Transfer Report

Younesse Kaddar

Supervisor: Sam Staton
Christ Church College

Contents

Abstract

In this report, we present five lines of work in the form of five "work packages". The first three work packages are more theoretical/foundational in nature: they pertain to understanding symmetries in probability, in the form of representation theorems (à la de Finetti and Aldous-Hoover) generalized to arbitrary exchangeable data types in probabilistic programming (section 2). We resort to categorical and model-theoretic tools falling under the general framework of toposic Galois theory (section 3). These ideas are then applied to giving a denotational semantics to stochastic memoization in a restricted setting (section 4).

The last two work packages are more applied: they concern the LazyPPL Haskell probabilistic programming library, developed and used by our team to express nonparametric Bayesian models declaratively, by taking advantage of Haskell's laziness (section 5). We then mention the idea of leveraging program synthesis and machine learning in deep probabilistic programming for automatic model generation (section 6).

1 Introduction

> I know of scarcely anything so apt to impress the imagination as the wonderful form of cosmic order expressed by the "Law of Frequency of Error". The law would have been personified by the Greeks and deified, if they had known of it. [...] The huger the mob, and the greater the apparent anarchy, the more perfect is its sway. [...] Whenever a large sample of chaotic elements are taken in hand and marshaled in the order of their magnitude, an unsuspected and most beautiful form of regularity proves to have been latent all along.

In this quote [Gal94], Galton – Darwin's half-cousin and one of the fathers of modern applied statistics – expresses his amazement at what later became, with Kolmogorov's rigorous formalism of probability theory [Kol46], the Central Limit Theorem. He went on to use it as the basis of numerous statistical methods in the fields, among others, of biology, anthropology, and psychology. The Central Limit Theorem, tellingly referred to as the "unofficial sovereign of probability theory" by Tijms [Tij07], states that the empirical mean of *independently and identically distributed* (iid) random variables tends toward a normally distributed variable. Since no assumption is made on the distribution of the original random variables, one of the far-reaching consequences of this theorem is that probabilistic methods pertaining to normal distributions have a larger range of applications to other kinds of distributions. However, similarly to many other key theorems in probability theory (law of large numbers, limit theorems, ...), it relies on an arguably strong assumption: the original variables being (jointly) *independent*. That is, their joint distribution ought to be determined by the individual (marginal) distributions, which greatly simplifies calculations of various derived probabilistic quantities.

But, taking a step back to more epistemological considerations, independence is not a property that can easily be externally observed in a stochastic process. Thus, it is usually left as a strong

statistical modeling assumption inherent to the underlying random variables that generated the observed data. A weaker notion that may be observed in practice (to a certain extent) is invariance under various symmetries, such as exchangeability. Such symmetry invariances are clearly implied by independence, but the converse does not hold in general (example 2.2). This is where representation theorems, such as de Finetti's theorem (theorem 2.3, for sequences of iid random variables) or Aldous-Hoover (theorem 2.10, for random arrays), come into play, by providing a sort-of reciprocal. Provided that the random variables are exchangeable (*i.e.* their joint probability distribution is permutation-invariant), they are independent *conditionally* on a latent random variable (*e.g.* in the case of coin tosses, the bias of the coin). So exchangeability (definition 2.1) plays a key role at a foundational level, as well as – as we will see later (section 2.2.3) – for more practical purposes.

2 Exchangeability and Representation theorems

2.1 de Finetti's Theorem

de Finetti's representation theorem [Fin37] is one of the most celebrated theorems in Bayesian statistics [DS18], partly because it is, at a fundamental level, at the root of the justification of statistical models with parameters distributed according to a prior distribution. As such, it can be regarded as one of the basis for both Bayesian and frequentist models involving iid random variables (where datapoints are often assumed to be generated by such sequences of iid random variables with distribution parameterized by an unknown parameter) [ONe09]. It relies on a notion of permutation-invariance of the joint distribution called *exchangeability*.

Definition 2.1 (Exchangeability)**.** A countable sequence of random variables $(X_n)_{n\in\mathbb{N}}$ valued in a standard Borel space is *exchangeable* iff the sequence $(X_n)_n$ is equal in distribution to the sequence $(X_{\sigma(n)})_n$ for every permutation $\sigma \in \mathfrak{S}(\mathbb{N})$ of finitely many indices, written:

$$(X_n)_{n\in\mathbb{N}} \overset{\mathcal{D}}{=} (X_{\sigma(n)})_{n\in\mathbb{N}}$$

Concretely, exchangeability expresses the fact that the probability of a particular sequence of observations does not depend on how these observations are ordered in the sequence. Joint independence clearly implies exchangeability, but the converse is not true.

Example 2.2 (Exchangeability is weaker than independence)**.** *Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of iid random variables, and consider the family:*

$$(X_0 + X_n)_{n\geq 1}$$

It can readily be shown to be exchangeable, but it is clearly not independent as soon as X_0 is non-deterministic. However, conditionally *on X_0, it is independent.*

In essence, de Finetti's theorem states that exchangeability does imply a weaker form of independence though: exchangeable sequences of observations are independent *conditionally* on some latent variable, or, equivalently, they can be written as mixtures of underlying iid sequences.

We write here two equivalent formulations of de Finetti's theorem (more precisely, a generalization thereof due to Hewitt and Savage), the first one being more convenient for conveying intuition (in our opinion), and the second one more amenable to generalizations to higher dimensions [Pan19].

Theorem 2.3 (de Finetti–Hewitt–Savage [Fin37; HS55])**.**
Empirical measure version: *A sequence $(X_i \colon \Omega \to S)_{i\geq 1}$ of random variables valued in a standard Borel space (complete separable metric space) $(S, \mathscr{B})$ (where $\mathscr{B}$ is the Borel σ-algebra) is exchangeable iff there almost surely exists a random probability measure μ, called the* limiting empirical measure*, which is a random element in the space $\mathcal{P}(S, \mathscr{B})$ of probability measures on $(S, \mathscr{B})$ (equipped with the topology of weak convergence) such that*

- *μ is the limit of the sequence of random* empirical measures*:*

$$\mu = \lim_{n \longrightarrow \infty} \mu_n$$

  where each random empirical measure $\mu_n \colon \Omega \to \mathcal{P}(S, \mathscr{B})$ is given by ($\mathbf{I}$ denotes the indicator function):

$$\mu_n(\omega)(A) \;:=\; \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}(X_i(\omega) \in A) \qquad \text{for every } A \in \mathscr{B},\ \omega \in \Omega$$

- *conditionally on μ, the sequence $(X_i)_{i\geq 1}$ is iid from the distribution μ:*

$$P\big(X_1 \in A_1, \ldots, X_n \in A_n \mid \mu\big) = \prod_{i=1}^{n} \mu(A_i) \quad \text{for all } n, A_1, \ldots, A_n$$

  or, more concisely:

$$P\big((X_i)_{i\geq 1} \in - \mid \mu\big) = \mu^{\otimes \infty}(-)$$

This is equally to say that the joint distribution π of the X_i's is a mixture of distributions of iid sequences: there exists a probability measure $\mathbf{M}$ (Bayesian prior, which is the distribution of the limiting empirical measure) on the space $\mathcal{P}(S, \mathscr{B})$ of probability measures on $(S, \mathscr{B})$ such that

$$P(X_1 \in A_1, \ldots, X_n \in A_n) = \int \prod_{i=1}^{n} \mu(A_i)\, \mathbf{M}(\mathrm{d}\mu)$$

or, more concisely, we have the integral representation:

$$\pi = \int \mu^{\otimes \infty}\, \mathbf{M}(\mathrm{d}\mu)$$

Noise-outsourcing version: *A sequence $(X_i)_{i\geq 1}$ of random variables valued in a Borel space $(S, \mathscr{B})$ is exchangeable iff there exists a measurable function $g\colon [0,1]^2 \to \mathbb{R}$ such that*

$$(X_i)_{i\geq 1} \overset{\mathcal{D}}{=} (g(w, u_i))_{i\geq 1}$$

where w, u_i ($i \geq 1$) are all iid uniform variables on $[0,1]$. The sequence $(g(w,u_i))_{i\geq 1}$ is called a representation *of the original sequence.*

Proof sketch. The main idea of the proof is to show that for every partition of the index set $\mathbb{N}_{\geq 1}$ into two *infinite* blocks I and $J := \mathbb{N}_{\geq 1}\backslash I$, the X_i's for $i \in I$ are iid *conditionally* on the X_j's for $j \in J = \mathbb{N}_{\geq 1}\backslash I$. The result will follow, because the limiting frequency can be computed from the X_j's, and, by exchangeability, we have:

$$(X_k)_{k\in K} \overset{\mathcal{D}}{=} (X_\ell)_{\ell\in\mathbb{N}_{\geq 1}} \qquad \text{for every infinite } K \subseteq \mathbb{N}_{\geq 1} \tag{1}$$

so in particular for $K := I$, the joint distribution of the X_i's is the joint distribution of the whole sequence.

The X_i's are independent conditionally on the X_j's: This is the trickiest part of the proof. To prove it, we use the machinery of conditional expectations, which have a couple of convenient properties that we will take advantage of. The desired independence is equivalent to showing, by induction on $n \geq 1$, that for every $i_1, \ldots, i_n \in I$ and $f_1, \ldots, f_n\colon S \to \mathbb{R}$ bounded measurable, we almost surely have:

$$\mathbb{E}\Big(\prod_{k=1}^{n} f_k(X_{i_k}) \mid \sigma(X_j : j \in J)\Big) = \prod_{k=1}^{n} \mathbb{E}\Big(f_k(X_{i_k}) \mid \sigma(X_j : j \in J)\Big) \tag{2}$$

where $\sigma(X_j : j \in J)$ denotes the σ-algebra generated by the X_j's. But the random variable on the left is, by definition, the unique (up to almost sure equality) random variable Y satisfying:

$$\mathbb{E}\Big(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k})\Big) = \mathbb{E}\left(\mathbf{I}_A\, Y\right) \qquad \text{for all } A \in \sigma(X_j : j \in J) \tag{3}$$

So eq. (2) is equivalent to showing, for all $A \in \sigma(X_j : j \in J)$:

$$\mathbb{E}\Big(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k})\Big) = \mathbb{E}\Big(\mathbf{I}_A \prod_{k=1}^{n} \mathbb{E}\Big(f_k(X_{i_k}) \mid \sigma(X_j : j \in J)\Big)\Big) \tag{4}$$

But the left-hand side (LHS)

$$\mathbb{E}\left(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k})\right) = \mathbb{E}\left(\mathbb{E}\left(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k}) \mid \sigma(X_k : k \in \mathbb{N}_{\geq 1}\backslash\{i_n\})\right)\right)$$

(by the law of iterated expectations) is equal to

$$\mathbb{E}\left(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k}) \underbrace{\mathbb{E}\left(f_n(X_{i_n}) \mid \sigma(X_k : k \in \mathbb{N}_{\geq 1}\backslash\{i_n\})\right)}_{\overset{\circledast}{=}\ \mathbb{E}\left(f_n(X_{i_n}) \ \mid\ \sigma(X_k : k \in J)\right)}\right)$$

by $\sigma(X_k : k \in \mathbb{N}_{\geq 1}\backslash\{i_n\})$-measurability of $\mathbf{I}_A, f_1(X_{i_1}), \ldots, f_{n-1}(X_{i_{n-1}})$. Assuming $\circledast$, we then have the following (which yields the desired eq. (2) by induction on n), by eq. (3):

$$\begin{aligned}\mathbb{E}\left(\prod_{k=1}^{n} f_k(X_{i_k}) \mid \sigma(X_j : j \in J)\right) &= \mathbb{E}\left(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k})\, \mathbb{E}\left(f_n(X_{i_n}) \mid \sigma(X_k : k \in J)\right) \mid \sigma(X_j : j \in J)\right) \\ &= \mathbb{E}\left(\mathbf{I}_A \prod_{k=1}^{n} f_k(X_{i_k}) \mid \sigma(X_j : j \in J)\right)\, \mathbb{E}\left(f_n(X_{i_n}) \mid \sigma(X_k : k \in J)\right)\end{aligned}$$

The equality $\circledast$ can be proved by observing that the RHS is the conditional expectation of the LHS given $\sigma(X_j : j \in J)$ (by the tower property of conditional expectations since $\sigma(X_j : j \in J) \subseteq \sigma(X_k : k \in \mathbb{N}_{\geq 1}\backslash\{i_n\})$). So, in the L^2 space (on the underlying sample space), the RHS is the orthogonal projection of the LHS on the subspace of $\sigma(X_j : j \in J)$-measurable functions. But both random variables have the same distribution (by exchangeability eq. (1)), and hence the same L^2-norm, so they are almost surely equal.

The X_i's have the same distribution conditionally on the X_j's: This follows from exchangeability (eq. (1)), since all the joint distributions $(X_i, X_j)_{j\in J}$, for $i \in I$, are equal. So conditioning each X_i on $(X_j)_{j\in J}$, for $i \in I$, yields the same distribution. □

Example 2.4 (Boolean case)**.** *de Finetti's original 1931 proof was in the Boolean case, where the X_i's are exchangeable Bernoulli-distributed coin flips ($X_i \in \{0, 1\}$ for every $i \geq 1$). In this setting, the empirical measure version of the theorem takes an intuitive meaning: the empirical average $\overline{X}_n := \frac{1}{n}\sum_{1\leq i\leq n} X_i$ converges to the limiting frequency $\overline{X}_\infty = \lim_{n\to\infty} \sum_{1\leq i\leq n} X_i/n$ (even almost surely in this case, by the strong law of large numbers). So de Finetti states that the observables being exchangeable implies the existence of a parameter $\overline{X}_\infty$ taking values in $[0, 1]$ (the asymptotic empirical bias of the coin) such that, given the value of the bias $\overline{X}_\infty$, the observables are conditionally iid:*

$$\forall n \geq 1,\ P(X_1 = x_1, \ldots, X_n = x_n \mid \overline{X}_\infty = \theta) = \theta^{\sum_i x_i}(1-\theta)^{n-\sum_i x_i}$$

Pólya's urn is another important example involving Boolean-valued exchangeable variables that we will come back to later:

Example 2.5 (Pólya urn model)**.** *The Pólya urn model is a stochastic process producing a countable sequence of exchangeable (but not independent) Booleans. Consider an urn that initially contains i balls labelled true and j balls labelled false. A ball is uniformly drawn at random (its label is the value of the draw) and is replaced in the urn together with a new ball of the same label. The process is then repeated. Let $X_k = 1$ if the k-th draw is a ball marked true and $X_k = 0$ otherwise. The empirical average $\overline{X}_n := \frac{1}{n}\sum_{1\leq k\leq n} X_k$ converges to the limiting frequency $\overline{X}_\infty = \lim_{n\to\infty} \overline{X}_n$ which has a Beta distribution $\mathcal{B}(i,j)$. Conditionally on $\overline{X}_\infty = \theta$, the X_k's are independent and Bernoulli-distributed with parameter θ.*

So the empirical measure version of de Finetti's theorem states that the joint distribution of an exchangeable sequence of random variables valued in a standard Borel space is entirely determined by a distribution of probability measures on this space (the distribution of the limiting empirical measure). This, in turn, enables us to decompose the distribution of the random sequence into a (continuous) weighted sum of conditionally independent components, the weight being given by a prior on the underlying parameter of the statistical model.

This is what brought Diaconis, Persi and Skyrms [DS18, p. 124] to say:

> de Finetti's theorem helps dispel the mystery of where the prior belief over the chances comes from. From exchangeable degrees of belief, de Finetti recovers both the chance statistical model of coin flipping and the Bayesian prior probability over the chances. The mathematics of inductive inference is just the same. If you were worried about where Bayes' priors came from, if you were worried about whether chances exist, you can forget your worries. de Finetti has replaced them with a symmetry condition on degrees of belief. This is, we think you will agree, a philosophically sensational result.

The noise-outsourcing version of de Finetti's theorem (theorem 2.3) takes advantage of the key following noise-outsourcing lemma:

Lemma 2.6 (Noise-outsourcing lemma)**.** *Let (X,Y) be random variables taking values in a measurable space $(S,\mathscr{S})$ and a Borel space $(T,\mathscr{B})$. Then, there exists a measurable function $f\colon S\times[0,1]\to T$ and a uniform random variable u on $[0,1]$ independent of X such that*

$$(X,Y) \overset{\mathcal{D}}{=} (X, f(X,u))$$

In other words, conditionally on X, Y can be expressed as a function of X and an independent uniform random variable u (which encodes the conditional randomness of Y given X).

Remark 2.7. If $S \cong \{*\}$ is a one-point space (so that X is deterministic), this is the standard fact that the law of every Borel-valued random variable Y can be obtained as a pushforward of the uniform measure $[0,1]$ (sometimes called randomization lemma).

In the representation $(g(w,u_i))_{i\geq 1}$ of the X_i's in the noise-outsourcing version of de Finetti's theorem, the uniform random variables w, u_i ($i \geq 1$) manifest the fact that, conditionally on a function of a random variable w valued in a Borel space, the sequence is iid (because the $g(w,u_i)$'s are iid when w is fixed).

Noise-outsourcing lemma and quasi-Borel spaces The noise-outsourcing lemma plays an important role, among many other things, in the implementation of the LazyPPL probabilistic programming library [PS21; Sta+] (section 5) (on the practical side) and in the axiomatization of quasi-Borel spaces [Heu+17; Heu+18] (on the theoretical side). The latter provide a well-behaved categorical model of probabilistic programming languages (whose types are interpreted as quasi-Borel spaces), supporting both function spaces (cartesian closedness) and a strong commutative monad of measures, contrary to the more standard category of measurable spaces[1].

Definition 2.8 (Category of quasi-Borel spaces [Heu+17])**.** Let Ω be a fixed uncountable standard Borel space. A *quasi-Borel space* (X, M_X) is a set X equipped with a set $M_X \subseteq X^\Omega$ of functions called *admissible random elements*, satisfying

- **deterministic random elements:** M_X contains all the constant functions.
- **stability under pre-composition with measurable functions**: for every measurable function $f\colon \Omega \to \Omega$ and every admissible random element $M_X \ni \alpha\colon \Omega \to X$, the composition $\Omega \xrightarrow{f} \Omega \xrightarrow{\alpha} X$ is in M_X.
- **gluing property**: if $(\alpha_i)_{i\in\mathbb{N}} \in M_X^{\mathbb{N}}$ is a countable family of admissible random elements and if $\Omega = \biguplus_{n=1}^{\infty} U_i$ is a covering of Ω where each U_i is Borel, then the amalgamation $\tilde{\alpha}$ is in M_X, where $\tilde{\alpha}(\omega) := \alpha_i(\omega)$ for every $\omega \in U_i$.

A morphism $f\colon (X, M_X) \to (Y, M_Y)$ of quasi-Borel spaces is a function $f\colon X \to Y$ such that for all $\alpha \in M_X$, $\Omega \xrightarrow{\alpha} X \xrightarrow{f} Y$ is in M_Y.

The category of standard Borel spaces fully embeds in the category of quasi-Borel spaces, where M_X is taken to be the measurable functions. So quasi-Borel spaces conservatively extend standard Borel spaces, while being more categorically/type-theoretically well-behaved. As such, they can be considered as an instantiation of *synthetic measure theory*, and synthetic approaches have also been used to tackle de Finetti's theorem over the past few years.

[1] which is not cartesian closed and not known to have a commutative monad of measures

2.1.1 Synthetic versions of de Finetti's Theorem

Synthetic probability and measure theory are new, emerging fields coming in various flavors [CJ19; Fri20; Heu+17; Koc11; Sim17]. One of their multiple aims is to unravel the structural properties of probability and measure theory from an axiomatic and algebraic standpoint, by changing the focus to the underlying categorical/axiomatic/type-theoretic foundations rather than models thereof (such as traditional measure theory).

de Finetti's theorem has recently been broached from the point of view of synthetic probability theory by:

- Jacobs and Staton [JS20] in the Boolean case, in the form of a categorical limit existence theorem.
- Fritz, Gonda and Perrone [FGP21], in the setting of Markov categories [Fri20].

The success of these approaches raises the question of whether we can push the analysis further to other kinds of representation theorems in probability theory. An convenient concrete framework to do so is *probabilistic programming*, which can be regarded as the internal language of synthetic/categorical probability theory [Ste21]. As such, it is a powerful tool to express and study symmetries in probability from an abstract and general point of view.

2.2 Exchangeable modules in probabilistic programming

Probabilistic programming. As previously alluded to, statistical models can also be seen through the lens of probabilistic programming [Bor+17; Rai17; Sta17; Sta20b; Ste21; Tol+16; vdMee+21], which constitutes a powerful and versatile way to express and analyze them, drawing on the machinery of programming language theory and synthetic approaches to probability theory. Probabilistic programming, a form of Bayesian machine learning, distinguishes itself from traditional programming by the addition of three new constructs – conveniently keeping the declaration of the statistical model and the statistical inference separate – each one corresponding to a clause of Bayes' law

$$\underbrace{p(x \mid d)}_{\text{posterior}} \propto \underbrace{p(d \mid x)}_{\text{likelihood}} \times \underbrace{p(x)}_{\text{prior}}$$

- prior: sample samples a random value from a prior distribution. This is the generative part, in statistical parlance.
- likelihood: score (or observe): scores a value according to a likelihood function (thereby forming an unnormalized measure) to perform (soft and/or exact) conditioning on the observed datapoints d. This process enables us to perform inference from effects (observations d) to causes (parameters x), hence the name "inverse probability" it was usually given before the 20's [Fie06]. This is the discriminative part, in statistical parlance.

- posterior: `normalize` (or `infer`): normalizes the resulting unnormalized measure obtained after applying likelihood weights, resulting in a posterior probability distribution. This is the inference part, in statistical parlance. Rather than yielding a probability distribution, in practice, it is enough for this construct to output a stream of samples from the unnormalized measure (obtained from a inference algorithm such as one of the Monte Carlo methods for example, which do not require to compute the normalizing constant). This is the approach taken in the Haskell library LazyPPL [PS21; Sta+] (section 5).

In the following (until section 5), we will mainly focus on generative models (setting aside the conditioning aspects for now).

2.2.1 de Finetti on a Boolean exchangeable module

Let us revisit the Boolean stochastic process examples (examples 2.4 and 2.5) in the context of probabilistic programming. Following [Sta+18], such processes can be presented as the interface (or signature) of an abstract module (done in a ML-like language in the paper), that we implement here in LazyPPL. The corresponding typeclass can be written as follows:

```
class BoolProcess hyperparam process where
    new :: hyperparam → Prob process
    get :: process → Prob Bool
```

A new Boolean process (of type `process`) parameterized by a hyperparameter (`hyperparam`) can be initialized at random with `new`, and we can then fetch a random sequence of Booleans from it by iteratively calling `get`. The abstract type of the process, in the form of a typeclass `BoolProcess` – that we will henceforth refer to as *module*, following the ML tradition – hides away implementation details, such as the data structure or internal state used to sample from the process. The observational behavior of such a module can be studied with an operational or adequate denotational semantics, irrespective of how the operations `new` and `get` are implemented under the hood, which makes for a powerful way to abstract away its properties. Following Staton [Sta20a; Sta21a; Sta22], we will refer to this "invariance under implementation details" principle as the *abstract type property* in the sequel.

Here are two instantiations of the module `BoolProcess` in LazyPPL: the left-hand one being an implementation of Pólya's urn (initiated with i `true` balls and j `false` balls), and the right-hand one of the Beta-Bernoulli process (Bernoulli random variables, where the prior on the coin bias is Beta-distributed).

```
newtype Polya = Polya (IORef (Int, Int))

instance BoolProcess (Int,Int) Polya where
  new (i,j) = return
    $ Polya $ unsafePerformIO $ newIORef (i,j)

  get (Polya ref) = do
    let (i,j) = unsafePerformIO $ readIORef ref
    b ← bernoulli
      (fromIntegral i/fromIntegral (i+j))
    if b then return
      $ unsafePerformIO $ writeIORef ref (i+1,j)
      >> return True
    else return
      $ unsafePerformIO $ writeIORef ref (i,j+1)
      >> return False
```

```
newtype BetaBern = BetaBern Double

instance BoolProcess (Int,Int) BetaBern where
  new (i,j) = do
    θ ← beta (fromIntegral i) (fromIntegral j)
    return $ BetaBern θ

  get (BetaBern θ) = bernoulli θ
```

As previously seen (example 2.5), a countable Boolean random sequence $(X_k)_{k\geq 1}$ generated by Polya is exchangeable, and de Finetti's theorem provides a representation of such a sequence: conditionally on $\overline{X}_\infty = \theta$, where $\overline{X}_\infty \sim \mathcal{B}(i,j)$ is the limiting frequency, the X_k's are independent and Bernoulli-distributed with biais θ. This is precisely what BetaBern is implementing. So Polya and BetaBern induce the same distribution on Boolean sequences, it can be shown that they are observationally equivalent [Sta+18], and de Finetti's theorem represents the former by the latter.

Besides the obvious simplification and stateless code transformation of the implementation (BetaBern no longer uses mutable references and unsafePerformIO like Polya), the conceptual clarity gained from it, and the theoretical significance of such a representation (2.1), it comes with a number of other key practical advantages [Fre]:

- **parallelization**: in BetaBern, successive calls to get can easily be parallelized, since we only need to know the coin bias. This is not the case for Polya, as every draw changes the internal state of the urn and thus depends on all the previous ones.
- **potential inference improvements**: in a MCMC inference algorithm, changing the site corresponding to any one of the X_k's in BetaBern has no impact on the others, since they are all conditionally independent. In Polya however, any change to one of the X_k's systematically affects all the subsequent ones that depend on it.
- **more efficient computations**: by conditional independence, the joint conditional distribution in BoolProcess can be factorized as a product of simpler independent factors, which can lead to faster computations.

2.2.2 Aldous-Hoover theorem

Exchangeable arrays. The natural question of whether we can extend the previous analysis to a module where get now takes as input a pair of processes (get :: (process, process) → Prob **Bool**) brings us to the setting of 2-dimensional random arrays [Ald81; Aus; OR15]. Random exchangeable arrays are a useful data analysis tool to model relational data, which are observations of binary relationships between collections of objects, *e.g.* graph social networks in social network analysis, world wide web, biochemical pathways, etc. (we mention other examples in section 5.1; the previously seen Boolean sequences can be seen as 1-dimensional arrays, modeling non-relational observations about individual objects). Such binary relations define 2-dimensional arrays $(X_{i,j})_{i,j}$ of Boolean random variables, where i and j range over countable collections of potentially related objects. When we have a single (countable) collection of objects that we wish to compare pairwise (*i.e.* i and j range over the same countable index), which we will assume in what follows, the 2-dimensional random array $(X_{i,j})_{i,j}$ can be seen as the adjacency matrix a random graph. Naturally, we also have a notion of exchangeability for random adjacency matrices $(X_{i,j})_{i,j}$, namely that the joint distribution is invariant under relabeling the nodes of the corresponding graph.

Definition 2.9 (Exchangeability of random arrays)**.** A random array $(X_{i,j})_{i,j\in\mathbb{N}}$ is said to be jointly (or weakly) exchangeable iff for any permutation $\sigma \in \mathfrak{S}(\mathbb{N})$ of finitely many indices, we have:

$$(X_{i,j})_{i,j\in\mathbb{N}} \overset{\mathcal{D}}{=} (X_{\sigma(i),\sigma(j)})_{i,j\in\mathbb{N}}$$

A random graph is said to be exchangeable iff its adjacency matrix is jointly exchangeable.

Jointly exchangeable arrays enjoy a celebrated representation theorem too: the Aldous-Hoover representation theorem [Ald81; Hoo79; Kal89] (which can be shown with conditional expectation techniques, in a similar way to de Finetti's theorem).

Theorem 2.10 (Aldous-Hoover [Ald81; Hoo79])**.** *A random array $(X_{i,j})_{i,j\in\mathbb{N}}$ is jointly exchangeable iff it can be represented as follows: there exists a measurable function $g\colon [0,1]^4 \to \mathbb{R}$ such that*

$$(X_{i,j})_{i,j\geq 1} \overset{\mathcal{D}}{=} (g(w, u_i, u_j, u_{\{i,j\}}))_{i,j\geq 1}$$

where w, u_i ($i \geq 1$) and $u_{\{i,j\}}$ ($i, j \geq 1$) are all iid uniform variables on $[0,1]$ (with the convention that $u_{\{i\}} = u_i$ on the diagonal).

Remark 2.11. The random variables $u_{\{i,j\}}$ being indexed by sets denotes the fact that i, j are unordered, so that the random array $(u_{\{i,j\}})_{i,j\geq 1}$ can be seen as triagonal. If g is additionally symmetric in its second and third arguments, *i.e.* if it satisfies $g(-, x, y, =) = g(-, y, x, =)$ for all x, y, then $(X_{i,j})_{i,j\in\mathbb{N}}$ is symmetric too, *i.e.* $X_{i,j} = X_{j,i}$ for all i, j. The corresponding random graph is then undirected.

Aldous-Hoover and graphons. A noteworthy example where the Aldous-Hoover representation takes a concrete and intuitive form is the case of random simple (*i.e.* undirected and self-loop-free) graphs, whose adjacency matrix $(X_{i,j})_{i,j\in\mathbb{N}}$ is symmetric and has zero diagonal [DJ07; Fre; Lov12; OR15; Roy]. In this case, the exchangeable adjacency matrix can be parametrized by a (random) measurable function $G\colon [0,1]^2 \to [0,1]$ called a *graphon*. If u_i, u_j are iid random variables corresponding to two nodes $i, j \in \mathbb{N}$ in the random simple graph, $G(u_i, u_j)$ gives the probability of there being an edge between i and j.

Lemma 2.12 (Aldous-Hoover for simple graphs [OR15]). *Let $(X_{i,j})_{i,j\in\mathbb{N}}$ be the random adjacency matrix of a simple graph. It is jointly exchangeable iff there is a random graphon G (*i.e. *a random measurable function from $[0,1]^2$ to $[0,1]$) and iid uniform variables $(u_i)_{i\in\mathbb{N}}$ and $(u_{\{i,j\}})_{i,j\in\mathbb{N}}$ on $[0,1]$ (all independent of G) such that*

$$(X_{i,j})_{i,j\in\mathbb{N}} \overset{\mathcal{D}}{=} \left(\mathbf{I}(u_{\{i,j\}} < G(u_i, u_j))\right)_{i,j\in\mathbb{N}}$$

Moreover, if $(g(w, u_i, u_j, u_{\{i,j\}}))_{i,j\geq 1}$ is the Aldous-Hoover representation (theorem 2.10) of $(X_{i,j})_{i,j\in\mathbb{N}}$, G can be defined as

$$G(x,y) \;:=\; P(g(w,x,y,u) = 1|w) = \int_0^1 g(w,x,y,u)\,\mathrm{d}u \qquad G(x,x) = 0 \quad \forall x \neq y \in [0,1]$$

where $u \sim \mathrm{Unif}([0,1])$ is independent of G.

2.2.3 Abstract exchangeable modules

As a consequence, we have the neat result that Bayesian models modeling exchangeable simple graphs are entirely determined (in distribution) by a prior on the space of graphons. Graphons will come back later, when we will consider a topos-theoretic model for a small language where we can sample from a random graph (section 3), and in the Mondrian process LazyPPL example (section 5.1).

General exchangeable modules. The two previous examples (exchangeable Boolean sequences and exchangeable graphs), each coming with a representation theorem (de Finetti and Aldous-Hoover), naturally raise the question of whether (and to what extent) this probabilistic representation phenomenon carries over to other kinds of generative processes.

In the Pólya urn example, de Finetti's theorem turned a stateful module (which had the abstract type property and was generating exchangeable sequences) for the Boolean process interface into an equivalent stateless module (so mutable state and finite probability have been replaced by stateless continuous probability). Abstracting away this example, one may wonder: given an "exchangeable" stateful module with a given interface, can we find an equivalent stateless module probabilistic module with the same interface?

But to even begin formalizing such a conjecture, we need to define a general notion of exchangeability for such processes. At first glance, this might seem like a daunting task, because exchangeability means invariance of the joint distribution under certain kinds of symmetries of the random structures at hand, but at such a general level, we cannot easily reason on concrete instances (such a sequences or arrays) anymore. However, Staton, Yang, Ackerman, Freer and Roy still managed to overcome this difficulty, by elegantly framing the problem in programming language theory [Sta+17]. It all starts with a commutativity and discardability property called the *dataflow property*.

Definition 2.13 (Dataflow property [Sta20b])**.** A programming language is said to have the *dataflow property* iff program lines can be reordered (commutativity) and discarded (discardability, or affineness) provided that the dataflow is preserved. In other words, the language satisfies the following commutativity and discardabily equations:

$$(\textbf{let}\ \mathsf{x}_1 = \mathsf{M}_1\ \textbf{in}\ \mathsf{x}_2 = \mathsf{M}_2\ \textbf{in}\ \mathsf{N}) = (\textbf{let}\ \mathsf{x}_2 = \mathsf{M}_2\ \textbf{in}\ \mathsf{x}_1 = \mathsf{M}_1\ \textbf{in}\ \mathsf{N})$$
$$(\textbf{let}\ \mathsf{x}_1 = \mathsf{M}_1\ \textbf{in}\ \mathsf{M}_2) = \mathsf{M}_2$$

where $\mathsf{x}_1 \notin \mathsf{FV}(\mathsf{M}_2)$ and $\mathsf{x}_2 \notin \mathsf{FV}(\mathsf{M}_1)$.

Remark 2.14. The dataflow property expresses the fact that, to give a meaning to programs, the only thing that matters is the topology of dataflow diagrams. In other words, the semantics of programs is stable under symmetries of their dataflow diagrams.

It turns out that this idea of commutativity and discardability is regarded, in synthetic probability theory, by various authors as a fundamental aspect of the abstract axiomatization of probability:

- Kock [Koc11] argues that any monad that is strong commutative and affine can be abstractly viewed as a probability monad.
- Affine (or semi-cartesian) monoidal categories (*i.e.* monoidal categories where the tensorial unit is terminal), of which the category of probability kernels is an example, are used as a basic setting for synthetic probability by several authors [CJ19; Fri20; Sta17; SS21; Ste21].

More concretely, in a probabilistic programming language, the dataflow property corresponds to Fubini's theorem in the denotational semantics.

Example 2.15 (Commutativity is Fubini's theorem and discardability marginalization)**.** *For example, the quasi-Borel spaces (QBS) categorical model (definition 2.8) satifies the dataflow property. In this model, the meaning of a probabilistic program p returning random values of type X is a QBS probability measure, which can be seen as an expectation calculator of the form*

$$[\![p]\!] := X \xrightarrow{k} \Omega \longmapsto \int_\Omega k(\alpha(\omega))\,\mathrm{d}\omega \in P(X) \subseteq ((X \to \Omega) \to \Omega)$$

where $k\colon X \to \Omega$ a QBS morphism and $M_X \ni \alpha\colon \Omega \to X$ is the admissible random element corresponding to the interpretation of the program as a measure on the QBS X (the monad of

probability measures is a submonad of the continuation monad generated by admissible random elements). Then, if the program makes, say, n consecutive samples and $\omega := (\omega_1, \ldots, \omega_n)$, the denotation of the program p^σ where the sample statements are shuffled by a permutation σ is

$$[\![p^\sigma]\!](k) := \int_\Omega k(\alpha(\omega_1, \ldots, \omega_n))\, \mathrm{d}(\omega_{\sigma(n)}, \ldots, \omega_{\sigma(1)}) = \int_\Omega k(\alpha(\omega_1, \ldots, \omega_n))\, \mathrm{d}(\omega_n, \ldots, \omega_1) = [\![p]\!](k)$$

This is Fubini's theorem (integrals over probability spaces can be reordered). In a similar vein, discardability corresponds to marginalizing unused variables.

This leads us to Staton et al.'s general definition of exchangeability for data types (or modules): an abstract data type for a random process expressed in a probabilistic programming language is said to be *exchangeable* iff adding it to the language does not break the dataflow property.

Definition 2.16 (Data type exchangeability [Sta+17])**.** A data type is *exchangeable* iff it satisfies the dataflow property (commutativity and discardability) and the abstract type property (invariance under implementation details).

Intuitively, the dataflow property is required because if p :: Prob a, then saying (for a permutation σ) that

```
sample_n :: Meas(a, ..., a) = do
  x_1 ← sample p
      ⋮
  x_n ← sample p
  return (x_1, ..., x_n)
```

has the same semantics (yields the same measure or expectation calculator) as

```
sample^σ_n :: Meas(a, ..., a) = do
  x_σ(1) ← sample p
      ⋮
  x_σ(n) ← sample p
  return (x_1, ..., x_n)
```

α-renaming $=$

```
sample^σ_n :: Meas(a, ..., a) = do
  x_1 ← sample p
      ⋮
  x_n ← sample p
  return (x_σ⁻¹(1), ..., x_σ⁻¹(n))
```

is precisely saying that the joint distribution is invariant under permuting finitely many indices.

Additionally, the abstract type property enforces the fact that we cannot inspect how the module operations for a given interface are specifically implemented when using the module (by, for example, looking inside Pólya's urn, comparing two processes if they are real-valued biases, etc.). The contrary would break the symmetries by changing the observational behavior of particular implementations of a module.

As a consequence, the conjecture we previously alluded to (a form of generalized Aldous-Hoover representation theorem for exchangeable modules) can finally be stated: exchangeable data

type/modules ought to enjoy a representation theorem (exhibiting the class of parameterized Bayesian models modeling the corresponding data). This constitutes our first work package (possible research direction) for the DPhil thesis.

Work Package 1: Solve Staton's conjecture:

Conjecture 2.17 (Staton's representation conjecture)**.** *In a probabilistic programming language including all probability distributions, every data type satisfying the dataflow and abstract type properties is observationally equivalent to one where each operation is sampling from a distribution.*

To prove it, we may need stochastic memoization (section 4), as we will now see on a telling example (section 3).

3 Categorical models of probabilistic generative models

Let us go back to the exchangeable random graph representation example (section 2.2.2), and analyze it semantically from a categorical standpoint. This section draws upon Staton's work [Sta20a]. We all the implementations are given in LazyPPL.

Consider the following a abstract data type interface, allowing us to draw nodes at random (with newNode) from a random graph with a countable set $V =$ v of vertices, and inspect the presence of edges between vertices (with isEdge):

```
class RandomGraph v where
  newNode :: Prob v
  isEdge  :: (v, v) → Bool
```

In the spirit of lemma 2.12, for every graphon $G =$ graphon$\colon [0,1]^2 \to [0,1]$, we can write an implementation for this interface:

```
class Graphon where
  graphon :: (Double, Double) → Double

instance Graphon => RandomGraph Double where
  newNode = uniform
  isEdge = unsafePerformIO $ do
    -- sample a random seed (ie. infinitely branching rose tree):
    newStdGen
    g ← getStdGen
    let rs = randomTree g

    -- return a randomly sampled function '(Double, Double) → Bool' from 'probIsEdge':
    return $ runProb probIsEdge rs
```

```
  where
    probIsEdge :: Prob ((Double, Double) → Bool)
    probIsEdge = memoize $ \(x, y) → bernoulli $ graphon (x, y)
```

assuming we have a memoization function `memoize :: (a → Prob b) → Prob (a → b)` (section 4). The idea is that the graphon specifies the probability of there being an edge between two nodes x, y. Then, once such an edge has been sampled, its presence (or absence) between x and y remain unchanged in the rest of the program (*i.e.* the edge is not resampled again), hence the need to memoize the result.

Staton then goes on to show that such graphon-based implementations are the only ones up to contextual equivalence, provided the module is exchangeable:

Theorem 3.1 (Staton's Aldous-Hoover representation [Sta20a])**.** *An implementation for the interface RandomGraph is exchangeable iff it is observationally equivalent to a graphon implementation.*

The proof involves building a topos model called the 'Rado topos', and paves the way for a very general and elegant way to tackle similar representation problems with the categorical machinery of Caramello's toposic Galois theory [Car13; Car18; CL19].

3.1 Rado topos and toposic Galois theory

The proof of theorem 3.1 relies on constructing a topos (the Rado topos) containing a generic object V (corresponding the vertex set of the Rado graph) to interpret a probabilistic language that allows sampling a node from a graphon and testing the presence of an edge between two nodes, in such a way that the key following observation can be made:

Lemma 3.2. *Graphons $[0,1]^2 \to [0,1]$ are in one-to-one correspondence with internal probability measures $2^V \to \mathbb{R}_{\geq 0}$ on V for which the Fubini theorem holds.*

Let us spell this out in a bit more details. In [Sta+18; Ste21], Staton, Stein, Yang, Ackerman, Freer and Roy construct a nominal-set-like combinatorial model of probability based on covariant presheaves on finite sets **FinSet** $\to$ **Set**, where the representable $I = \mathbf{FinSet}(1, -)$ is a distinguished object thought of as the space of Pólya urns. They then freely generate an affine commutative monad (satisfying the Stone axioms for finite probability [Sto49] and the Beta-Bernoulli conjugacy equations), treating I as such.

In a similar fashion, instead of finite sets as the category of contexts of parameters and their substitutions, the Rado topos is built on the category of finite graphs:

Definition 3.3 (Rado topos)**.** The Rado topos is defined as the topos

$$\mathrm{Sh}(\mathrm{FinGrph}_{emb}^{\mathrm{op}}, \mathrm{J}_{at}) \hookrightarrow\!\!\!\!\rightarrow [\mathrm{FinGrph}_{emb}, \mathbf{Set}]$$

of (covariant) sheaves for the atomic topology on the category $\mathrm{FinGrph}_{emb}$ of finite graphs and embeddings (in the model-theoretic sense, as we will see later), that is, injections that do not add or remove edges.

So for each type $\mathscr{X}$ (interpreted as a sheaf $[\![\mathscr{X}]\!]$) and finite graph g, we have a set $[\![\mathscr{X}]\!](g)$, which is thought of as the set of programs that sample nodes from a subgraph of g.

Example 3.4.

- *The interpretation of* ***Bool*** *is the constant functor* $[\![\textbf{\textit{Bool}}]\!] := 2$ *(since the return value has to be a Boolean):* $\forall g,\ [\![\textbf{\textit{Bool}}]\!](g) = 2$.
- *The representable* $V := \mathrm{FinGrph}_{emb}(1, -)$ *is seen as the object of vertices:* $\forall g,\ V(g) = |g|$.
- *There is a natural transformation* $\mathrm{isEdge}\colon V \times V \to 2$ *testing the presence of an edge between two vertices:* $\forall g,\ \mathrm{isEdge}_g\colon |g| \times |g| \to 2$.

The generic object V is intuitively thought of as the vertex set of the *Rado graph*, also known as the *Erdős-Rényi graph*, or, more simply, as *the (countable) random graph* (because it is universal in some sense that will be made precise).

Definition 3.5 (Rado/Erdős-Rényi graph [Ack37; Cam13; ER59; Rad64])**.** The Rado graph $R = (V_R, E_R)$ is the almost surely unique graph that can be obtained in any one of these equivalent ways:

(i) **Universality**: It is the countable graph that embeds every at most countable graph.

(ii) $V_R \cong \mathbb{N}$, and $E_R(n, m)$ is set to be true iff the n-th binary digit of m is 1, or vice versa.

(iii) $V_R \cong \{p \in \mathbb{N} \mid p \text{ is prime and } p \equiv 1 \mod 4\}$, and $E_R(p, q)$ is set to be true iff p is a quadratic residue mod q, or vice versa.

(iv) **Erdős-Rényi construction**: Let V_R be a countable set. For every $v_1, v_2 \in V_R$, independently flip a biased coin (for a bias in $(0, 1)$), and $E_R(v_1, v_2)$ is set to be true iff the result is heads.

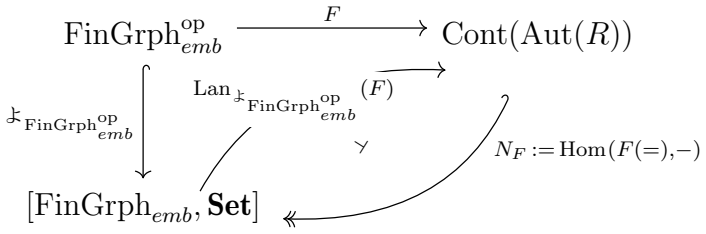
As it happens, the Rado topos is equivalent to the category of continuous actions of the automorphism group of the Rado graph R (which is an instance of a much more general toposic Galois phenomenon (section 3.1.1)).

$$\mathrm{Sh}(\mathrm{FinGrph}_{emb}^{\mathrm{op}}, \mathrm{J}_{at}) \simeq \mathrm{Cont}(\mathrm{Aut}(R)) \tag{5}$$

Indeed, by left Kan extending the functor

$$F\colon \mathrm{FinGrph}_{emb}^{\mathrm{op}} \to \mathrm{Cont}(\mathrm{Aut}(R)),\ g \mapsto \mathrm{Aut}(R/g) := \{\sigma \in \mathrm{Aut}(R) \mid \sigma|_g = \mathrm{id}_g\}$$

along the covariant Yoneda embedding, we have[2] the following "nerve realization paradigm" [Lan78; Lor20]:



where the nerve N_F satisfies, for every $X \in \mathrm{Cont}(\mathrm{Aut}(R))$:

$$N_F(X)(g) \cong \{x \in X \mid \underbrace{x \text{ is supported by } g}_{\text{i.e. } x \text{ is fixed by all } \sigma \in \mathrm{Aut}(R/g)} \}$$

Furthermore, N_F is fully faithful (which is equally to say that $\mathrm{Lan}_F(F) \cong \mathrm{id}_{\mathrm{Cont}(\mathrm{Aut}(R))}$ (*i.e.* F is dense)), so $\mathrm{Cont}(\mathrm{Aut}(R))$ is a reflective subcategory of the presheaf category $[\mathrm{FinGrph}_{emb}, \mathbf{Set}]$, hence a sheaf category [nLa], and we can show that $\mathrm{Cont}(\mathrm{Aut}(R)) \simeq \mathrm{Sh}(\mathrm{FinGrph}_{emb}^{\mathrm{op}}, \mathrm{J}_{at})$.

In this Grothendieck topos $\mathrm{Cont}(\mathrm{Aut}(R))$, objects are sets X (considered as discrete spaces) equipped with a continuous action $\alpha_X \colon \mathrm{Aut}(R) \times X \to X$ of the automorphism group $\mathrm{Aut}(R)$ with the product topology, and morphisms are equivariant functions preserving the actions. The axiom of choice does not hold, but the topos is nonetheless Boolean, and the internal powerobject 2^X is the set of definable subsets (in the model theoretic sense, for the theory of graphs) of X. This convenient description of objects as sets with structure and powerobjects as definable subsets enables us to work more easily in the internal language of the Rado topos, and define the notion of internal probability measures on the set $V := V_R$ of vertices of R:

Definition 3.6 (Internal probability measure)**.** An internal probability measure on $V \in \mathrm{Cont}(\mathrm{Aut}(R))$ (for the powerobject σ-algebra) is a countably additive equivariant map from 2^V to $\mathbb{R}_{\geq 0}$. In other words, it is a morphism $p \colon 2^V \to R_{\geq 0}$ such that $p(V) = 1$ and $p(\biguplus_{i=1}^{\infty} U_i) = \sum_{i=1}^{\infty} p(U_i)$, for every internal countable disjoint sequence $(U_i)_{i \in \mathbb{N}}$ of subsets of V. The sequence being internal countable means that there exists a finite subset $C \subseteq V$ of parameters such that every U_i is C-definable.

Every graphon $G \colon [0,1]^2 \to [0,1]$ gives rise to an internal probability measure $p_G \colon 2^V \to \mathbb{R}_{\geq 0}$ defined, on every C-definable subset $U \subseteq V$, where $C := \{v_1, \ldots, v_n\} \subseteq V$, as:

[2] since $\mathrm{FinGrph}_{emb}^{\mathrm{op}}$ is small and $\mathrm{Cont}(\mathrm{Aut}(R))$ locally small and cocomplete

$$p_G(U) := \frac{\displaystyle\sum_{v_{n+1}\in U} \int_{[0,1]^{n+1}} \prod_{\substack{i,j\in[\![1,n+1]\!]\\ i\neq j}} G(r_i,r_j)^{E_R(v_i,v_j)}(1-G(r_i,r_j))^{1-E_R(v_i,v_j)}\,\mathrm{d}(r_1,\ldots,r_{n+1})}{\displaystyle\int_{[0,1]^n} \prod_{\substack{i,j\in[\![1,n]\!]\\ i\neq j}} G(r_i,r_j)^{E_R(v_i,v_j)}(1-G(r_i,r_j))^{1-E_R(v_i,v_j)}\,\mathrm{d}(r_1,\ldots,r_n)}$$

that is, the probability that the next vertex generated in the Rado graph will be in U. However, not all internal probability measures arise from a graphon in such a way. Staton shows that the ones that do are exactly those that are Fubini:

Definition 3.7 (Fubini measure)**.** An internal measure $\mu\colon 2^V \to \mathbb{R}_{\geq 0}$ is said to be Fubini iff for every equivariant function $f\colon X^2 \to \mathbb{R}_{\geq 0}$,

$$\int_X \int_X f(x_1,x_2)\mu(\mathrm{d}x_1)\mu(\mathrm{d}x_2) = \int_X \int_X f(x_1,x_2)\mu(\mathrm{d}x_2)\mu(\mathrm{d}x_1)$$

As a result, the representation given by lemma 3.2 can be established.

3.1.1 Toposic Galois approach

A very natural question to ask now, when it comes to the previous approach, is: what was adhoc/specific to the theory of graphs, and what, on the contrary, can be abstracted away and generalized to other module signatures? The Rado graph has many remarkable properties (definition 3.5), but what makes most of these assertions true and implies the convenient equivalence of categories eq. (5) is the following extension property:

Lemma 3.8 (Extension property)**.** *If g and g' are finite graphs such that $g \subseteq g'$ and $|g'| = |g| + 1$, then every embedding $g \hookrightarrow R$ can be extended to an embedding $g' \hookrightarrow R$. Equivalently, for every finite disjoint subsets $C, D \subseteq V_R$, there exists a vertex v joined to all $c \in C$ and to no $d \in D$.*

Proof sketch. Using the natural numbers construction (item (iii)), we can see that it is the case by setting, for example, $v := 2^{\max(C\cup D)+1} + \sum_{c\in C} 2^c$. □

As a direct consequence, the Rado graph is, in model-theoretic terms, a homogeneous first-order structure, for the relational theory of graphs (containing a single symmetric and irreflexive binary relation E):

Definition 3.9 (Homogeneous first-order structure)**.** An first-order structure M is *homogeneous* if every isomorphism between finitely generated substructures extends to an automorphism of M.

And the existence of such homogeneous first-order structures was studied in a general model-theoretic setting by Fraïssé [Fra54] in the 50's, who showed that they emerge as (co)limits of well-behaved classes of substructures. More precisely, let L denote a first-order countable language. Fraïssé defines the age of M, $\mathrm{Age}(M)$, as the class of structures isomorphic to some finitely generated substructure of M. He then characterizes classs of countable L-structures that are of the form $\mathrm{Age}(M)$, for a countable homogeneous structure M:

Theorem 3.10 (Fraïssé's theorem)**.** *If a non-empty class $\mathscr{C}$ of finitely generated L-structures satisfies the following conditions:*

- *it is an* amalgamation class*, that is:*
  - *$\mathscr{C}$ is closed under isomorphisms*
  - ***Hereditary Property (HP)**: $\mathscr{C}$ is closed under finitely generated substructures*
  - ***Joint Embedding Property, (JEP)**: if $C_1, C_2 \in \mathscr{C}$, there exists $D \in \mathscr{C}$ such that C_1, C_2 embed in D*
  - ***Amalgamation Property (AP)**: if $C_0, C_1, C_2 \in \mathscr{C}$ such that C_0 embed in C_1, C_2 via $f_i \colon C_0 \to C_i$ (for $i = 1, 2$), there exists $D \in \mathscr{C}$ and embeddings $g_i \colon C_i \to D$ (for $i = 1, 2$) such that $g_1 f_1 = g_2 f_2$*
- *$\mathscr{C}$ contains only countably many non-isomorphic structures*

then there exists a unique (up to isomorphism) countable homogeneous L-structure M, called the Fraïssé limit *of $\mathscr{C}$, such that $\mathscr{C} = \mathrm{Age}(M)$.*

The Rado graph is the Fraïssé limit of the amalgamation class of finite graphs and embeddings. This suggests a recipe to mimic the approach followed in section 3.1 for more general module interfaces:

- Start with the signature of a countable first-order language L.
- Consider the category $\mathbb{C}$ of finitely generated L-structures and embeddings.
- If $\mathscr{C} := \mathrm{ob}\,\mathbb{C}$ is a suitable amalagmation class with Fraïssé limit M, when do we have

$$\mathrm{Sh}(\mathbb{C}^{\mathrm{op}}, \mathrm{J}_{at}) \simeq \mathrm{Cont}(\mathrm{Aut}(M)) \quad ? \tag{6}$$

This is an instance of Caramello's toposic Galois theory [Car13; Car18; CL19], who gives, in full generality, a categorical account of when we can expect a Grothendieck site to induce such an equivalence.

Remark 3.11. As the name suggests, it is a (vast) generalization of traditional Galois theory. A Galois extension L of a field $F \subseteq L$ is a field extension of F which is algebraic over F, and such that F is exactly the intermediate extension $F \subseteq K \subseteq L$ fixed by all $\sigma \in \mathrm{Aut}(L/F)$ (the Galois group of the extension $F \subseteq L$, that is, the group of automorphisms of L that fix F). The Galois correspondence then states that finite intermediate extensions $F \subseteq K \subseteq L$ are in one-to-one correspondence with open subgroups $U \subseteq \mathrm{Aut}(L/F)$. Each open subgroup gives rise to a transitive continuous action of the Galois group, so that we have the following equivalence of categories: $\mathscr{L}_F^{L^{\mathrm{op}}} \simeq \mathrm{Cont}_t(\mathrm{Aut}(L/F))$, where $\mathscr{L}_F^L$ is the category of finite intermediate extensions and $\mathrm{Cont}_t(\mathrm{Aut}(L/F))$ is the category of transitive continuous actions. By taking sheaves for the atomic topology on both sites, we then get $\mathrm{Sh}(\mathscr{L}_F^{L^{\mathrm{op}}}, \mathrm{J}_{at}) \simeq \mathrm{Cont}(\mathrm{Aut}(L/F))$.

The question then remains as to whether (and to what extent) we can obtain representation theorems such as theorem 3.1, by working in the internal language of the topos of continuous actions. This brings us to Work Package 2:

> **Work Package 2**: Push further the toposic semantics approach and see if we can recover a general class of probabilistic representation theorems such as theorem 3.1 (ties in with Work Package 1, but from a more categorical/model-theoretic angle).

3.2 Bipartite random graph topos

We now describe our own work in progress. Drawing on the previous considerations, we turn to the burning question of giving a semantics to stochastic memoization, which played a key role in the graphon-based implementation involved in theorem 3.1. For Boolean-valued functions (to start small), the idea is that we could now wish to have the following interface:

```
-- Atoms (randomly generated fresh names)
new_atom :: 𝔸

-- Function labels : type to be thought of as 𝔸 → Bool
new_function :: 𝔽

-- Application operator making every function memoized: type of a bipartite graph
(@) :: (𝔽, 𝔸) → Bool
```

where every function from a countable set of atoms[3] $\mathbb{A}$ (each one of which can be randomly generated) to **Bool** would be defunctionalized and viewed as an inhabitant of a type $\mathbb{F}$ (which would then be thought of as $\mathbb{A} \to$ **Bool**). Applying a function to an argument and memoizing the result would then be made possible by an 'apply' operator $(@) :: \mathbb{F} \times \mathbb{A} \to$ **Bool**.

[3] in the nominal set tradition [Pit13]

But requiring that the results be memoized is precisely saying that $(@) :: \mathbb{F} \times \mathbb{A} \to \textbf{Bool}$ ought to be seen as the 'edge' relation of a bipartite graph (also called *bigraph*, for short) – with set $\mathbb{F}$ of left nodes and $\mathbb{A}$ of right nodes – whose edges are such that their presence (or absence) remain unchanged after being sampled, exactly like isEdge in the RandomGraph module (at the beginning of section 3).

Analogously to the Rado topos setting, this suggests, on the denotational semantics side, that we are looking for a topos where a random countable bigraph would play the role of the Rado graph in the Rado topos, giving rise to a toposic Galois situation like eq. (6).

The setting is not as ideal as for the Rado graph, because there is no such thing as an almost surely unique countable homogeneous bigraph. But it is not too bad either: there are only five kinds of countable homogeneous bigraphs [GGK94]: the complete bipartite graphs, the empty bipartite graph, the perfect matchings and their complements, and, finally, the countable random bipartite graph (obtained by taking two disjoint countable sets and selecting edges between them at random[4]), which is a Fraïssé limit.

So we are in the advantageous position where we may benefit from the toposic Galois toolbox to study the topos $\mathrm{Sh}(\mathbf{BiGrph}_{emb}{}^{\mathrm{op}}, \mathrm{J}_{at})$ of covariant sheaves on the category $\mathbf{BiGrph}_{emb}$ of finite bigraphs and embeddings, in an attempt to give a semantics to stochastic memoization.

4 Stochastic Memoization

Stochastic memoization is, on the practical side, a very convenient way to implement infinite sequences of random variables (section 1, section 2), point processes, clustering [Sta21a] and, more generally, nonparametric Bayesian models in probabilistic programming. Consider, for example, the following Poisson point process implementation in LazyPPL:

```
poissonPP :: Double → Double → Prob [Double]
poissonPP lower rate = do
  -- sample a sequence ' intervals :: Int → Double'
  -- of successive exponential gaps between points on the real line :
  intervals ← memoize $ \(_ :: Int) → exponential rate

  -- return the corresponding list of points (with exponential interoccurence times),
  -- starting from the 'lower' point :
  return $ scanl (+) lower $ map intervals [1..]
```

The memoization function memoize :: (**Int** → Prob **Double**) → Prob (**Int** → **Double**) enables us to elegantly write this infinite point process in a short and purely decalartive way, with no arbitrary truncation on the size of the returned list of points (even if, as we will see later

[4] a slight modification of the Erdős-Rényi proof shows its existence and almost sure uniqueness.

(section 5), in this specific example where the domain **Int** of the memoized function is discrete, the laziness property of the Haskell language would be sufficient).

On the theoretical side, stochastic memoization may become of utmost importance to obtain general representability results for exchangeable data types, such as theorem 3.1.

In the deterministic setting, memoization is known to be nothing but an innocuous speed-up that do not change the semantics. However, in the presence of probabilistic side-effects, memoizing a stochastic function f is no longer just an optimization technique. It does change the semantics [Roy+08; Sta21a; Woo+09], enabling us to define (possibly infinite) random sequences, which are of paramount importance in probability and statistics.

Categorically, stochastic functions are interpreted as probabilistic kernels $f\colon X \to PY$ in a suitable cartesian closed category[5] – *e.g.* the category of quasi-Borel spaces (definition 2.8) – where P is a probability monad. Stochastic memoization (simply referred to as 'memoization' henceforth) is then internally expressed as a morphism

$$\mathrm{mem}_{X,Y}\colon (PY)^X \longrightarrow P(Y^X)$$

converting a probabilistic kernel (which associates, for every given input in X, a random output in Y) into a random function $X \to Y$ (randomly choosing all the outputs for all the possible inputs at once). If f is written as a lambda-abstraction $\lambda x.\, u\colon X \to PY$, $\mathrm{mem}_{X,Y}(f)$ will be denoted by[6] $\lambda_{\text{מ}} x.\, u$.

For finite types X, memoization is straightforward (it is a matter of simply sampling a value of $f(x)$ for all inhabitants of $x \in X$, and returning the assignment as a finite mapping). In practice, even for **Int** in Haskell, we can exploit laziness (see section 5). But what about an uncountable type X? (We cannot just range over an uncountable space for all x anymore.)

The category of quasi-Borel spaces is unfortunately known not to support memoization in general [Sta21b]: Staton came up with a proof of this fact using graphons, but there is also a more elementary proof by Kallianpur [Kal13, example 1.2.5, p.10] using Fubini's theorem (special thanks to Dario Stein for pointing this out).

The first implementation of 'memoize' that comes to mind uses state to store previously seen values. But usually, using memory compromises the dataflow property (the state monad is not commutative). However, we conjecture that 'memoize' is a special kind of stateful operation that do preserve the dataflow property:

Conjecture 4.1. *Stochastic memoization still admits the dataflow property (even if one resorts to hidden state).*

[5] cartesian closedness enables us to model higher-order functions

[6] borrowing Melliès' use of the Hebrew letter "mem" [Mel14], to mean "memoization" here

One way to prove such a conjecture is to exhibit a denotational model, which we attempt in this section (drawing on our term paper work, where more details can be found), in a restricted setting. We consider a small simply typed language to shed light on three features that we model semantically:

- **name generation**: we can generate fresh names (referred to as *atomic* names or *atoms*, in the sense of Pitt's nominal set theory [Pit13]) with constructs such as $\mathrm{let}\, x \;=\; \mathsf{fresh}()\, \mathrm{in}\, \cdots$.
- basic **probabilistic effects**: for illustrative purposes, the only distribution we consider, as a first step, is the Bernoulli distribution with bias $p = 1/2$. Constructs like $\mathrm{let}\, b \;=\; \mathsf{flip}()\, \mathrm{in}\, \cdots$ amount to flipping a fair coin and storing its result in a variable b.
- **stochastic memoization**: if a stochastic function f – memoized with the new $\lambda_{\text{מ}}$ operator – is called twice on the same argument, it should return the same result.

We have the following base types: bool (booleans), $\mathbb{A}$ (atomic names), and $\mathbb{F}$ (intended for memoized functions $\mathbb{A} \to \mathsf{bool}$). For the sake of simplicity, we do not have arbitrary function types.

$$A, B \;::=\; \mathsf{bool} \mid \mathbb{A} \mid \mathbb{F} \mid A \times B$$

In fine-grained call-by-value fashion [Lev06], there are two kinds of judgments: typed values, and typed computations.

Values:

$$\frac{-}{\Gamma, x : A \vdash^{\mathrm{v}} x : A} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v : A \qquad \Gamma \vdash^{\mathrm{v}} w : B}{\Gamma \vdash^{\mathrm{v}} (v, w) : A \times B} \qquad \frac{-}{\Gamma \vdash^{\mathrm{v}} \mathsf{true} : \mathsf{bool}} \qquad \frac{-}{\Gamma \vdash^{\mathrm{v}} \mathsf{false} : \mathsf{bool}}$$

Computations:

$$\frac{\Gamma \vdash^{\mathrm{v}} v : A}{\Gamma \vdash^{\mathrm{c}} \mathsf{return}(v) : A} \qquad \frac{\Gamma \vdash^{\mathrm{c}} u : A \qquad \Gamma, x : A \vdash^{\mathrm{c}} t : B}{\Gamma \vdash^{\mathrm{c}} \mathsf{let\ val}\ x \;\leftarrow\; u\, \mathsf{in}\, t : B}$$

Matching:

$$\frac{\Gamma \vdash^{\mathrm{v}} v : \mathsf{bool} \qquad \Gamma \vdash^{\mathrm{c}} u : A \qquad \Gamma \vdash^{\mathrm{c}} t : A}{\Gamma \vdash^{\mathrm{c}} \mathsf{if}\, v\, \mathsf{then}\, u\, \mathsf{else}\, t : A} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v : A \times B \qquad \Gamma, x : A, y : B \vdash^{\mathrm{c}} t : C}{\Gamma \vdash^{\mathrm{c}} \mathsf{match}\, v\, \mathsf{as}\, (x, y)\, \mathsf{in}\, t : C}$$

Language-specific commands:

$$\frac{-}{\Gamma \vdash^{\mathrm{c}} \mathsf{flip}() : \mathsf{bool}} \qquad \frac{-}{\Gamma \vdash^{\mathrm{c}} \mathsf{fresh}() : \mathbb{A}} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v : \mathbb{A} \qquad \Gamma \vdash^{\mathrm{v}} w : \mathbb{A}}{\Gamma \vdash^{\mathrm{c}} (v = w) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash^{\curlyvee} v : \mathbb{F} \qquad \Gamma \vdash^{\curlyvee} w : \mathbb{A}}{\Gamma \vdash^{\curlywedge} (v@w) : \mathsf{bool}} \qquad \frac{\Gamma, x : \mathbb{A} \vdash^{\curlywedge} u : \mathsf{bool}}{\Gamma \vdash^{\curlywedge} \lambda_{\text{m}} x.\, u : \mathbb{F}}$$

We work in the (cartesian closed) category of covariant presheaves on the category

$$\mathbf{BiGrph}_{emb} \;:=\; (\mathbf{Grph}_{emb} \hookrightarrow \mathbf{Grph}) \downarrow (\mathbf{Grph} \xleftarrow{\Delta_{K_2}} *)$$

of finite bipartite graphs (henceforth called *bigraphs*) and embeddings (that do not add or remove edges).

Notation 4.2. For a bigraph g, we denote by g_L (resp. g_R) and E^g its set of left (resp. right) nodes and its edge relation.

The denotation of basic types is given by:

$$[\![\mathbb{F}]\!] = \mathbf{BiGrph}_{emb}(\circ, -) \qquad [\![\mathbb{A}]\!] = \mathbf{BiGrph}_{emb}(\bullet, -)$$

where $\circ$ and $\bullet$ are the one-vertex left and right graphs respectively. The denotation of the type of booleans is the constant presheaf $2 \cong 1 + 1$, as usual.

For a bigraph g and a presheaf $X = [\![\mathcal{X}]\!]$, $X(g)$ is thought of as the set of generative models/programs of type $\mathcal{X}$ that may use the bigraph g, in the following sense: probabilistic function (that we want to memoize) and atom labels are stored as left and right nodes respectively. The presence (resp. absence) of an edge between a given left and right node memoizes the fact that a probabilistic call of the corresponding function on the corresponding atom has resulted in true (resp. false).

For every embedding $\iota \colon g \hookrightarrow g'$, the function $X\iota \colon X(g) \to X(g')$ models substitution in the programs in $X(g)$ according to ι.

Remark 4.3. The model will soon be refined to the subtopos of sheaves over $\mathbf{BiGrph}_{emb}$ for the atomic Grothendieck topology, as discussed in section 3.2. The sheaf condition expresses the fact that if two bigraphs g_1, g_2 have a common intersection g and $x \in X(g_1) \cap X(g_2)$, then the program x can be regarded as only using g, so that $x \in X(g)$ [Sta06].

4.1 Probabilistic local state monad

In the following, $X, Y, Z \colon \mathbf{BiGrph}_{emb} \to \mathbf{Set}$ denote presheaves, $g = (g_L, g_R, E^g), g', h, h' \in \mathbf{BiGrph}_{emb}$ bigraphs, and $\iota, \iota' \colon g \hookrightarrow g'$ bigraph embeddings. We will omit subscripts when they are clear from the context.

Inspired from Plotkin and Power's local state monad [PP02] (which was defined on the covariant presheaf category $[\mathbf{Inj}, \mathbf{Set}]$, where $\mathbf{Inj}$ is the category of finite sets and injections), we model

probabilistic and name generation effects by the following monad, that we name 'probabilistic local state monad':

Definition 4.4 (Probabilistic local state monad)**.** For all covariant presheaf $X : \mathbf{BiGrph}_{emb} \to \mathbf{Set}$ and bigraph $g \in \mathbf{BiGrph}_{emb}$:

$$T(X)(g) \;:=\; \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} \left(X(h) \times [0,1]^{(h-g)_L}\right)\right)^{[0,1]^{g_L}}$$

where P_{f} is the finite distribution monad.

The monad T is similar to the read-only local state monad, except that any fresh node can be initialized. Every $\lambda \in [0,1]^{g_L}$ is thought of as the probability of the corresponding function/left node being true on a new fresh atom. We will refer to such a λ as a *state of biases*. The coend takes care of garbage collection.

Notation 4.5. Equivalence classes in $\int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L}$ are written $[x_h, \lambda^h]_g$. We use Dirac's bra-ket notation $\left|[x_h, \lambda^h]_g\right\rangle_h$ to denote a formal column vector of equivalence classes ranging over a finite set of h's. As such, a formal convex sum $\sum_i p_i [x_{h_i}, \lambda^{h_i}]_g \in P_{\mathrm{f}} \int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L}$ will be concisely denoted by $\left\langle \overrightarrow{p} \;\middle|\; [x_h, \lambda^h]_g \right\rangle_h$.

Definition 4.6 (Action of $T(X)$ on morphisms)**.**

$$T(X)(g \overset{\iota}{\hookrightarrow} g') : \begin{cases} \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L}\right)^{[0,1]^{g_L}} \longrightarrow \left(P_{\mathrm{f}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L}\right)^{[0,1]^{g'_L}} \\ \vartheta \mapsto [0,1]^{g'_L} \xrightarrow{-\circ \iota_L} [0,1]^{g_L} \xrightarrow{\vartheta} P_{\mathrm{f}} \int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L} \\ \qquad\qquad \xrightarrow{P_{\mathrm{f}} \psi_{g,g'}} P_{\mathrm{f}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L} \end{cases}$$

where

- the map $\int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L} \xrightarrow{\psi_{g,g'}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L}$ is given by:

$$\frac{\begin{cases} X(h) \times [0,1]^{(h-g)_L} \to X(h \coprod_g g') \times [0,1]^{(h \coprod_g g' - g')_L} \to \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L} \\ (x_h,\, \lambda^h) \longmapsto (X(h \hookrightarrow h \coprod_g g')(x_h),\, \lambda^h) \longmapsto [X(h \hookrightarrow h \coprod_g g')(x_h),\, \lambda^h]_{g'} \end{cases}}{\int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L} \xrightarrow{\psi_{g,g'}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L}} \text{ extranat. in } h$$

- $\iota_L : g_L \hookrightarrow g'_L$ is the embedding restricted to left nodes.

- $h \coprod_g g'$ is the pushout in the category of graphs regarded as an object of $\mathbf{BiGrph}_{emb}$:

$$\begin{array}{ccc} g & \hookrightarrow & g' \\ \downarrow & \ulcorner & \downarrow \\ h & \hookrightarrow & h \coprod_g g' \end{array}$$

Notation 4.7. More concretely, with Dirac's convenient bra-ket notation, $T(X)(g \overset{\iota}{\hookrightarrow} g')$ can be written as:

$$T(X)(\iota) = \begin{cases} \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} X(h) \times [0,1]^{(h-g)_L}\right)^{[0,1]^{g_L}} \longrightarrow \left(P_{\mathrm{f}} \int^{g' \hookrightarrow h'} X(h') \times [0,1]^{(h'-g')_L}\right)^{[0,1]^{g'_L}} \\ \vartheta \longmapsto \lambda' \mapsto \text{let } \vartheta(\lambda' \iota_L) \;=\; \left\langle \overrightarrow{p} \;\middle|\; [x_h, \lambda^h]_g \right\rangle_h \text{ in } \left\langle \overrightarrow{p} \;\middle|\; [X(h \hookrightarrow h \coprod_g g')(x_h), \lambda^h]_{g'} \right\rangle_h \end{cases}$$

As argued in section 2.2.3, to construct an abstract model of probability, we show that the monad is commutative and affine:

Theorem 4.8. *The probabilistic local state monad T is strong commutative and affine.*

4.2 Categorical semantics

In our language, the denotational interpretation of values, computations (return and let binding), and matching (elimination of bool's and product types) is standard.

We interpret computation judgements $\Gamma \vdash^{\subset} t : A$ as morphisms $[\![\Gamma]\!] \to T([\![A]\!])$, by induction on the structure of typing derivations. The context Γ is built of bool's, $\mathbb{A}$ and $\mathbb{F}$ and products. Therefore, $[\![\Gamma]\!]$ is isomorphic to an object of the form $2^k \times \mathbf{BiGrph}_{emb}(\circ, -)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, -)^m$.

Definition 4.9. For every bigraph g, we denote by R_g (resp. L_g) the set of bigraphs $h \in g/\mathbf{BiGrph}_{emb}$ having one more right (resp. left) node than g, and that are the same otherwise.

$$\begin{aligned} R_g &:= \{\, h \in \mathbf{BiGrph}_{emb} \mid h_L = g_L,\ g_R \subseteq h_R \text{ and } |h_R| = |g_R| + 1 \,\} \\ L_g &:= \{\, h \in \mathbf{BiGrph}_{emb} \mid h_R = g_R,\ g_L \subseteq h_L \text{ and } |h_L| = |g_L| + 1 \,\} \end{aligned}$$

Denotation of $\Gamma \vdash^{\subset} \mathsf{fresh}() : \mathbb{A}$

The map $[\![\mathsf{fresh}]\!]_g : [\![\Gamma]\!](g) \to T([\![\mathbb{A}]\!])(g)$ randomly chooses connections to each left node according to the state of biases, and makes a fresh right node with those connections.

$$[\![\mathsf{fresh}]\!]_g : \begin{cases} 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \longrightarrow P_{\mathrm{f}}(g_R + 2^{g_L})^{[0,1]^{g_L}} \\ \_, \_, \_ \mapsto \lambda \mapsto \left\langle \dfrac{1}{Z} \displaystyle\prod_{\mathsf{f} \in g_L} \lambda(\mathsf{f})^{E^h(\mathsf{f}, a_h(\bullet))} (1 - \lambda(\mathsf{f}))^{1 - E^h(\mathsf{f}, a_h(\bullet))} \;\middle|\; [\underbrace{\bullet}_{\cong (h-g)_R} \overset{a_h}{\hookrightarrow} h,\ !]_g \right\rangle_{h \in R_g} \end{cases}$$

where Z is a normalization constant.

Remark 4.10. It is enough to consider these h's only, by garbage collection of the coend.

Denotation of $\Gamma \vdash \lambda_{\text{ם}} x.\, u : \mathbb{F}$

As $\lambda_{\text{ם}}$-abstractions are formed based on computation judgements of the form $\Gamma, x : \mathbb{A} \vdash u : \mathsf{bool}$, we first note that

$$T(\llbracket \mathsf{bool} \rrbracket) g \;\cong\; P_{\mathrm{f}}(2)^{[0,1]^{g_L}} \;\cong\; [0,1]^{[0,1]^{g_L}}$$

Also, we can decompose the extra variable x in the environment $\Gamma,\ x : \mathbb{A}$, the denotation of which is of the form $\llbracket \Gamma,\ x : \mathbb{A} \rrbracket(g) \;=\; 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \;\times\; \mathbf{BiGrph}_{emb}(\bullet,\ g)$ for a bigraph $g \in \mathbf{BiGrph}_{emb}$.

Now, the extra part x is a right node, and its valuation will either be a node already in the graph described in the rest of the environment, or a new one with particular edges to the rest of the environment. The argument u can test (if it wants) what kind of node x is, before returning a probability.

As a result, if

$$\llbracket u \rrbracket_g \colon 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \times \mathbf{BiGrph}_{emb}(\bullet, g) \longrightarrow [0,1]^{[0,1]^{g_L}}$$

the denotation $\llbracket u \rrbracket$ gives us the edge probability of the left node that we need to generate, both to the existing right nodes, and to any future right nodes (which needs to be remembered). This can be formalized into a natural transformation $\llbracket \lambda_{\text{ם}} x.\, u \rrbracket \colon \llbracket \Gamma \rrbracket \to T(\llbracket \mathbb{F} \rrbracket)$.

$$\llbracket \lambda_{\text{ם}} x.u \rrbracket_g \colon \begin{cases} 2^k \times \mathbf{BiGrph}_{emb}(\circ, g)^\ell \times \mathbf{BiGrph}_{emb}(\bullet, g)^m \longrightarrow P_{\mathrm{f}}(g_L + 2^{g_R} \times [0,1])^{[0,1]^{g_L}} \\ b^k,\ \begin{matrix} (\circ \overset{\kappa_i}{\hookrightarrow} g)_i, \\ (\bullet \overset{\tau_j}{\hookrightarrow} g)_j \end{matrix} \mapsto \lambda \mapsto \left\langle \dfrac{1}{Z} \displaystyle\prod_{a \in g_R} p_a^{E^h(\int_h(\circ), a)} (1 - p_a)^{1 - E^h(\int_h(\circ), a)} \;\middle|\; \Big[\underbrace{\circ}_{\cong (h-g)_L} \overset{\int_h}{\hookrightarrow} h,\ \_ \mapsto \tilde{p}\Big]_g \right\rangle_{h \in L_g} \end{cases}$$

where

- Z is a normalization constant.
- for every $a \in g_R$, $\quad p_a \;:=\; \llbracket u \rrbracket_g \big(b^k,\ (\circ \overset{\kappa_i}{\hookrightarrow} g)_i,\ (\bullet \overset{\tau_j}{\hookrightarrow} g)_j,\ \bullet \overset{a}{\hookrightarrow} g,\ \lambda\big)$
- $\tilde{p} \;:=\; \llbracket u \rrbracket_g \big(b^k,\ (\circ \overset{\kappa_i}{\hookrightarrow} g \overset{\iota 1}{\hookrightarrow} g + \bullet)_i,\ (\bullet \overset{\tau_j}{\hookrightarrow} g \overset{\iota 1}{\hookrightarrow} g + \bullet)_j,\ \bullet \overset{\iota 2}{\hookrightarrow} g + \bullet,\ \lambda\big)$ where $\iota_1,\ \iota_2$ are the coprojections.

This semantics analysis of stochastic memoization can be extended in many ways, which brings us to the third Work Package:

Work Package 3:

- Prove termination, adequacy, and full abstraction (if it holds) results.
- Give a denotational semantics to unrestricted stochastic memoization, *i.e.* not just for Boolean-valued functions, and extend it to a more expressive type system (*e.g.* with arbitrary function types, a non-memoized lambda abstraction, a $\lambda_{\text{מ}}$ primitive, etc.).
- Is $\mathbb{F}$ giving some sort of closed structure (it likely is not an internal Hom, but do we have a closed Freyd category [PT99; Sta14])?
- Rather than the finite distribution monad in the definition of the probabilistic local state monad T (definition 4.4), can we consider more general probabilistic monads (*e.g.* the Giry monad, QBS monad, or a generic Kock monad (strong commutative affine monad))?

5 Probabilistic programming and applications

We have been using LazyPPL examples all throughout this report. LazyPPL is a probabilistic programming library, implemented in Haskell, that Staton, Paquet, Dash and I are using to experiment with Bayesian nonparametric probabilistic programming concepts, among other things [PS21; Sta+]. Bayesian nonparametrics is a general and elegant setting where the space of parameters of statistical models is unbounded or infinite dimensional.

As mentioned before (section 4), we can sometimes successfully take advantage of the host language's laziness to circumvent the issue of manipulating potentially infinite structures. For example, in LazyPPL, the opening Poisson point process example in section 4 can be rewritten in a stateless way (without resorting to stochastic memoization) as follows:

```
poissonPP :: Double → Double → Prob [Double]
poissonPP lower rate = do
  step ← exponential rate
  let x = lower + step
  xs ← poissonPP x rate
  return (x : xs)
```

Thus, Haskell's laziness enables us to express nonparametric models in a declarative way, without explicitly and arbitrarily truncating the viewport. We now illustrate this advantage on two examples we have personally worked on: the Mondrian process (section 5.1) and the Stanford substitution cipher algorithm (section 5.2).

5.1 Relational modeling and Mondrian process

This will appear in a recently submitted paper with Sam Staton, Hugo Paquet, and Swaraj Dash.

The Mondrian process [RT09] can be seen as a k-dimensional generalization of the Poisson point process. It is a stochastic generative process valued in axis-aligned block partitions of a k-dimensional hypercube $\Theta_1 \times \cdots \times \Theta_k$ (where each Θ_d is a closed interval of the form $[a_d, b_d]$), that can be used as a prior on k-d trees [Ben75]. We will use it to model k-dimensional relations $\Theta_1 \times \cdots \times \Theta_k \to \{\textbf{True}, \textbf{False}\}$.

Due to its convenient self-consistency property, the Mondrian process has been successfully used in machine learning as a form of decision tree to define a new class of random forests (called *Mondrian forests*), achieving competitive performance in online nonparametric classification [LRT15] and large-scale regression [BT15; LRT16]. It has also been used as an applied modeling tool – *e.g.* in computational biology, in conjunction with a Bayesian hierarchical regression model, to recover species interactions from the ecological system they live in [AHS13].

On top of its block domain $\Theta_1, \ldots, \Theta_k$, a Mondrian process can be truncated in time by setting a fixed lifetime (also referred to as *budget*) hyperparameter $\lambda \in \mathbb{R}$. The resulting process, denoted by $M(\lambda, \Theta_1, \ldots, \Theta_k)$, partitions the k-dimensional hypercube as follows. At each recursive step, we sample the next residual time t at which a cut x_d – orthogonal to one of the axes in a dimension $d \in \{1, \ldots, k\}$ to be determined – will occur. Each axis $\Theta_d = [a_d, b_d]$ is then thought of as an independent clock whose residual time until ringing is exponentially distributed with rate its length $b_d - a_d$. In this competing exponential clock setting, the residual time t until one of the clocks rings is exponentially distributed with rate the sum of the lengths $\sum_{d=1}^{k} b_d - a_d$, and the probability that the d-th clock Θ_d be the first one to do so is proportional to $b_d - a_d$. If t exceeds the budget λ, we stop there and return the block $\Theta_1 \times \cdots \times \Theta_k$. Otherwise, we cut the domain orthogonally to the d-th axis at x_d, and recurse with remaining budget $\lambda - t$ on a left (resp. right) subdomain where Θ_d has been restricted to $[a_d, x_d]$ (resp. $[x_d, b_d]$) and the other dimensions are left unchanged.

This yields a k-d tree whose leaves are k-dimensional blocks partitioning the initial domain Θ (the root of the tree) and whose internal nodes are given by a cut point $x_d \in \mathbb{R}$ splitting a subdomain $\Theta' \subseteq \Theta$ along a given axis $d \in \mathbb{N}$. On top of that, we annotate each leaf block with a sample p from a base distribution μ, which will be the hyperparameter of a distribution ν from which we will sample the truth value of the relation we model at this given block. The sampling distribution ν will typically be bernoulli for example: each block is then equipped with a bias p and k elements (a point in the block) will be related with probability p.

In LazyPPL, we can implement it with the following abstract type and generative process (note that the number of blocks is unbounded, so this works for any dimension $k \in \mathbb{N}$):

```
data Mondrian a =
  -- parameters: block bias and intervals making up the block
  Block a [(Double, Double)]
  -- parameters: dimension cutPosition domainIntervals subTree1 subTree2
```

```
  | Partition Int Double [(Double, Double)] (Mondrian a) (Mondrian a)
  deriving (Eq, Show)

-- Generate a random Mondrian tree:
-- a random block partition of the rectangle Θ := [a_1, b_1] × ··· × [a_k, b_k]
-- where each block has an associated random draw from the base distribution μ.
randomMondrian :: Prob a → Double → [(Double, Double)] → Prob (Mondrian a)
randomMondrian μ λ Θ = do
  let lengths = map (\(a, b) → b - a) Θ
  let sumLengths = sum lengths
  t ← exponential sumLengths
  if λ < t
    then do p ← μ; return $ Block p Θ
    else do
      let remaining = λ - t
      dim ← 1 + categorical $ map (/sumLengths) lengths
      -- if dim = d, then the cut x_d is perpendicular to (a_d, b_d)
      let (a_d, b_d) = Θ !! dim
      x_d ← uniformbounded a_d b_d
      leftMondrian ← randomMondrian μ remaining
        $ zipWith (\ab i → if i == dim then (a_d, x_d) else ab) Θ [1 ..]
      rightMondrian ← randomMondrian μ remaining
        $ zipWith (\ab i → if i == dim then (x_d, b_d) else ab) Θ [1 ..]
      return $ Partition dim x_d Θ leftMondrian rightMondrian
```

Remark 5.1. In dimension 1 (when $k = 1$), the Mondrian process is a Poisson point process on the one-dimensional domain $\Theta := [a_1, b_1]$.

5.1.1 Relational modeling in the 2-dimensional case

In the sequel, we assume that $k = 2$. Let us go back to 2D relational data modeling (mentioned at the beginning of section 2.2.2). We have two ways of generating a synthetic relation from a Mondrian tree: we can either sample what we will refer to as

- a 'map relation', of type Map (**Double**, **Double**) **Bool**. That is, we sample a finite number of $(x, y) \in [0, 1]^2$ pairs, and associate to each (x, y) the result of a Bernoulli trial with bias the bias of the block containing (x, y).
- a (potentially infinite) 'matrix relation', of type [[**Bool**]]. That is, we sample two iid sequences $(x_i)_{i \in \mathbb{N}}$ and $(y_j)_{j \in \mathbb{N}}$, and set each coefficient (i, j) of the matrix to be the result of a Bernoulli trial with bias the bias of the block containing (x_i, y_j).

This leads to two sampling functions (the latter taking advantage of laziness to return potentially infinite matrices):

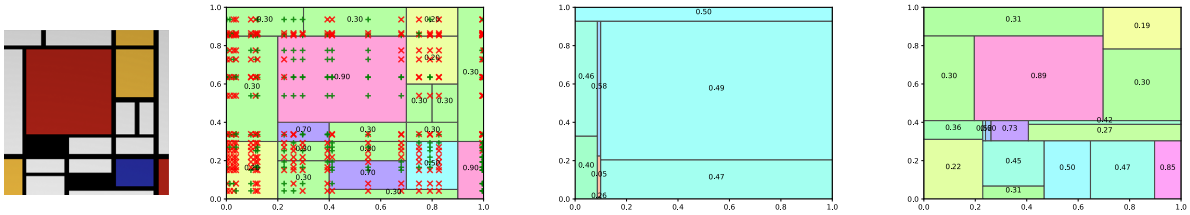


Figure 1: From left to right: **(a)** "Composition with Large Blue Plane, Red, Black, Yellow, and Gray" by Piet Mondrian (1921); **(b)** 2D Mondrian tree mimicking the 1921 Mondrian painting, and an example of a random synthetic 'map relation' generated from it (red cross = `false` at this (x, y) coordinate, green plus = `true`); **(c)(d)** Two examples sampled from the posterior distribution (using LazyPPL's Metropolis-Hastings algorithm), trying to recover (b) from, respectively, observed synthetic 'matrix' (c) and 'map' (d) relations.

```
sampleMapRelationFromMondrian2D :: Mondrian Double → Int → Prob (Map (Double, Double) Bool)

data Matrix = Matrix [[Bool]]
sampleMatrixRelationFromMondrian2D :: Mondrian Double → Prob Matrix
```

We can then come up with two statistical models to infer a Mondrian from a dataset of either 'map' or 'matrix' observed relations. First, we sample a random Mondrian tree with randomMondrian, and then

- in the 'map' case: we score every Boolean observation at a (x, y) pair (in the domain of a 'map relation') with the likelihood of getting this Boolean value at this (x, y) coordinate from our Mondrian tree.
- in the 'matrix' case: for every 'matrix relation', we randomly sample iid sequences $(x_i)_{i\in\mathbb{N}}$ and $(y_j)_{j\in\mathbb{N}}$ and score every Boolean observation in position (i, j) with the likelihood of getting this Boolean value at (x_i, y_j) from our Mondrian tree.

As an example, inspired from [RT09], we try to infer a Mondrian tree stemming from an actual 1921 painting of Piet Mondrian, by observing finite 'matrix' and 'map' relations randomly generated from it (1).

Other examples our interface will support in future versions will be inference (with visualization) of 3D relations, the Mondrian forest algorithm [LRT16; LRT15] for regression and classification, and making use of lazy Mondrians in the Infinite Relational Model [Kem+06].

5.2 Stanford substitution cipher algorithm

We now turn to another application, which ended up highlighting several drawbacks of LazyPPL's general-purpose Metropolis-Hastings-Green inference algorithm: the Stanford substitution

cipher algorithm presented by Diaconis in [Dia08].

As alluded to at the beginning of section 2.2.3, in LazyPPL, the 'infer' construct does not return a normalized posterior, but rather produces an unbounded (by laziness) list of samples from the unnormalized measure defined by the statistical model we perform the inference on, in the form of a procedure

```
mh :: forall a. Double → Meas a → IO [(a, Product (Log Double))]
```

Based on the noise-outsourcing lemma (lemma 2.6), the source of randomness in LazyPPL, for probability and unnormalized measures, does not simply come from the sample space $\Omega := [0,1]$ (to use the QBS notations of definition 2.8, on the semantic side), but from the (Borel isomorphic) sample space $\Omega := [0,1]^{\mathbb{N}^*}$ (where $\mathbb{N}^*$ is the set of finite strings of natural numbers) of random rose trees, *i.e.* random infinitely branching trees whose nodes (called *sites*) are in $[0,1]$. So $\Omega \cong [0,1] \times \Omega^\omega$, which enables us to bijectively split the seed tree into an arbitrary number of random seeds at each step.

Executing `mh p model` (where `p` $\in [0,1]$ and `model` :: `Meas a`) then roughly goes at follows:

1. The model is simulated with a randomly-generated seed tree t, producing a sample x with a likelihood weight w.
2. We repeat the following steps *ad infinitum* to produce a stream of weighted samples, starting on (t, x, w):
   (a) Randomly (lazily) resample every site in t with probability `p`, producing a new seed tree t'.
   (b) Run the model with t', yielding a new sample-weight pair (x', w'), which is the Metropolis-Hastings (MH) proposal.
   (c) Compute the MH acceptance ratio $r = \min(w'/w, 1)$, and accept the proposal with probability r.
   (d) Start again with (t', x', w') if the proposal has been accepted, or with (t, x, w) otherwise.

Keeping the inner workings of this MH algorithm in mind, let us now turn to the Stanford substitution cipher algorithm described by Diaconis in [Dia08], to implement it in LazyPPL.

The goal of this substitution cipher algorithm is to decode a message encoded by an unknown injective mapping of every letter from an input alphabet (typically the Latin/English alphabet) A_{in} to an output alphabet (the coding alphabet) A_{out}. The Stanford algorithm goes as follows:

1. At the beginning, compute a transition matrix $M_t := (p_t(\ell_1, \ell_2))_{\ell_1, \ell_2 \in A_{in}}$, where p_t is the probability of letter ℓ_2 appearing after letter ℓ_1, based on a corpus of the input language

(say English, in the following).[7]

2. Start with a given encrypted message $m := m_1 \cdots m_n$ (where $m_i \in A_{out}$ for all i) and guess at random a substitution cipher, *i.e.* an injective map $\sigma \colon A_{out} \to A_{in}$.

3. Then, iterate the following MH procedure, starting from (m, σ):

   (a) Apply a random transposition to σ (*i.e.* swap the images of two letters), producing a new substitution cipher σ', the MH proposal.

   (b) Compute the MH ratio

   $$r = \min\left(\frac{\text{score}(\sigma', m)}{\text{score}(\sigma, m)}, 1\right) \text{ where } \text{score}(\tau, m) := \prod_{1 \leq i \leq n-1} p_t(\tau(m_i), \tau(m_{i+1}))$$

   and accept the proposal with probability r.

   (c) Repeat with the new proposal σ' if accepted, or with σ otherwise.

Thus, it is based on a simple MH variant, where proposals are substitution ciphers slightly different from the current one (differing by a transposition only). However, implementing it in LazyPPL was not straightforward (we had to tweak the model in several ways) for the following reasons:

- **No custom MH proposals:** in LazyPPL, the state space of proposals is the space of random seeds (rose trees), and changing sites of a rose tree can result in dramatically different substitution ciphers, leading to big leaps in the space of substitutions, which prevents us from incrementally improving the score efficiently. To partially counterbalance this effect, we sample a Poisson number of transpositions in the model, and then apply them to the current substitution cipher.

- **Prior on the initial σ and number of existing words:** Sampling σ from a uniform distribution resulted in a very small initial score (leading to a prohibitively long convergence time of the MH algorithm), so we instead sampled it based on a categorical distribution with parameters the letter frequencies in the corpus (so that the most common coded letter was more likely to be mapped by σ to a 'e' rather than a 'x', for example). We also balanced the score obtained from the transition matrix (the 'transition score') with another score (the 'existing-words score'), proportional to the proportion of existing English words in the guessed decoded message $\sigma(m) := \sigma(m_1) \cdots \sigma(m_n)$, to avoid biasing the decoding process towards the most common syllables only (irrespective of whether they are part of an existing English word or not).

[7] To be more precise, we include the probability of transitioning from a letter to a space or punctuation mark (and vice versa) in M_t, because the words splitting is important (for the sake of simplicity, space and punctation marks be be considered as letters henceforth).

- **Vanishing and not-varying-enough scores:** As it was, the 'transition score' scaled exponentially with the size $|m|$ of the coded message, so that it was either quickly vanishing or taking over the 'existing-words score', depending on their relative weights.[8] To overcome this, we took the $|m|$-th root of the transition score (and to limit numerical approximation errors, we had to directly work with log numbers within the model (rather than letting LazyPPL handle it as usual)). The MH ratios were also too close to one another in practice: the weights of the MH runs were all of the form $c + \varepsilon$, for a constant c such that $\varepsilon \ll c$. So it was too easy to haphazardly jump around in the proposal space, even when the score diminished (because the MH ratio was still high). To mitigate this issue, we introduced hyperparameters to diminish c accordingly (resulting in the MH ratios varying more, and decreases of the proposal scores having more impact).

This led to the interesting question of considering, for a given statistical model, equivalent priors that are better suited to the inference algorithm at hand. Staton has begun experimenting with this, and this is a line of work we may pursue in the future, if we want to make the most of a general-purpose Metropolis-Hastings-Green inference algorithm.

Work Package 4: Further develop LazyPPL, on the implementation and semantic side. Experiment with the idea of 'equivalent prior' and other inference algorithms too.

6 Further directions and Conclusion

Program Synthesis and Machine Learning for automatic model generation. We briefly mention another fascinating research direction we would like to explore: probabilistic programming is all about separating the model specification and the statistical inference by automating the latter; but the model still has to be written by hand, requiring expert domain knowledge and/or adhoc model design. How about automating it as much as possible too? There are a many recent lines of work attempting to do so, leveraging program synthesis and machine learning. To name a few: on the traditional program synthesis front, various SMT solvers and sketching-based approaches are used [EST; Nor+15]. The thriving area of machine learning is not short of innovative ideas either: generative adversarial networks [Goo+14] (GANs), variational auto-encoders [KW14] (VAEs), and more recently, deep learning approaches leading to the new and vibrant field of 'deep probabilistic programming' [BHM18; Bin+18; 20; Tra20; Tra+17] (requiring a good integration of probabilistic computations with automatic differentiation, as these are, for the most, gradient-based methods). More recently, large language models (LLMs) based on transformers [Vas+17] led to breakthroughs in natural language processing and computer

[8] This resulted in certain letters (like f and p) not being swapped albeit being in the wrong position at the very end.

vision (going as far as almost unifying the two fields), and are already considered as viable options for program generation [Aus+21; Che+21; 22; Jai+21; Nij+22].

Work Package 5: Explore automatic statistical model generation (with deep probabilistic programming, program synthesis, or other machine learning generative models (GANs, VAEs, LLMs (transformers), etc.)).

Conclusion. We have proposed various research directions in the form of five work packages, ranging from

- *foundational aspects*: probabilistic representation theorems, toposic Galois analysis of exchangeable data types (to get a better grasp on symmetries in probability, from a synthetic point of view), and semantics of stochastic memoization.
- to *more applied ones*: new LazyPPL feature extensions (make MH more efficient and flexible, experiment with new inference algorithms, etc.) and applications (implement other nonparametric Bayesian models that may shed light on other issues), explore the idea of 'equivalent priors', and leverage program synthesis and machine learning in deep probabilistic programming for automatic model generation.

References

[Ack37] Wilhelm Ackermann. "Die Widerspruchsfreiheit der allgemeinen Mengenlehre". In: *Mathematische Annalen* 114.1 (Dec. 1937), pages 305–315. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01594179. URL: http://link.springer.com/10.1007/BF01594179 (visited on 04/08/2022).

[AHS13] Andrej Aderhold, Dirk Husmeier and V. Anne Smith. "Reconstructing Ecological Networks with Hierarchical Bayesian Regression and Mondrian Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. PMLR, Apr. 29, 2013, pages 75–84. URL: https://proceedings.mlr.press/v31/aderhold13a.html (visited on 04/05/2022).

[Ald81] David J. Aldous. "Representations for Partially Exchangeable Arrays of Random Variables". In: *Journal of Multivariate Analysis* 11.4 (Dec. 1, 1981), pages 581–598. ISSN: 0047-259X. DOI: 10.1016/0047-259x(81)90099-3. URL: http://www.sciencedirect.com/science/article/pii/0047259X81900993 (visited on 11/25/2020).

[Aus+21] Jacob Austin et al. "Program Synthesis with Large Language Models". Aug. 15, 2021. arXiv: 2108.07732 [cs]. URL: http://arxiv.org/abs/2108.07732 (visited on 01/26/2022).

[Aus] Tim Austin. “Exchangeable Random Arrays”.

[BT15] Matej Balog and Yee Whye Teh. “The Mondrian Process for Machine Learning”. July 18, 2015. arXiv: 1507.05181 [cs, stat]. URL: http://arxiv.org/abs/1507.05181 (visited on 10/16/2021).

[BHM18] Guillaume Baudart, Martin Hirzel and Louis Mandel. “Deep Probabilistic Programming Languages: A Qualitative Study”. Apr. 17, 2018. arXiv: 1804.06458 [cs]. URL: http://arxiv.org/abs/1804.06458 (visited on 04/11/2022).

[Ben75] Jon Louis Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Communications of the ACM* 18.9 (Sept. 1, 1975), pages 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: https://doi.org/10.1145/361002.361007 (visited on 03/02/2022).

[Bin+18] Eli Bingham et al. “Pyro: Deep Universal Probabilistic Programming”. Oct. 18, 2018. arXiv: 1810.09538 [cs, stat]. URL: http://arxiv.org/abs/1810.09538 (visited on 04/11/2022).

[Bor+17] Johannes Borgström et al. “A Lambda-Calculus Foundation for Universal Probabilistic Programming”. Jan. 23, 2017. arXiv: 1512.08990 [cs]. URL: http://arxiv.org/abs/1512.08990 (visited on 11/26/2019).

[Cam13] Peter J. Cameron. “The Random Graph”. Jan. 31, 2013. arXiv: 1301.7544 [math]. URL: http://arxiv.org/abs/1301.7544 (visited on 02/16/2021).

[Car13] Olivia Caramello. “Topological Galois Theory”. Jan. 2, 2013. arXiv: 1301.0300 [math]. URL: http://arxiv.org/abs/1301.0300 (visited on 02/15/2021).

[Car18] Olivia Caramello. *Theories, Sites, Toposes: Relating and Studying Mathematical Theories Through Topos-theoretic ’Bridges’*. Oxford University Press, 2018. 381 pages. ISBN: 978-0-19-875891-4. Google Books: nAJCDwAAQBAJ.

[CL19] Olivia Caramello and Laurent Lafforgue. “Some Aspects of Topological Galois Theory”. In: *Journal of Geometry and Physics* 142 (Aug. 1, 2019), pages 287–317. ISSN: 0393-0440. DOI: 10.1016/j.geomphys.2019.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0393044019300671 (visited on 03/13/2021).

[Che+21] Mark Chen et al. “Evaluating Large Language Models Trained on Code”. July 14, 2021. arXiv: 2107.03374 [cs]. URL: http://arxiv.org/abs/2107.03374 (visited on 04/11/2022).

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

[22] *CodeGen*. Salesforce, Apr. 10, 2022. URL: https://github.com/salesforce/CodeGen (visited on 04/10/2022).

[20] *Deep Probabilistic Programming with Pyro*. Broad Institute. July 17, 2020. URL: https://www.broadinstitute.org/talks/deep-probabilistic-programming-pyro (visited on 08/17/2021).

[Dia08] Persi Diaconis. "The Markov Chain Monte Carlo Revolution". In: *Bulletin of the American Mathematical Society* 46.2 (Nov. 20, 2008), pages 179–205. ISSN: 0273-0979. DOI: 10.1090/s0273-0979-08-01238-x. URL: http://www.ams.org/journal-getitem?pii=S0273-0979-08-01238-X (visited on 04/29/2021).

[DJ07] Persi Diaconis and Svante Janson. "Graph Limits and Exchangeable Random Graphs". Dec. 17, 2007. arXiv: 0712.2749 [math]. URL: http://arxiv.org/abs/0712.2749 (visited on 09/16/2021).

[DS18] Persi Diaconis and Brian Skyrms. *Ten Great Ideas about Chance*. Princeton: Princeton University Press, 2018. 255 pages. ISBN: 978-0-691-17416-7.

[EST] Kevin Ellis, Armando Solar-Lezama and Josh Tenenbaum. "Unsupervised Learning by Program Synthesis". In: (), page 9.

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

[Fie06] Stephen E. Fienberg. "When Did Bayesian Inference Become "Bayesian"?" In: *Bayesian Analysis* 1.1 (Mar. 2006), pages 1–40. ISSN: 1936-0975, 1931-6690. DOI: 10.1214/06-BA101. URL: https://projecteuclid.org/journals/bayesian-analysis/volume-1/issue-1/When-did-Bayesian-inference-become-Bayesian/10.1214/06-BA101.full (visited on 04/03/2022).

[Fin37] Bruno De Finetti. "La prévision : ses lois logiques, ses sources subjectives". In: *Annales de l'institut Henri Poincaré* 7 (1937), page 69. URL: http://www.numdam.org/item/?id=AIHP_1937__7_1_1_0.

[Fra54] Roland Fraïssé. "Sur l'extension aux relations de quelques propriétés des ordres". In: *Annales scientifiques de l'École normale supérieure* 71.4 (1954), pages 363–388. ISSN: 0012-9593, 1873-2151. DOI: 10.24033/asens.1027. URL: http://www.numdam.org/item?id=ASENS_1954_3_71_4_363_0 (visited on 02/16/2021).

[Fre] Cameron Freer. "Computability of Representations of Exchangeable Data in Probabilistic Programming" (CIRM, Marseille).

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

[FGP21] Tobias Fritz, Tomáš Gonda and Paolo Perrone. "De Finetti's Theorem in Categorical Probability". In: *Journal of Stochastic Analysis* 2.4 (Nov. 4, 2021). ISSN: 2689-6931. DOI: 10.31390/josa.2.4.06. URL: https://digitalcommons.lsu.edu/josa/vol2/iss4/6 (visited on 03/26/2022).

[Gal94] Francis Galton. *Natural Inheritance*. Macmillan and Company, 1894. 288 pages. Google Books: a51UeN5hsEQC.

[GGK94] Martin Goldstern, R. Grossberg and Menachem Kojman. "Infinite Homogeneous Bipartite Graphs with Unequal Sides". Sept. 5, 1994. arXiv: math/9409204. URL: http://arxiv.org/abs/math/9409204 (visited on 02/16/2021).

[Goo+14] Ian J. Goodfellow et al. "Generative Adversarial Networks". June 10, 2014. arXiv: 1406.2661 [cs, stat]. URL: http://arxiv.org/abs/1406.2661 (visited on 04/11/2022).

[Heu+17] Chris Heunen et al. "A Convenient Category for Higher-Order Probability Theory". Apr. 18, 2017. arXiv: 1701.02547 [cs, math]. URL: http://arxiv.org/abs/1701.02547 (visited on 02/11/2020).

[Heu+18] Chris Heunen et al. "The Semantic Structure of Quasi-Borel Spaces". In: *Los Angeles* (2018), page 5.

[HS55] E. Hewitt and L. J. Savage. "Symmetric Measures on Cartesian Products". In: (1955). DOI: 10.1090/S0002-9947-1955-0076206-8.

[Hoo79] Douglas N. Hoover. *Relations on Probability Spaces and Arrays of Random Variables*. Institute for Advanced Study, Princeton, 1979. URL: http://www.stat.berkeley.edu/~aldous/Research/hoover.pdf.

[JS20] Bart Jacobs and Sam Staton. "De Finetti's Construction as a Categorical Limit". Sept. 26, 2020. arXiv: 2003.01964 [math]. URL: http://arxiv.org/abs/2003.01964 (visited on 11/13/2020).

[Jai+21] Naman Jain et al. "Jigsaw: Large Language Models Meet Program Synthesis". Dec. 6, 2021. arXiv: 2112.02969 [cs]. URL: http://arxiv.org/abs/2112.02969 (visited on 01/26/2022).

[Kal89] Olav Kallenberg. "On the Representation Theorem for Exchangeable Arrays". In: *Journal of Multivariate Analysis* 30.1 (1989), pages 137–154. DOI: 10.1016/0047-259X(89)90092-4. URL: https://ideas.repec.org/a/eee/jmvana/v30y1989i1p137-154.html (visited on 04/05/2022).

[Kal13] Gopinath Kallianpur. *Stochastic Filtering Theory*. Volume 13. Stochastic Modelling and Applied Probability. Springer Science & Business Media, 2013. 318 pages. ISBN: 978-1-4757-6592-2.

[Kem+06] Charles Kemp et al. "Learning systems of concepts with an infinite relational model". In: *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, AAAI-06/IAAI-06*. 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, AAAI-06/IAAI-06. Nov. 13, 2006, pages 381–388. URL: https://collaborate.princeton.edu/en/publications/learning-systems-of-concepts-with-an-infinite-relational-model (visited on 03/02/2022).

[KW14] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". May 1, 2014. arXiv: 1312.6114 [cs, stat]. URL: http://arxiv.org/abs/1312.6114 (visited on 04/11/2022).

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

[Kol46] Andrei Nikolaevich Kolmogoroff. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Chelsea Publishing Company, 1946. 72 pages. Google Books: PDM5AAAAIAAJ.

[LRT16] Balaji Lakshminarayanan, Daniel M. Roy and Y. Teh. "Mondrian Forests for Large-Scale Regression When Uncertainty Matters". In: *AISTATS* (2016).

[LRT15] Balaji Lakshminarayanan, Daniel M. Roy and Yee Whye Teh. "Mondrian Forests: Efficient Online Random Forests". Feb. 16, 2015. arXiv: 1406.2673 [cs, stat]. URL: http://arxiv.org/abs/1406.2673 (visited on 02/25/2022).

[Lan78] Saunders Mac Lane. *Categories for the Working Mathematician*. 2nd edition. Graduate Texts in Mathematics. New York: Springer-Verlag, 1978. ISBN: 978-0-387-98403-2. URL: https://www.springer.com/gp/book/9780387984032 (visited on 06/23/2019).

[Lev06] Paul Blain Levy. "Call-by-Push-Value: Decomposing Call-by-Value and Call-by-Name". In: *Higher-Order and Symbolic Computation* 19.4 (Dec. 2006), pages 377–414. ISSN: 1388-3690, 1573-0557. DOI: 10/fwb7vh. URL: http://link.springer.com/10.1007/s10990-006-0480-6 (visited on 11/12/2021).

[Lor20] Fosco Loregian. "Coend Calculus". Dec. 11, 2020. arXiv: 1501.02503 [math]. URL: http://arxiv.org/abs/1501.02503 (visited on 12/29/2020).

[Lov12] László Lovász. *Large Networks and Graph Limits*. American Mathematical Society Colloquium Publications volume 60. Providence, Rhode Island: American Mathematical Society, 2012. 475 pages. ISBN: 978-0-8218-9085-1.

[Mel14] Paul-André Melliès. “Local States in String Diagrams”. In: *Rewriting and Typed Lambda Calculi*. Edited by Gilles Dowek. Redacted by David Hutchison et al. Volume 8560. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pages 334–348. ISBN: 978-3-319-08917-1 978-3-319-08918-8. DOI: 10.1007/978-3-319-08918-8_23. URL: http://link.springer.com/10.1007/978-3-319-08918-8_23 (visited on 05/23/2021).

[Nij+22] Erik Nijkamp et al. “A Conversational Paradigm for Program Synthesis”. Mar. 30, 2022. arXiv: 2203.13474 [cs]. URL: http://arxiv.org/abs/2203.13474 (visited on 04/10/2022).

[nLa] nLab authors. *Sheaf Toposes Are Equivalently the Left Exact Reflective Subcategories of Presheaf Toposes*. URL: https://ncatlab.org/nlab/show/sheaf+toposes+are+equivalently+the+left+exact+reflective+subcategories+of+presheaf+toposes (visited on 04/08/2022).

[Nor+15] Aditya Nori et al. “Efficient Synthesis of Probabilistic Programs”. In: Programming Language Design and Implementation (PLDI). June 1, 2015. URL: https://www.microsoft.com/en-us/research/publication/efficient-synthesis-of-probabilistic-programs/ (visited on 01/26/2022).

[ONe09] Ben O’Neill. “Exchangeability, Correlation, and Bayes’ Effect”. In: *International Statistical Review* 77.2 (2009), pages 241–250. ISSN: 1751-5823. DOI: 10.1111/j.1751-5823.2008.00059.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2008.00059.x (visited on 03/27/2022).

[OR15] Peter Orbanz and Daniel M. Roy. “Bayesian Models of Graphs, Arrays and Other Exchangeable Random Structures”. Feb. 13, 2015. arXiv: 1312.7857 [math, stat]. URL: http://arxiv.org/abs/1312.7857 (visited on 12/03/2021).

[Pan19] Dmitry Panchenko. *Lecture Notes on Probability Theory*. Dmitriy Panchenko, Dec. 4, 2019. 320 pages. ISBN: 978-1-9994190-4-2.

[PS21] Hugo Paquet and Sam Staton. “LazyPPL: Laziness and Types in Non-Parametric Probabilistic Programs”. In: Advances in Programming Languages and Neurosymbolic Systems Workshop. Oct. 8, 2021. URL: https://openreview.net/forum?id=yHox9OyegeX (visited on 04/04/2022).

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

[PP02] Gordon Plotkin and John Power. “Notions of Computation Determine Monads”. In: *Foundations of Software Science and Computation Structures*. Edited by Mogens Nielsen and Uffe Engberg. Redacted by Gerhard Goos, Juris Hartmanis and Jan van Leeuwen. Volume 2303. Lecture Notes in Computer Science. Berlin, Heidelberg:

Springer Berlin Heidelberg, 2002, pages 342–356. ISBN: 978-3-540-43366-8 978-3-540-45931-6. DOI: 10.1007/3-540-45931-6_24. URL: http://link.springer.com/10.1007/3-540-45931-6_24 (visited on 05/24/2021).

[PT99] A. John Power and Hayo Thielecke. "Closed Freyd- and Kappa-Categories". In: *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*. ICAL '99. Berlin, Heidelberg: Springer-Verlag, July 11, 1999, pages 625–634. ISBN: 978-3-540-66224-2.

[Rad64] R. Rado. "Universal Graphs and Universal Functions". In: (1964). DOI: 10.4064/AA-9-4-331-340.

[Rai17] Tom Rainforth. "Automating Inference, Learning, and Design Using Probabilistic Programming". Oxford, United Kingdom: University of Oxford, 2017. 252 pages.

[Roy+08] D. Roy et al. "A Stochastic Programming Perspective on Nonparametric Bayes". In: 2008. URL: https://www.semanticscholar.org/paper/A-stochastic-programming-perspective-on-Bayes-Roy-Mansinghka/946ed04f705a2a28539a8af1f5e9ccdedd7fabc2 (visited on 01/21/2022).

[Roy] Daniel M Roy. "Exchangeable Graphs, Conditional Independence, and Computably-Measurable Samplers". In: (), page 19.

[RT09] Daniel M Roy and Yee Teh. "The Mondrian Process". In: *Advances in Neural Information Processing Systems*. Volume 21. Curran Associates, Inc., 2009. URL: https://papers.nips.cc/paper/2008/hash/fe8c15fed5f808006ce95eddb7Abstract.html (visited on 10/16/2021).

[Sim17] Alex Simpson. "Probability Sheaves and the Giry Monad". In: (2017). In collaboration with Marc Herbstritt, 6 pages. DOI: 10.4230/lipics.calco.2017.1. URL: http://drops.dagstuhl.de/opus/volltexte/2017/8051/ (visited on 01/25/2021).

[Sta06] Sam Staton. "Name-Passing Process Calculi: Operational Models and Structural Operational Semantics". PhD thesis. Cambridge, UK: University of Cambridge, 2006. 245 pages.

[Sta14] Sam Staton. "Freyd Categories Are Enriched Lawvere Theories". In: *Electronic Notes in Theoretical Computer Science* 303 (Mar. 2014), pages 197–206. ISSN: 15710661. DOI: 10.1016/j.entcs.2014.02.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S157106611400036X (visited on 03/10/2021).

[Sta17] Sam Staton. “Commutative Semantics for Probabilistic Programming”. In: *Programming Languages and Systems*. Edited by Hongseok Yang. Volume 10201. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pages 855–879. ISBN: 978-3-662-54433-4 978-3-662-54434-1. DOI: 10.1007/978-3-662-54434-1_32. URL: http://link.springer.com/10.1007/978-3-662-54434-1_32 (visited on 02/11/2020).

[Sta20a] Sam Staton. “Categorical Models of Probability with Symmetries”. Categorical Probability and Statistics. June 2020. URL: http://perimeterinstitute.ca/personal/tfritz/2019/cps_workshop/slides/staton.pdf.

[Sta20b] Sam Staton. “Probabilistic Programs as Measures”. In: *Foundations of Probabilistic Programming*. Edited by Alexandra Silva, Gilles Barthe and Joost-Pieter Katoen. Cambridge: Cambridge University Press, 2020, pages 43–74. ISBN: 978-1-108-48851-8. DOI: 10.1017/9781108770750.003. URL: https://www.cambridge.org/core/books/foundations-of-probabilistic-programming/probabilistic-programs-as-measures/B136E6A9577C (visited on 04/03/2022).

[Sta21a] Sam Staton. “Some Formal Structures in Probability”. In: FSCD. 2021, page 4. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14242/pdf/LIPIcs-FSCD-2021-4.pdf.

[Sta21b] Sam Staton. “Some Formal Structures in Probability”. 2021. URL: http://www.cs.ox.ac.uk/people/samuel.staton/2021fscd.pdf.

[Sta22] Sam Staton. “Abstract Types in Probabilistic Programming”. LAFI Workshop (POPL). 2022. URL: https://popl22.sigplan.org/details/lafi-2022-papers/6/Abstract-types-in-probabilistic-programming (visited on 04/05/2022).

[Sta+17] Sam Staton et al. “Exchangeable Random Processes and Data Abstraction”. In: PPS Workshop. Paris, France, 2017, page 4. URL: http://www.cs.ox.ac.uk/people/hongseok.yang/paper/pps17a.pdf.

[Sta+18] Sam Staton et al. “The Beta-Bernoulli Process and Algebraic Effects”. In: (2018). In collaboration with Michael Wagner, 15 pages. DOI: 10.4230/lipics.icalp.2018.141. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9145/ (visited on 11/13/2020).

[Sta+] Sam Staton et al. *LazyPPL*. URL: https://lazyppl.bitbucket.io/ (visited on 04/04/2022).

[SS21] Dario Stein and Sam Staton. “Compositional Semantics for Probabilistic Programs with Exact Conditioning”. Jan. 27, 2021. arXiv: 2101.11351 [cs, math]. URL: http://arxiv.org/abs/2101.11351 (visited on 02/07/2021).

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

[Sto49] M. H. Stone. “Postulates for the Barycentric Calculus”. In: (1949). DOI: 10.1007/BF02413910.

[Tij07] H. C. Tijms. *Understanding Probability: Chance Rules in Everyday Life*. 2nd ed. Cambridge: Cambridge University Press, 2007. 442 pages. ISBN: 978-0-521-70172-3.

[Tol+16] David Tolpin et al. “Design and Implementation of Probabilistic Programming Language Anglican”. In: *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages - IFL 2016*. The 28th Symposium. Leuven, Belgium: ACM Press, 2016, pages 1–12. ISBN: 978-1-4503-4767-9. DOI: 10.1145/3064899.3064910. URL: http://dl.acm.org/citation.cfm?doid=3064899.3064910 (visited on 09/14/2021).

[Tra20] Dustin Tran. “Probabilistic Programming for Deep Learning”. Columbia University, 2020. DOI: 10.7916/d8-95c9-sj96. URL: https://doi.org/10.7916/d8-95c9-sj96 (visited on 08/17/2021).

[Tra+17] Dustin Tran et al. “Deep Probabilistic Programming”. Mar. 7, 2017. arXiv: 1701.03757 [cs, stat]. URL: http://arxiv.org/abs/1701.03757 (visited on 04/11/2022).

[vdMee+21] Jan-Willem van de Meent et al. *An Introduction to Probabilistic Programming*. Oct. 19, 2021. arXiv: 1809.10756. URL: http://arxiv.org/abs/1809.10756 (visited on 12/18/2021).

[Vas+17] Ashish Vaswani et al. “Attention Is All You Need”. Dec. 5, 2017. arXiv: 1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 04/11/2022).

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