 26) and methods from category theory [Hermida and Tennent 2012; Hu and Tholen 1995]. Since this is an abstract method, we also considered two concrete styles of semantic interpretation that give rise to classes of graphons: traditional measure-theoretic interpretations give rise to black-and-white graphons (Prop. 28), and an interpretation using the internal probability theory of Rado-nominal sets gives rise to Erdős–Rényi graphons (Corollary 45).

Further context, and future work. The idea of studying exchangeable structures through program equations is perhaps first discussed in the abstract [Staton et al. 2017], whose §3.2 ends with an open question about semantics of languages with graphs that the present paper addresses. Subsequent work addressed the simpler setting of exchangeable sequences and beta-bernoulli conjugacy through program equations [Staton et al. 2018], and stochastic memoization [Kaddar and Staton 2023]; the latter uses a category similar to **RadoNom**, although the monad is different. Beyond sequences [Staton et al. 2018] and graphs (this paper), a natural question is how to generalize to arbitrary exchangeable interfaces (see e.g. [Orbanz and Roy 2015]). For example, we could consider exchangeable random boolean arrays via the interface

$$\text{new-row} : \text{unit} \to \text{row}, \quad \text{new-column}() : \text{unit} \to \text{column}, \quad \text{entry} : \text{row} * \text{column} \to \text{bool}$$

and random hypergraphs with the interface

$$\text{new} : \text{unit} \to \text{vertex}, \quad \text{hyperedge}_n : \text{vertex}^n \to \text{bool}.$$

We could also consider interfaces for hierarchical structures, such as arrays where every entry contains a graph. Diverse exchangeable random structures have been considered from the model-theoretic viewpoint [Ackerman 2015; Crane and Towsner 2018] and from the perspective of probability theory (e.g. [Campbell et al. 2023; Jung et al. 2021; Kallenberg 2010]), but it remains to be seen whether the programming perspective here can provide a unifying view. Another point is that graphons correspond to dense graphs, and so a question is how to accommodate sparse graphs from a programming perspective (e.g. [Caron and Fox 2017; Veitch and Roy 2019]).

This paper has focused on a very simple programming language (§2.1). As mentioned in Section 1.5, several implementations of probabilistic programming languages do support various Bayesian nonparametric primitives based on exchangeable sequences, partitions, and relations (e.g. [Dash et al. 2023; Goodman et al. 2008; Kiselyov and Shan 2010; Mansinghka et al. 2014; Roy et al. 2008; Wood et al. 2014]). In particular, the 'exchangeable random primitive' (XRP) interface [Ackerman et al. 2016b; Wu 2013] provides a built-in abstract data type for representing exchangeable sequences. This aids model design by its abstraction, but also aids inference performance by clarifying the independence relationships.

Aside from practical inference performance, we can ask whether representation and inference are computable. For the simpler setting of exchangeable sequences, this is dealt with positively by [Freer and Roy 2010, 2012]. The question of computability for graphons and exchangeable graphs is considerably subtler, and some standard representations are noncomputable [Ackerman et al. 2019] (see also [Ackerman et al. 2017a]). This suggests several natural questions about whether certain natural classes of computable exchangeable graphs can be identified by program analyses in the present context.

ACKNOWLEDGMENTS

REFERENCES

Nathanael Ackerman. 2015. Representations of Aut(M)-Invariant Measures. *arXiv e-print 1509.06170* (2015).

Nathanael Ackerman, Cameron Freer, Alex Kruckman, and Rehana Patel. 2017b. Properly ergodic structures. *arXiv e-print 1710.09336* (2017).

Nathanael Ackerman, Cameron Freer, and Rehana Patel. 2016a. Invariant measures concentrated on countable structures. *Forum Math. Sigma* 4 (2016), e17.

Nathanael L. Ackerman, Jeremy Avigad, Cameron E. Freer, Daniel M. Roy, and Jason M. Rute. 2017a. On computable representations of exchangeable data. Workshop on Probabilistic Programming Semantics (PPS 2017). https://pps2017.luddy.indiana.edu/files/2016/12/compAH.pdf

Nathanael L. Ackerman, Jeremy Avigad, Cameron E. Freer, Daniel M. Roy, and Jason M. Rute. 2019. Algorithmic barriers to representing conditional independence. In *Proc. 34th ACM/IEEE Symp. Logic in Comp. Sci. (LICS 2019)*. 1–13.

Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. 2016b. Exchangeable Random Primitives. Workshop on Probabilistic Programming Semantics (PPS 2016). http://pps2016.luddy.indiana.edu/files/2015/12/xrp.pdf

Wilhelm Ackermann. 1937. Die Widerspruchsfreiheit der allgemeinen Mengenlehre. *Math. Ann.* 114, 1 (1937), 305–315.

J Adámek and J Rosický. 1994. *Locally presentable and accessible categories*. Cambridge University Press.

Pablo Andrés-Martínez, Chris Heunen, and Robin Kaarsgaard. 2022. Universal Properties of Partial Quantum Maps. In *Proc QPL 2022*.

Omer Angel, Alexander S. Kechris, and Russell Lyons. 2014. Random orderings and unique ergodicity of automorphism groups. *J. Eur. Math. Soc. (JEMS)* 16, 10 (2014), 2059–2095.

Pedro Henrique Azevedo de Amorim. 2023. A Higher-Order Language for Markov Kernels and Linear Operators. In *Proc. FOSSACS 2023*.

Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. 1992. Linear λ-calculus and categorical models revisited. In *Proc. CSL 1992*.

Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. 2014. Automata theory in nominal sets. *Logical Methods in Computer Science* 10, 3 (2014).

M. Bojańczyk and T. Place. 2012. Towards model theory with data values. In *Proc. ICALP 2012*.

Stefan Bornholdt and Hans Georg Schuster (Eds.). 2002. *Handbook of Graphs and Networks*. Wiley.

Sébastien Bubeck, Jian Ding, Ronen Eldan, and Miklós Z. Rácz. 2016. Testing for high-dimensional geometry in random graphs. *Random Structures & Algorithms* 49, 3 (2016), 503–532.

Trevor Campbell, Saifuddin Syed, Chiao-Yu Yang, Michael I Jordan, and Tamara Broderick. 2023. Local exchangeability. *Bernoulli* 29, 3 (2023), 2084–2100.

Olivia Caramello. 2013. Topological Galois Theory. *Advances in Mathematics* 291 (2013).

Olivia Caramello. 2014. Fraïssé's Construction from a Topos-Theoretic Perspective. *Logica Universalis* 8 (2014), 261–281.

Aurelio Carboni, Stephen Lack, and R F C Walters. 1993. Introduction to extensive and distributive categories. *J. Pure Appl. Algebra* 84, 2 (1993), 145–158.

François Caron and Emily B. Fox. 2017. Sparse Graphs Using Exchangeable Random Measures. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 79, 5 (09 2017), 1295–1366.

J R B Cockett. 1993. Introduction to distributive categories. *Math. Struct. Comput. Sci.* 3, 3 (1993), 277–307.

Harry Crane and Henry Towsner. 2018. Relatively exchangeable structures. *J. Symbolic Logic* 83, 2 (2018), 416–442.

Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. 2021. Categorical Foundations of Gradient-Based Learning. In *Proc. ESOP 2021*.

Fredrik Dahlqvist, Louis Parlant, and Alexandra Silva. 2018. Layer by layer: composing monads. In *Proc. ICTAC 2018*.

S. Dash, Y. Kaddar, H. Paquet, and S. Staton. 2023. Affine monads and lazy structures for Bayesian programming. In *Proc. POPL 2023*.

Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2018. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *Proc. ACM Program. Lang.* 2, POPL (2018), 59:1–59:28.

Thomas Ehrhard and Christine Tasson. 2019. Probabilistic call by push value. *Logical Methods in Computer Science* 15, 1 (2019).

P. Erdős and A. Rényi. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen* 6 (1959), 290.

M P Fiore. 2005. Mathematical models of computational and combinatorial structures. In *Proc. FOSSACS 2005*.

Brendan Fong, David Spivak, and Rémy Tuyéras. 2021. Backprop as Functor: A Compositional Perspective on Supervised Learning. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science* (Vancouver, Canada) (*LICS '19*). IEEE Press, Article 11, 13 pages.

Cameron Freer and Daniel Roy. 2010. Posterior distributions are computable from predictive distributions. In *Proc. 13th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2010)*. 233–240.

Cameron E. Freer and Daniel M. Roy. 2012. Computable de Finetti measures. *Annals of Pure and Applied Logic* 163, 5 (2012), 530–546.

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

Tobias Fritz, Fabio Gadducci, Paolo Perrone, and Davide Trotta. 2023. Weakly affine monads. *arXiv e-print 2303.14049* (2023).

Murdoch Gabbay and Andrew M Pitts. 1999. A New Approach to Abstract Syntax Involving Binders. In *Proc. LICS 1999*. 214–224.

Haim Gaifman. 1964. Concerning measures in first order calculi. *Israel J. Math.* 2 (1964), 1–18.

Michèle Giry. 1980. A categorical approach to probability theory. In *Categorical aspects of topology and analysis (Lecture Notes in Math., Vol. 915)*. 68–85.

Noah Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A universal language for generative models. In *Proc. 24th Conf. Uncertainty in Artificial Intelligence (UAI 2008)*. AUAI Press, 220–229.

N. D. Goodman and J. B. Tenenbaum. 2023. Probabilistic Models of Cognition. http://v1.probmods.org

Jean Goubault-Larrecq, Xiaodong Jia, and Clément Théron. 2021. A Domain-Theoretic Approach to Statistical Programming Languages. *arXiv e-print 2106.16190* (2021).

Claudio Hermida and Robert Tennent. 2012. Monoidal indeterminates and categories of possible worlds. *Theoretical Computer Science* 430 (2012).

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

Hongde Hu and Walter Tholen. 1995. Limits in free coproduct completions. *Journal of Pure and Applied Algebra* 105 (1995), 277–291.

Mathieu Huot and Sam Staton. 2018. Universal Properties in Quantum Theory. In *Proc. QPL 2018*.

Bart Jacobs. 1994. Semantics of weakening and contraction. *Ann. Pure & Appl. Logic* 69, 1 (1994), 73–106.

Bart Jacobs. 2016. *Introduction to coalgebra: Towards mathematics of states and observations.* CUP.

Bart Jacobs. 2018. From probability monads to commutative effectuses. *J. Log. Algebr. Methods Program.* 94 (2018), 200–237.

Bart Jacobs and Jorik Mandemaker. 2012. The Expectation Monad in Quantum Foundations. In Proceedings 8th International Workshop on *Quantum Physics and Logic (Electronic Proceedings in Theoretical Computer Science, Vol. 95)*. Open Publishing Association, 143–182.

Svante Janson. 2013. Graphons, cut norm and distance, couplings and rearrangements. *New York Journal of Mathematics Monographs* 4 (2013).

C Barry Jay. 1993. Tail recursion through universal invariants. *Theoret. Comput. Sci* 115, 1 (1993), 151–189.

Xiaodong Jia, Bert Lindenhovius, Michael W. Mislove, and Vladimir Zamdzhiev. 2021. Commutative Monads for Probabilistic Programming Languages. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 1–14.

Peter T. Johnstone. 2002. *Sketches of an Elephant – A Topos Theory Compendium.* OUP.

P. Jung, J. Lee, S. Staton, and H. Yang. 2021. A generalization of hierarchical exchangeability on trees to directed acyclic graphs. *Annales Henri Lebesgue* 4 (2021).

Younesse Kaddar and Sam Staton. 2023. A model of stochastic memoization and name generation in probabilistic programming: categorical semantics via monads on presheaf categories. In *Proc. MFPS 2023*. To appear.

Olav Kallenberg. 2010. *Foundations of Modern Probability* (2nd ed.). Springer New York.

Ohad Kammar. 2014. *An algebraic theory of type-and-effect systems.* Ph. D. Dissertation. University of Edinburgh.

Ohad Kammar and Dylan McDermott. 2018. Factorisation Systems for Logical Relations and Monadic Lifting in Type-and-effect System Semantics. In *Proc. MFPS 2018*. 239–260.

A. S. Kechris, V. G. Pestov, and S. Todorčević. 2005. Fraïssé limits, Ramsey theory, and topological dynamics of automorphism groups. *Geom. Funct. Anal.* 15, 1 (2005), 106–189.

Oleg Kiselyov and Chung-Chieh Shan. 2010. Probabilistic programming using first-class stores and first-class continuations. In *Proc. 2010 ACM SIGPLAN Workshop on ML*.

Bartek Klin, Sławomir Lasota, Joanna Ochremiak, and Szymon Toruńczyk. 2016. Homomorphism Problems for First-Order Definable Structures. In *Proc. FSTTCS 2016*.

Anders Kock. 1970. Monads on symmetric monoidal closed categories. *Arch. Math.* 21 (1970), 1–10.

Anders Kock. 2012. Commutative monads as a theory of distributions. *Theory Appl. Categ.* 26, 4 (2012), 97–131.

Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350.

H. Lindner. 1979. Affine parts of monads. *Arch. Math.* (1979).

László Lovász. 2012. *Large networks and graph limits.* Amer. Math. Soc., Providence, RI. xiv+475 pages.

László Lovász and Balázs Szegedy. 2006. Limits of dense graph sequences. *Journal of Combinatorial Theory, Series B* 96, 6 (2006), 933–957.

Saunders Mac Lane. 1998. *Categories for the Working Mathematician.* Springer.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv e-print 1404.0099* (2014).

David Marker. 2002. *Model Theory : An Introduction.* Springer Science & Business Media.

Cristina Matache, Sean K. Moss, and Sam Staton. 2022. Concrete categories and higher-order recursion: With applications including probability, differentiability, and full abstraction. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 57:1–57:14.

E. Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science* (Pacific Grove, California, USA). IEEE Press, 14–23.

Peter Orbanz and Daniel M. Roy. 2015. Bayesian Models of Graphs, Arrays and Other Exchangeable Random Structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 2 (2015), 437–461.

Duško Pavlović. 1997. Categorical logic of names and abstraction in action calculi. *Mathematical Structures in Computer Science* 7, 6 (1997), 619–637.

Mathew Penrose. 2003. *Random geometric graphs.* Oxford University Press.

Andrew M Pitts. 2001. Categorical logic. In *Handbook of Logic in Computer Science.* Vol. 5. OUP.

Andrew M. Pitts. 2013. *Nominal Sets: names and symmetry in computer science.* CUP.

Gordon D. Plotkin and John Power. 2002. Notions of Computation Determine Monads. In *Proc. FOSSACS 2002*.

A J Power. 2006a. Semantics for local computational effects. In *Proc. MFPS 2006*.

John Power. 2006b. Generic models for computational effects. *Theor. Comput. Sci.* 364, 2 (2006), 254–269.

R. Rado. 1964. Universal Graphs and Universal Functions. *Acta Arithmetica* 9 (1964), 331–340. Issue 4.

Daniel Roy, Vikash K. Mansinghka, Noah Goodman, and Joshua B. Tenenbaum. 2008. A stochastic programming perspective on nonparametric Bayes. In *Workshop on Nonparametric Bayes (co-located with ICML 2008)*.

Dan Shiebler. 2021. Categorical Stochastic Processes and Likelihood. *Compositionality* 3 (April 2021). Issue 1.

Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *Proc. ESOP 2017*. 855–879.

Sam Staton, Dario Stein, Hongseok Yang, Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. 2018. The Beta-Bernoulli Process and Algebraic Effects. In *Proc. ICALP 2018*. Appendix at arXiv:1802.09598.

Sam Staton, Hongseok Yang, Nathanael Ackerman, Cameron Freer, and Daniel M. Roy. 2017. Exchangeable Random Processes and Data Abstraction. Workshop on Probabilistic Programming Semantics (PPS 2017). https://pps2017.luddy.indiana.edu/files/2017/01/staton-yang-ackerman-freer-roy.pdf

Dario Stein. 2021. *Structural Foundations for Probabilistic Programming Languages.* Ph. D. Dissertation. University of Oxford.

Ross Street and Robert Walters. 1978. Yoneda structures on 2-categories. *Journal of Algebra* 50, 2 (1978), 350–379.

Terence Tao. 2013. Ultraproducts as a Bridge Between Discrete and Continuous Analysis. https://terrytao.wordpress.com/2013/12/07/

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

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *arXiv e-print 1809.10756* (2018).

Victor Veitch and Daniel M. Roy. 2019. Sampling and estimation for (sparse) exchangeable graphs. *The Annals of Statistics* 47, 6 (2019), 3274 – 3299.

R F C Walters. 1989. Data types in distributive categories. *Bulletin of the Australian Mathematical Society* 40 (1989), 79–82.

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proc. 17th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2014)*. 1024–1032.

Jeff Wu. 2013. *Reduced Traces and JITing in Church.* Master's thesis. Mass. Inst. of Tech.