Semantics of Higher-Order Probabilistic Programs with Conditioning

FREDRIK DAHLQVIST, University College London, UK and Imperial College London, UK
DEXTER KOZEN, Cornell University, USA

We present a denotational semantics for higher-order probabilistic programs in terms of linear operators between Banach spaces. Our semantics is rooted in the classical theory of Banach spaces and their tensor products, but bears similarities with the well-known semantics of higher-order programs à la Scott through the use of *ordered* Banach spaces which allow definitions in terms of fixed points. Our semantics is a model of intuitionistic linear logic: it is based on a symmetric monoidal closed category of ordered Banach spaces which treats randomness as a linear resource, but by constructing an exponential comonad we can also accommodate non-linear reasoning. We apply our semantics to the verification of the classical Gibbs sampling algorithm.

CCS Concepts: • **Software and its engineering → General programming languages**; • **Social and professional topics** → *History of programming languages.*

Additional Key Words and Phrases: Probabilistic programming, semantics, type system

ACM Reference Format:
Fredrik Dahlqvist and Dexter Kozen. 2020. Semantics of Higher-Order Probabilistic Programs with Conditioning. *Proc. ACM Program. Lang.* 4, POPL, Article 57 (January 2020), 29 pages. https://doi.org/10.1145/3371125

1 INTRODUCTION

Probabilistic programming has enjoyed a recent resurgence of interest driven by new applications in machine learning and statistical analysis of large datasets. The emergence of probabilistic programming languages such as Church and Anglican, which allow statisticians to construct and sample distributions and perform Bayesian inference, has created a need for sound semantic foundations and tools for specification and reasoning. Several recent works have approached this task from various perspectives [Ehrhard et al. 2017, 2014; Heunen et al. 2017; Ścibior et al. 2017; Staton 2017; Vákár et al. 2019].

One of the earliest works on the semantics of probabilistic programs was [Kozen 1981], in which operational and denotational semantics were given for an idealized first-order imperative language with random number generation. Data were interpreted over ordered Banach spaces. Programs were modelled as positive and continuous linear operators on an ordered Banach space of measures. In [Kozen 1985], an equivalent predicate-transformer semantics was introduced based on ordered Banach spaces of measurable functions and shown to be dual to the measure-transformer semantics of [Kozen 1981].

This paper revisits this approach. We identify a symmetric monoidal closed category **RoBan** of regular ordered Banach spaces and regular maps that can serve as a foundation for higher-order

Authors' addresses: Fredrik Dahlqvist, Computer Science, University College London, UK, f.dahlqvist@ucl.ac.uk, Electrical and Electronic Engineering, Imperial College London, UK, f.dahlqvist09@imperial.ac.uk; Dexter Kozen, Computer Science, Cornell University, USA, dexter.kozen@cornell.edu.

2475-1421/2020/1-ART57
https://doi.org/10.1145/3371125

probabilistic programming with sampling, conditioning, and Bayesian inference. Bayesian inference can be viewed as reversing the computation of a probabilistic program to infer information about a prior distribution from observations. We model Bayesian inference as computing the *adjoint* of a linear operator and show how it corresponds to computing a *disintegration*.

The extension to higher types is achieved through a tensor product construction in the category **RoBan** that gives symmetric monoidal closure. Although not cartesian, the construction does admit an adjunction with homsets enriched with an ordered Banach space structure acting as internalized exponentials. To accommodate conditioning and Bayesian inference, we introduce 'Bayesian types', in which values are decorated with a prior distribution. Based on this foundation, we give a type system and denotational semantics for an imperative higher-order probabilistic language with sampling, conditioning, and Bayesian inference.

Being linear our system treats randomness as a *resource*, a perspective which fits well with the view of entropy as a computation resource, like time or space. True random number generators can only produce randomness at a limited rate; physically, randomness *is* a resource [Hayes 2001]. Being resource-sensitive, our type system has some nice crypographical properties: by default it is forbidden to use a sample more than once; that is, each operation consuming a random sample requires a fresh sample (component in a tensor product). However, as we shall see with the example of Gibbs sampling in § 2 and § 6, it is often desirable to have data types which are *duplicable*. For this reason, our system also includes an *exponential* type constructor, and our model is thus linear-non-linear in the same sense as [Mellies 2009; Selinger and Valiron 2008]. In fact, our semantics is a model of *intuitionistic linear logic*: the symmetric monoidal closed structure provides the interpretation of the tensor $\otimes$ and linear implication $\multimap$, and the exponential type constructor provides the interpretation of the unary operator ! and of classical implication via the usual encoding $A \to B \;= !A \multimap B$.

We believe our approach should appeal to computer scientists, as it is true to traditional Scott-style denotational semantics (see § 5.3.3) as well as being a model of intuitionistic linear logic.

On the other hand, we also believe that defining a semantics in terms of linear operators will appeal to mathematicians, statisticians and machine learning theorists. This is for two reasons. First, because the most natural way to represent Markov process is arguably through *stochastic operators* (i.e. positive operators between Banach lattices, which preserve the norm of positive vectors [Aliprantis and Border 1999, Ch. 19]). This simply generalises the fact that *stochastic matrices* are the natural way to represent Markov chains. Stochastic operators are precisely the morphisms of our semantics. Second, because working with linear operators connects our semantics to an immense body of classical results from linear algebra and functional analysis. We shall see in particular that standard results from *ergodic theory* – an important part of the spectral theory of linear operators [Dunford et al. 1971; Eisner et al. 2015] – will prove crucial in the verification of the correctness of Gibbs sampling in § 6, and can be applied directly to our denotational semantics. We believe that this seamless connection between program semantics and mathematics will simplify the task of validating stochastic machine learning algorithms.

Related Works: Since the formalisms for describing Markov processes (stochastic operators or Kleisli arrows for the Giry monad) form categories which cannot be Cartesian closed, two strategies are possible when designing a semantics for probabilistic programs: either keep the familiar formalism and abandon Cartesian closedness in favour of a monoidal closed system, or retain Cartesian closedness and design a new mathematical universe which can nonetheless be used to encode probabilistic processes. To our knowledge, we provide the first comprehensive semantics following the first strategy.

Two very powerful semantics for higher-order probabilistic programming have been recently developed in the literature following the second strategy. In [Heunen et al. 2017; Ścibior et al.

2017], a semantics is given in terms of so-called quasi-Borel spaces. These form a Cartesian closed category and admit a notion of probability distribution and of a Giry-like monad of probability distributions. In [Ehrhard et al. 2017] the authors develop a semantics in terms of measurable cones. These form a cpo-enriched Cartesian closed category which provides a semantics to a probabilistic extension of PCF that includes conditioning. The key differences with the present semantics are the following. First, these proposed mathematical universes come directly from the world of theoretical computer science, whilst as mentioned above, our semantics is rooted in the traditional mathematics of the objects being constructed by the programs. Second, quasi-Borel spaces and measurable cones form Cartesian closed categories, whereas we work in a monoidal closed category, with obvious implications in terms of resources (e.g. we cannot copy non-duplicable types). Finally, our semantics of conditioning has been reduced to a mathematically very simple, but also very general construction (taking the adjoint of a linear operator, see § 5.2.9), whilst in [Heunen et al. 2017] un-normalized posteriors and normalization constants are computed pointwise, and [Ehrhard et al. 2017] hard-codes the rejection-sampling algorithm into the semantics. This being said, our work is very closely related to [Ehrhard et al. 2017]. In fact, we believe that the exponential comonad constructed in § 5.1.4 establishes a bridge between the two models which *in fine* correspond to the multiplicative fragment (our work) and the additive fragment ([Ehrhard et al. 2017]) of a common system. We conjecture that each pre-stable function $f : E \to F$ from [Ehrhard et al. 2017] corresponds to a positive operator $\tilde{f} : !\tilde{E} \to \tilde{F}$ – where $\tilde{E}$ is the space generated by the cone E – in our system and vice-versa.

2 A TARGET: VERIFYING THE CORRECTNESS OF GIBBS SAMPLING

Gibbs sampling [Geman and Geman 1987] is one of the classic Monte Carlo Markov Chain (MCMC) sampling algorithms. It is a simple algorithm whose purpose is to sample from a joint distribution $\mathbb{P}$ on a space $X_1 \times \ldots \times X_n$ when all the conditional distributions $\mathbb{P}(X_i \mid X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ are known (we keep the notation informal for the moment, we will describe the algorithm rigorously in § 6). Starting from an initial guess $(x_1, \ldots, x_n)$, the algorithm cyclically updates the components of the vector by sampling from the corresponding conditional distribution. Thus starting from $(x_1, \ldots, x_n)$ the algorithm will first update x_1 by sampling $\mathbb{P}(X_1 \mid X_2, \ldots, X_n)$ evaluated at $(x_2, \ldots, x_n)$, then update x_2 by sampling $\mathbb{P}(X_2 \mid X_1, X_3, \ldots, X_n)$ evaluated at $(x_1, x_3, \ldots, x_n)$ where x_1 was updated at the previous step, but x_3 is still the original guess, etc. The loop repeats once x_n has been updated, and exits after a pre-determined number of iterations.

For simplicity's sake, a three-dimensional Gibbs sampling algorithm is presented in Fig. 1. The algorithm is written in the probabilistic language which we will describe in § 4. The reader will immediately notice that the program does not terminate, and indeed the Gibbs sampler of Fig. 1 is an idealization which abstracts away the thorny question of how many iterations of the loop are enough to provide a 'good' sample. We consider that we can run the loop infinitely often, the purpose of the exercise being to show the correctness of the algorithm in the limit where the number of iterations of the loop tends to infinity. Let us make a few comments about the syntax of the language in Fig. 1. Note first that this is a *typed, higher-order, imperative language*. The first three inputs represent the conditional distributions. They are implemented as higher-order types whose codomains are built using a type constructor for measure types. They are also tagged as *duplicable types* via the exponential type constructor ! since they will be repeatedly called during the execution of the loop. The square brackets `[i, j]` are just syntactic sugar for the obvious projections. Note also the presence of a `while` loop; our language will allow certain fixpoint operations (as in [Kozen 1981]).

```
/* Pre-conditions:
1) cond_1, cond_2, cond_3 are the conditional distributions of
   a joint distribution μ on ⟦X×Y×Z⟧
2) the initial xyz is sampled absolutely continuous w.r.t. μ
*/
gibbs_sampling(cond_1: !(XxY -> M(XxYxZ)), cond_2: !(XxZ -> M(XxYxZ)),
               cond_3: !(YxZ -> M(XxYxZ)), xyz: XxYxZ): XxYxZ
{
        while true do
                xyz:=write_Z(xyz, sample(cond_1(xyz[1,2])));
                xyz:=write_Y(xyz, sample(cond_2(xyz[1,3])));
                xyz:=write_X(xyz, sample(cond_3(xyz[2,3])));
        return xyz
}
```

Fig. 1. Idealized three-dimensional Gibbs sampler

We will provide a simple proof of the correctness of the algorithm in Fig 1 in § 6. The proof that Gibbs sampling is correct, see for example the original [Geman and Geman 1987, §XXII], i.e. that it does sample from the joint distribution in the limit, relies on results from *ergodic theory*. This is typical of MCMC sampling algorithms in which a distribution is computed as the equilibrium distribution of some cleverly designed Markov chain. The 'sampleability' of the joint distribution (the 'space average' in ergodic terminology) by repeated iteration of the operator representing the chain (the 'time average') is as the heart of ergodic theorems [Dunford et al. 1971; Eisner et al. 2015]. Our correctness proof will use the same ideas and, as hinted to in the introduction, our denotational semantics will therefore target a mathematical universe in which ergodicity and ergodic theorems can be expressed naturally. Finally, let us mention the fact that although the program above does not contain any conditioning instruction, its proof of correctness will nonetheless rely on the mathematics which will interpret the conditioning operation (technically, because the input conditional distributions are the *disintegrations* of the projections onto two components).

Our task in the next sections will therefore be to design a type system and a language, and to provide them with a denotational semantics which can accommodate:

(1) continuous distributions
(2) all the imperative commands of [Kozen 1981], including the fixpoints defined by while loops
(3) higher-order types
(4) duplicable types
(5) conditioning
(6) results from ergodic theory

3 PRELIMINARIES

The semantics developed in [Kozen 1981] is framed in terms of operators between *Banach lattices*. We start by presenting some basic facts about these important objects, and show how they relate to the important probabilistic notion of *disintegration* which semantically underpins conditioning in probabilistic programs. We will then see that Banach lattices do not form a closed category and we will therefore present a generalisation of Banach lattices, called *regular ordered Banach spaces* (the category **RoBan**), which forms a complete, symmetric monoidal closed category.

3.1 Banach Lattices

3.1.1 Definitions and Examples. An *ordered vector space* V is a vector space together with a partial order $\leq$ which is compatible with the linear structure in the sense that for all $u, v, w \in V, \lambda \in \mathbb{R}^+$

$$u \leq v \Rightarrow u + w \leq v + w \qquad \text{and} \qquad u \leq v \Rightarrow \lambda u \leq \lambda v$$

A vector v in an ordered vector space V is called *positive* if $v \geq 0$ and the collection of all positive vectors is called the *positive cone* of V and denoted V^+. The positive cone entirely specifies the order since $x \leq y$ iff there exists $0 \leq u$ such that $x + u = y$. The positive cone is said to be *generating* if $V = V^+ - V^+$, i.e. if every vector can be expressed as the difference of two positive vectors.

An ordered vector space $(V, \leq)$ is a *Riesz space* if its partial order is a lattice. This allows the definition of the *positive and negative part of a vector* $v \in V$ as $v^+ = v \vee 0, v^- = (-v) \vee 0$ and its *modulus* as $|v| = v \vee (-v)$. Note that $v = v^+ - v^-$, with v^+, v^- positive, and the positive cone of a Riesz space is thus generating. A Riesz space is *order complete* or *Dedekind-complete* (resp. σ-order complete or σ-Dedekind complete) if every non-empty (resp. non-empty countable) subset of V which is order bounded has a supremum[1]. A *normed Riesz space* is a Riesz space equipped with a *lattice norm*, that is to say a norm satisfying:

R1 if $-y \leq x \leq y$ then $\|x\| \leq \|y\|$

A normed Riesz space is called a *Banach lattice* if it is (norm-)complete, i.e. if every Cauchy sequence converges to an element of the space.

Example 1. *Given a measurable space $(X, \mathcal{F})$ we introduce the space $\mathcal{M}(X, \mathcal{F})$, or simply $\mathcal{M}X$, as the set of signed measures of bounded variation over X. $\mathcal{M}X$ is a Banach space: the linear structure is inherited pointwise from $\mathbb{R}$, and the norm is given by the total variation. The space $\mathcal{M}(X, \mathcal{F})$ can also be shown to be a Banach lattice for the lattice structure given by*

$$(\mu \vee \nu)(A) = \sup\{\mu(B) + \nu(A \setminus B) \mid B \text{ measurable}, B \subseteq A\}$$

and dually for meets. The Hahn-Jordan decomposition theorem provides the positive part $\mu^+ = \mu \vee 0$ and negative part $\mu^- = -\mu \vee 0$ of a signed measure $\mu \in \mathcal{M}(X, \mathcal{F})$.

Example 2. *Given a measured space $(X, \mathcal{F}, \mu)$ and $1 \leq p < \infty$, the* Lebesgue space $\mathrm{L}_p(X, \mu)$ *is the set of equivalence classes of μ-almost everywhere equal, p-integrable, real-valued functions, that is to say functions $f : X \to \mathbb{R}$ such that*

$$\int |f|^p \ d\mu < \infty.$$

The linear structure is inherited pointwise from $\mathbb{R}$ and the norm is given by $\|f\|_p = \int |f|^p \ d\mu$. When $p = \infty$, the space $\mathrm{L}_\infty(X, \mu)$ is defined as the set of equivalence classes of μ-almost everywhere equal bounded real-valued functions with the norm given by the essential supremum:

$$\|f\|_\infty = \inf\{C \geq 0 \mid |f(x)| \leq C \ \mu\text{-a.e.}\}.$$

Lebesgue spaces are Banach lattices when equipped with the pointwise order. In particular, for any $f \in \mathrm{L}_p(X, \mu)$, the positive and negative parts f^+ and f^- of a function used in the definition of the Lebesgue integral defines the positive-negative decomposition of f in the Banach lattice $\mathrm{L}_p(X, \mu)$. We will say that $p, q \in \mathbb{N} \cup \{\infty\}$ are Hölder conjugate *if either of the following conditions hold: (i) $1 < p, q < \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$, or (ii) $p = 1$ and $q = \infty$, or (iii) $p = \infty$ and $q = 1$.*

The most important spaces for what follows are the Banach lattices $\mathcal{M}X$ of signed measures over a measurable space, and the Lebesgue spaces $\mathrm{L}_1(X, \mu)$. These are instances of a class of objects

[1] Order-completeness was called *conditional completeness* in [Kozen 1981]

called *abstract Lebesgue spaces* or *AL-spaces* which are characterised by the interaction of the norm with the additive structure. A Banach lattice V is an AL-space iff for all $u, v \in V^+$ with $u \wedge v = 0$

$$\|u + v\| = \|u\| + \|v\| \tag{AL}$$

3.1.2 Order and Norm-Convergence. There are two modes of convergence in a Banach lattice: *order convergence* and *norm convergence*. The latter is well-known, the former less so. Let D be a directed set, and let $\{v_\alpha\}_{\alpha \in D}$ be a net in an ordered Banach space V. We say that $\{v_\alpha\}$ *converges in order to* v if there exists a *decreasing* net $\{u_\alpha\}_{\alpha \in D}$ with $\bigwedge u_\alpha = 0$ – notation $u_\alpha \downarrow 0$ – s.th.

$$-u_\alpha \leq v_\alpha - v \leq u_\alpha \text{ for all } \alpha \in D$$

If the directed set D is $\mathbb{N}$ we get the notion of *order-convergent sequence*. Order and norm convergence of sequences are disjoint concepts, i.e. neither implies the other (see [Zaanen 2012, Ex. 15.2] for two counter-examples). However if a sequence converges both in order and in norm then the limits are the same (see [Zaanen 2012, Th. 15.4]). Moreover, for *monotone* sequences norm convergence implies order convergence [Zaanen 2012, Th. 15.3].

It is well known that bounded operators are continuous, i.e. preserve norm-converging sequences. The corresponding order-convergence concept is defined as follows: an operator $T : V \to W$ between ordered vector spaces is said to be *σ-order continuous* if $Tv_n \downarrow 0$ whenever $v_n \downarrow 0$ [2]. We can thus consider two types of dual spaces on an ordered Banach space V: on the one hand we can consider the *norm-dual*:

$$V^* = \{f : V \to \mathbb{R} : f \text{ is norm-continuous}\}$$

and on the other the *σ-order-dual*:

$$V^\sigma = \{f : V \to \mathbb{R} : f \text{ is } \sigma\text{-order continuous}\}$$

The latter is also known as the *Köthe dual* of V [Dieudonné 1951; Zaanen 2012].

Theorem 3. *The Köthe dual of a Banach lattice is an order-complete Banach lattice.*

Example 4. *It is shown in e.g. [Chaput et al. 2014; Zaanen 2012] that* $\mathrm{L}_p(X,\mu)^\sigma = \mathrm{L}_q(X,\mu)$ *for* any *Hölder conjugate pair* $1 \leq p, q \leq \infty$. *In particular the spaces* $\mathrm{L}_1(X,\mu)$ *and* $\mathrm{L}_\infty(X,\mu)$ *are Köthe dual of each other, although they are* not *ordinary duals.*

3.1.3 Bands. The order structure of Riesz spaces gives rise to classes of subspaces which are far richer than the traditional linear subspaces. An *ideal* of a Riesz space V is a linear subspace $U \subseteq V$ with the property that if $|u| \leq |v|$ and $v \in U$ then $u \in U$. An ideal U is called a *band* when for every subset $D \subseteq U$ if $\bigvee D$ exists in V, then it also belongs to U. Every band in a Banach lattice is itself a Banach lattice. Of particular importance in what follows will be the *principal band generated by an element* $v \in V$, which we denote V_v and can be described explicitly by

$$V_v = \{w \in V \mid (|w| \wedge n\,|v|) \uparrow |w|\}$$

where $n\,|v|$ is the sequence given by the scalar multiplication of $n \in \mathbb{N}$ with $|v|$.

Example 5. *Given a measure* $\mu \in \mathcal{M}X$, *the band* $(\mathcal{M}X)_\mu$ *generated by* μ *is the set of signed measures of bounded variation which are absolutely continuous w.r.t.* μ *[Aliprantis and Border 1999, Th. 10.61]. Note that* $(\mathcal{M}X)_\mu$ *is typically a* much *larger subspace than the (one-dimensional) subspace spanned from* μ *using the linear structure only. The ordered version of the Radon-Nikodym theorem states that* $(\mathcal{M}X)_\mu \simeq \mathrm{L}_1(X,\mu)$ *as Banach lattices [Aliprantis and Border 1999, Th. 13.19].*

[2] Equivalently: if $Tv_n \uparrow Tv$ whenever $v_n \uparrow v$, i.e. whenever v_n is an increasing sequence with $\bigvee v_n = v$. Note the similarity with Scott-continuity, the only difference being the condition that sequences must be order-bounded.

3.2 Disintegrations

3.2.1 Basic Definitions. Let **Meas** denote the category of measurable spaces and measurable functions. The Giry monad [Giry 1982] is defined as the functor $\mathcal{G} : \mathbf{Meas} \to \mathbf{Meas}$ associating to $(X, \mathcal{F})$ the set $\mathcal{G}X$ of probability measures on X, equipped with the smallest σ-algebra making all evaluation maps $ev_B : \mathcal{G}X \to \mathbb{R}$, $\mu \mapsto ev_B(\mu) = \mu(B), B \in \mathcal{F}$ measurable. On a morphism $f : (X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$, $\mathcal{G}f : \mathcal{G}X \to \mathcal{G}Y$ is defined as the map $\mu \mapsto f_*(\mu)$, sending μ to the pushforward measure under f. A *Markov kernel* is a measurable map $f : X \to \mathcal{G}Y$, and we will often refer to these maps simply as *kernels*. We can generalise the pushforward operation to kernels $f : X \to \mathcal{G}Y$ by defining $f_* : \mathcal{G}X \to \mathcal{G}Y$ as

$$f_*(\mu)(B) = \int_X f(x)(B)\, d\mu. \tag{1}$$

With these definitions in place we can introduce the important notion of *disintegration* which underlies the semantics of Bayesian conditioning (see § 5.2.9). We provide a slightly simplified version of the definition which will be enough for our purpose (see [Chang and Pollard 1997, Def. 1] for a very general definition). Intuitively, given a measurable map $f : X \to Y$ and a probability measure μ on X, we say that μ *has a disintegration w.r.t.* f if the fibres $f^{-1}(y)$ of f can be equipped with probability measures $f^\dagger_\mu(y)$ which average out to μ over the pushforward measure $f_*(\mu)$. Formally, the disintegration of μ w.r.t. to f is a kernel $f^\dagger_\mu : Y \to \mathcal{G}X$ such that

- $f_*(f^\dagger_\mu(y)) = \delta_y$ for $f_*(\mu)$-almost all $y \in Y$
- $(f^\dagger_\mu)_*(f_*(\mu)) = \mu$

As can be seen from the first condition, a disintegration – if it exists at all – is only defined up to a null set for the pushforward measure. For sufficiently well-behaved spaces, for example standard Borel spaces [Kechris 1995, 17.35] or more generally metric spaces with Radon measures [Chang and Pollard 1997, Th. 1], disintegrations can be shown to always exist.

3.2.2 Bayesian Inversion. The notion of disintegration is key to the understanding of Bayesian conditioning. The traditional setup is as follows: we are given a kernel $f : X \to \mathcal{G}Y$ where X is regarded as a parameter space and f is regarded as a parametrized statistical model on Y, a space of observable values. We also start with a probability distribution μ on X (the prior) which is regarded as the current state of belief of where the 'true' parameters of the model lie. The problem is, given an observation $y \in Y$, to update the state of belief μ to a new distribution (the posterior) reflecting the observation. We must therefore find a kernel going in the opposite direction $f^\dagger_\mu : Y \to \mathcal{G}X$. As shown in [Clerc et al. 2017; Dahlqvist et al. 2018] this reverse kernel can be built using a disintegration as follows. First we define a joint distribution $\gamma \in \mathcal{G}(X \times Y)$ defined by

$$\gamma(A \times B) = \int_A f(x)(B)\, d\mu,$$

The Bayesian inverse $f^\dagger$, if it exists, is given by the kernel

$$f^\dagger_\mu = (\pi_X)_* \circ (\pi_Y)^\dagger_\gamma$$

where $(\pi_Y)^\dagger_\gamma$ is the disintegration of the measure γ along the projection $\pi_Y : X \times Y \to Y$.

3.2.3 Categorical Connections. We conclude this section on disintegrations with a summary of some results from [Dahlqvist et al. 2018] which provide a categorical connection between Banach lattices, kernels, and disintegrations. We define the category **Krn** as the category whose objects are pairs (X, μ) where X is standard Borel spaces [Kechris 1995] and $\mu \in \mathcal{G}X$. A morphism between (X, μ) and (Y, ν) is a measure kernel $f : X \to \mathcal{G}Y$ such that $f_*(\mu) = \nu$ (where f_* is defined in (1)),

in which case the morphism is denoted f as well. As was shown in [Dahlqvist et al. 2018], any two morphisms which disagree only on a null set can be identified, and the morphisms of **Krn** can be taken to be *equivalence classes* of almost everywhere equal measure kernels.

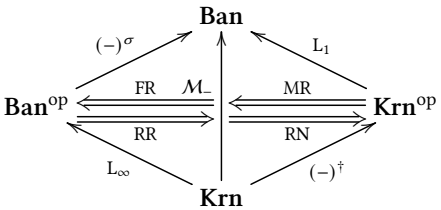
Now, we define two contravariant endofunctors. First, as was shown in [Dahlqvist et al. 2018], the Bayesian inversion operation described in § 3.2.2 defines a functor $(-)^\dagger : \mathbf{Krn} \to \mathbf{Krn}^{\mathrm{op}}$ which leaves objects unchanged and sends a morphism $f : (X, \mu) \to (Y, \nu)$ to its Bayesian inverse $f^\dagger : (Y, \nu) \to (X, \mu)$ (we drop the subscript μ of $f^\dagger_\mu$ because it is made explicit from the typing). Note that $(f^\dagger)^\dagger = f$. We also define the functor $(-)^\sigma : \mathbf{Ban} \to \mathbf{Ban}^{\mathrm{op}}$ which sends a Banach lattice to its Köthe dual, and an operator $T : U \to V$ to its adjoint $T^\sigma : V^\sigma \to U^\sigma$ defined in the usual way via the equation $\phi(Tu) = T^\sigma(\phi)(u)$ for all $u \in U, \phi \in V^\sigma$. Note that just as taking the Köthe dual gives an order-complete space, the adjoint T^σ of an operator is an order-continuous operator [Zaanen 2012, Ch. 26].

Connecting the categories, we define for each $1 \leq p \leq \infty$ the functor $\mathrm{L}_p : \mathbf{Krn} \to \mathbf{Ban}^{\mathrm{op}}$ which sends a **Krn**-object (X, μ) to the Lebesgue space $\mathrm{L}_p(X, \mu)$ and a **Krn**-arrow $f : (X, \mu) \to (Y, \nu)$ to the operator $\mathrm{L}_p f : \mathrm{L}_p(Y, \nu) \to \mathrm{L}_p(X, \mu)$, $\phi \mapsto \lambda x \,.\, \int_Y \phi \, df(x)$. We also define the functor $\mathcal{M}_- : \mathbf{Krn} \to \mathbf{Ban}$ which sends an object (X, μ) to the band $(\mathcal{M}X)_\mu$ and a morphism $f : (X, \mu) \to (Y, \nu)$ to the operator $\mathcal{M}f : (\mathcal{M}X)_\mu \to (\mathcal{M}Y)_\nu$, $\rho \mapsto \lambda B \,.\, \int_X f(x)(B) \, d\rho$.

Finally, we can connect the functors $\mathcal{M}_-, \mathrm{L}_1 \circ (-)^\dagger$ and $(-)^\sigma \circ \mathrm{L}_\infty$ of type $\mathbf{Krn} \to \mathbf{Ban}$ via natural transformations which play a major role in measure theory [Dahlqvist et al. 2018]:

- $\mathrm{RN} : \mathcal{M}_- \to \mathrm{L}_1 \circ (-)^\dagger$ at (X, μ) sends a measure $\nu \ll \mu$ to its Radon-Nikodym derivative $\frac{d\nu}{d\mu}$.
- $\mathrm{MR} : \mathrm{L}_1 \circ (-)^\dagger \to \mathcal{M}_-$ at (X, μ) sends an L_1-map f to its Measure Representation $f\mu$.
- $\mathrm{FR} : \mathcal{M}_- \to (-)^\sigma \circ \mathrm{L}_\infty$ at (X, μ) sends a measure μ to its Functional Representation $\lambda\phi. \int \phi \, d\mu$.
- $\mathrm{RR} : (-)^\sigma \circ \mathrm{L}_\infty \to \mathcal{M}_-$ at (X, μ) sends an L_∞-functional F to its Riesz Representation $\lambda B.F(1_B)$.

The natural transformations RN and MR are inverse of each other, as are FR and RR, proving natural isomorphisms between the three functors. These relationships are summarized in the diagram:



(2)

3.3 The Category RoBan

3.3.1 **Ban** *Is Not Monoidal Closed.* The category **Ban** of Banach lattices provides a very natural semantic universe for the interpretation of first-order probabilistic programs [Kozen 1981], and, as illustrated by Diagram (2), is deeply connected to well-known measure-theoretic constructions, including the notion of disintegration [Dahlqvist et al. 2018] which underlies the semantics to conditioning. However, **Ban** lacks a monoidal closed structure in which to interpret higher-order programs.

It is shown in for example [Aliprantis and Burkinshaw 2006, Example 1.17] or [Wickstead 2007, §3] that the space of operators between two Riesz spaces need not even be a lattice. A set of additional conditions is presented in [Kozen 1981, §5] in order for higher-order types to be Banach lattices. Unfortunately these additional conditions are not stable under the natural tensor operation

(described in § 3.3.3 below) which provides the monoidal structure. The solution is to consider a larger category, described in [Min 1983], which can be equipped with a closed monoidal structure.

3.3.2 Basic Definitions. An *ordered normed vector space* V is an ordered vector space in which the positive cone V^+ is closed for the topology generated by the norm. A subset of the positive cone of particular importance will be the *positive unit ball* $B^+(V) = \{v \geq 0 : \|v\| \leq 1\}$. An *ordered Banach space* is an ordered normed vector space which is complete. We can now describe the central class of object of this work: an ordered normed space is said to be *regular* [Davies 1968; Wong and Ng 1973] if its norm is a lattice norm, i.e. satisfies **R1**, and additionally satisfies

R2 $\|x\| = \inf\{\|y\| : -y \leq x \leq y\}$

A regular ordered Banach space is an ordered Banach space which is regular. A few comments are in order. First note that if $-y \leq y$ then $0 \leq 2y$, and thus y is positive, so **R2** says that the norm of any vector can be approximated arbitrarily well by the norm of positive vectors. **R2** also implies that the positive cone is generating: for any $x \in V$, fix $\epsilon > 0$, then by **R2** there exists y with $-y \leq x \leq y$ whose norm is ϵ-close to that of x. Since $x = \frac{y+x}{2} - \frac{y-x}{2}$, and since it follows from $-y \leq x \leq y$ that $y + x$ and $y - x$ are positive, x can indeed be expressed as the difference of two positive vectors.

Regularity can be understood as the fact that the space is fully characterised by its positive unit ball [Min 1983]. It is therefore natural to consider linear operators $f : U \to V$ between regular ordered Banach spaces which send positive vectors to positive vectors, i.e. such that $u \geq 0 \Rightarrow f(u) \geq 0$. Such operators are called *positive operators* and constitute a field of mathematical research in their own right [Aliprantis and Burkinshaw 2006; Zaanen 2012]. The collection $[U, V]^+$ of positive operators between two regular ordered Banach spaces clearly does not form a vector space, and we therefore consider the span of this collection, that is to say the operators $f : U \to V$ which can be expressed as the difference between two positive operators, i.e. $f = h - g$ with $h, g \in [U, V]^+$. Such operators are called *regular operators*, and we define the category **RoBan** as *the category whose objects are regular ordered Banach spaces and whose morphisms are regular operators.* We will also take regular operators as the appropriate notion of morphism in **Ban** and note that: (a) Diagram 2 remains well-typed since the functors involved turn **Krn**-morphisms into positive operators, and (b) the failure of **Ban** to be closed under taking spaces of operators is already witnessed at the level of regular operators (see references cited in § 3.3.1).

Proposition 6. *Banach lattices are regular, in particular* **Ban** *is a sub-category of* **RoBan**.

Regular operators have the following important properties.

Proposition 7. *Regular operators on regular ordered Banach spaces are (norm)-bounded and thus continuous.*

Theorem 8 ([Min 1983])**.** *If U, V are regular ordered Banach spaces and $[U, V]$ is equipped with the obvious linear structure, pointwise order and the* regular norm

$$\|f\|_r = \inf\{\|g\| : -g \leq f \leq g\}$$

where $\|g\| = \sup\{\|g(u)\| : \|u\| \leq 1\}$ is the usual operator norm, then $[U, V]$ is a regular ordered Banach space.

This result justifies the following notation: we will denote the regular ordered Banach space of regular operators between the regular ordered Banach spaces U, V by $[U, V]$.

3.3.3 The Symmetric Monoidal Structure of **RoBan**. Tensor products of Banach spaces were originally developed by Grothendieck [Grothendieck 1955] who introduced a whole family of constructions, the most famous of which is arguably the *projective tensor product* defined by the *projective*

norm (see [Ryan 2013] for a gentle introduction to tensor products of Banach spaces). The theory was then extended to ordered Banach spaces in *inter alia* [Fremlin 1972, 1974; Min 1983; Wittstock 1974], with an emphasis on adapting the projective tensor product. The main idea is to adapt Grothendieck's definition of the projective norm in a way which reflects the central role of *positive* vectors in the theory of regular ordered Banach spaces, and in particular the fact that the positive cone is generating and determines the norm (**R1**, **R2**). This basic intuition gives rise to the definition of the *positive projective norm*: given two regular ordered Banach spaces U, V, their algebraic tensor product $U \otimes V$ is equipped with the positive projective norm $\|\cdot\|_{|\pi|}$ defined as

$$\|x\|_{|\pi|} = \inf \left\{ \sum_{i=1}^{n} \|u_i\| \, \|v_i\| : u_i \in U^+, v_i \in V^+, -\sum_{i=1}^{n} u_i \otimes v_i \leq x \leq \sum_{i=1}^{n} u_i \otimes v_i \right\}$$

As in the classical unordered case, $U \otimes V$ is not complete for the positive projective norm, and we must therefore take its completion which we call the *positive projective tensor product* of U and V and denote by $U \mathbin{\widehat{\otimes}_{|\pi|}} V$. The closure under $\|-\|_{|\pi|}$ of the positive cone for $U \otimes V$ given by

$$U^+ \otimes V^+ = \left\{ \sum_{i=1}^{n} u_i \otimes v_i : u_i \in U^+, v_i \in V^+ \right\}$$

is a positive generating cone for $U \mathbin{\widehat{\otimes}_{|\pi|}} V$. As was described in 3.1 this positive cone uniquely characterises the order structure of $U \mathbin{\widehat{\otimes}_{|\pi|}} V$.

Since L_1-spaces and $\mathcal{M}(X, \mathcal{F})$-spaces are examples of AL-spaces, the following result shows that we can in practice often ignore the subtleties of the positive projective tensor product and rely on the descriptions of the ordinary projective tensor products of Banach spaces [Ryan 2013].

Theorem 9 ([Fremlin 1974], Th. 2B)**.** *If E is an AL-space and F is a regular ordered Banach space, then $E \mathbin{\widehat{\otimes}_{|\pi|}} F$ is isomorphic as Banach space to the usual projective tensor product $E \mathbin{\widehat{\otimes}_{\pi}} F$.*

Theorem 10 (Radon-Nikodym and [Ryan 2013])**.** *For finite measures μ, ν on measurable spaces X, Y respectively, $(\mathcal{M}X)_\mu \mathbin{\widehat{\otimes}_{\pi}} (\mathcal{M}Y)_\nu \simeq (\mathcal{M}(X \times Y))_{\mu \times \nu}$.*

3.3.4 The Closed Monoidal Structue of **RoBan**. Through its familiar universal property, the tensor product of two vector spaces linearizes *bilinear* maps. Similarly, the projective tensor product of two Banach spaces linearizes *bounded bilinear* maps. The positive projective tensor product fulfils the same role for *positive* (and thus bounded by Prop. 7) bilinear maps [Wittstock 1974, 2.7]. Specifically, there exists a universal positive bilinear map $U \times V \to U \mathbin{\widehat{\otimes}_{|\pi|}} V$ such that for any positive bilinear map $f : U \times V \to W$ there exists a unique positive *linear* map $\tilde{f} : U \mathbin{\widehat{\otimes}_{|\pi|}} V \to W$ making the following diagram commutes:

$$\begin{array}{ccc} U \times V & \xrightarrow{\otimes} & U \mathbin{\widehat{\otimes}_{|\pi|}} V \\ {\scriptstyle f}\downarrow & \swarrow_{\tilde{f}} & \\ W & & \end{array} \tag{3}$$

This universal property of tensor products provides a definition of $\widehat{\otimes}_{|\pi|}$ as a bifunctor on **RoBan**. Let $f : U \to X, g : V \to Y$ be positive operators, then the map

$$\widehat{\otimes}_{|\pi|} \circ (f \times g) : U \times V \to X \times Y \to X \mathbin{\widehat{\otimes}_{|\pi|}} Y$$

is positive and bilinear, and thus there exists a unique positive operator $U \mathbin{\widehat{\otimes}_{|\pi|}} V \to X \mathbin{\widehat{\otimes}_{|\pi|}} Y$ which is denoted $f \mathbin{\widehat{\otimes}_{|\pi|}} g$. This provides the definition of the bifunctor $\widehat{\otimes}_{|\pi|}$ on morphisms. As

we saw in Th. 8, the category **RoBan** has internal homs, and these interact correctly with positive projective tensor products.

Theorem 11 ([Min 1983]). *For every regular ordered Banach space U, the tensoring and homming operations $- \widehat{\otimes}_{|\pi|} U$ and $[U, -]$ define functors* **RoBan** $\to$ **RoBan** *such that*

$$- \widehat{\otimes}_{|\pi|} U \dashv [U, -]$$

The positive projective tensor defines a symmetric monoidal structure on **RoBan** with $\mathbb{R}$ as its unit – since $U \otimes \mathbb{R} \simeq U \simeq \mathbb{R} \otimes U$ at the level of the underlying vector spaces – and the obvious isomorphisms $U \widehat{\otimes}_{|\pi|} V \to V \widehat{\otimes}_{|\pi|} U$ inherited from the isomorphism $U \otimes V \to V \otimes U$ between the algebraic tensor product. The category **RoBan** is thus *symmetric monoidal closed*.

3.3.5 Completeness. We end this mathematical prologue by sketching how limits are computed in **RoBan**. It is well-known that it is enough to show the existence of products and equalizers in **RoBan** to prove completeness. Products are defined by

$$\prod_{i \in I} F_i = \{(x_i)_{i \in I} \mid x_i \in F_i, \sup_{i \in I} \|x_i\| < \infty\}$$

together with the obvious pointwise linear structure and order, and the supremum norm. Note how the normed nature of the spaces involved is reflected in how products are built. Equalizers are computed as follows ([Min 1983, 5.2]). Given $f, g : E \rightrightarrows F$ let

$$D^+ = \{x \in E^+ \mid f(x) = g(x)\}, \qquad D := D^+ - D^+$$

where $D^+ - D^+$ is the vector space of formal differences of vectors in D^+. By construction, D is an ordered vector space with D^+ as its positive cone, and it can be equipped with the *regular norm*

$$\|x\|_r = \inf \left\{\|y\| : -y \leq x \leq y, y \in D^+\right\} \tag{4}$$

D equipped with the norm $\|\cdot\|_r$ is a regular ordered Banach space and the equalizer of $f, g : E \rightrightarrows F$.

4 A HIGHER-ORDER PROBABILISTIC LANGUAGE WITH CONDITIONING

4.1 Type System

We start by defining a type system for our language. Our aims are to (a) have enough types to write some realistic programs for example including multivariate normal or chi-squared distributions, (b) have higher-order types, (c) provide special types for Bayesian learning: *Bayesian types*.

4.1.1 Grammar. Our type system is given by the following grammar:

$$\mathtt{T} ::= m^n \mid \mathtt{int}^n \mid \mathtt{real}^n \mid \mathtt{PosDef}(n) \mid (\mathtt{T}, \mu) \mid !\mathtt{T} \mid \mathtt{T} \otimes \mathtt{T} \mid \mathtt{T} \to \mathtt{T} \mid \mathtt{MT} \tag{5}$$

where $1 \leq m, n \in \mathbb{N}$ and $\mu : \mathtt{T}$ is a term. We will refer to m, $\mathtt{int}$, $\mathtt{real}$, $\mathtt{PosDef}(n)$ as *ground types*. As their name suggest they are to be regarded as the types of (possibly random) vectors of elements of a finite set, vectors of integers, vectors of reals and $n \times n$ positive semi-definite matrices, that is to say *covariance matrices*. We will write $\mathtt{int}$ for $\mathtt{int}^1$ and $\mathtt{real}$ for $\mathtt{real}^1$. This is by no means an exhaustive set of ground types, but sufficiently rich to consider some realistic probabilistic programs. The type $1 \in \mathbb{N}$ will be referred to as the *unit type* and denoted $\mathtt{unit}$ and the type $2 \in \mathbb{N}$ will be referred to as the boolean type and denoted $\mathtt{bool}$.

The type constructors are the following. First, given a term μ of type $\mathtt{T}$, we can build the pointed type $(\mathtt{T}, \mu)$. We will call these types *Bayesian types* because the term $\mu : \mathtt{T}$ will be interpreted as a *prior*. Bayesian types will support conditioning and thus Bayesian learning. As we shall see, our Bayesian types also fulfil a role in the semantics of variable assignment. As is the tacit practice in Anglican, we will consider that assigning a (possibly random) value to a variable is equivalent to

assigning a prior to the type of this variable. For example, the program $\mathtt{x} := 2.5$ which assigns the value 2.5 to the variable $\mathtt{x}$ can be understood as placing a (deterministic) prior on the reals, namely $\delta_{2.5}$. Similarly, the program $\mathtt{x} := \mathtt{sample}(\mathtt{normal}(0,1))$ which assigns to $\mathtt{x}$ a value randomly sampled from the normal distribution $\mathcal{N}(0,1)$ with mean 0 and standard deviation 1 can be understood as setting the prior $\mathcal{N}(0,1)$ on the reals. In a slogan:

Bayesian type = type + assignment

However this slogan is only valid for assignments *without free variables*: a prior cannot be parametric in some variables, it represents definite information. This caveat will be reflected in the type system.

We then have two binary type constructors, *tensor types* and *functions types*, which will support higher-order reasoning. Finally, the unary type constructor $\mathtt{M}$ defines higher-order measures.

4.1.2 Assignable, Order-Complete and Measure Types. We now isolate three sub-grammars of types which we call *assignable, order-complete* and *measure types* respectively. These families of types will be essential to correctly type and interpret assignments (with assignable types), conditionals and while loops (with order-complete types), conditioning and sampling (with measure types). First we define *assignable types* as the types generated by the grammar

$$\begin{aligned} &\mathtt{S} ::= \mathtt{G} \mid (\mathtt{G},\mu) \mid (\mathtt{G},\mu) \to (\mathtt{G},\mu) \qquad \mathtt{G} \text{ in ground types} \\ &\mathtt{T} ::= \mathtt{S} \mid !\mathtt{S} \mid \mathtt{MS} \mid \mathtt{S} \otimes \mathtt{S} \end{aligned} \tag{6}$$

Second, we define *order-complete types* as the types generated by the 'dual' grammar

$$\begin{aligned} &\mathtt{S} ::= \mathtt{G} \mid (\mathtt{G},\mu) \mid (\mathtt{G},\mu) \otimes (\mathtt{G},\mu) \qquad \mathtt{G} \text{ in ground types} \\ &\mathtt{T} ::= \mathtt{S} \mid !\mathtt{S} \mid \mathtt{MS} \mid \mathtt{S} \to \mathtt{S} \end{aligned} \tag{7}$$

Finally, *measure types* are the types generated by all the constructors of grammar (5) apart from ! and function types.

4.1.3 Contexts. are maps $\Gamma : \mathbb{N} \to \mathtt{Types}$ – the free algebra of all types generated by (5) – which send cofinitely many integers to the unit type $\mathtt{unit}$. We will write $\mathrm{supp}(\Gamma)$ for the set $\{i \mid \Gamma(i) \neq \mathtt{unit}\}$ and use the traditional notation $\Gamma[i \mapsto \mathtt{T}]$ to denote the context mapping i to $\mathtt{T}$ and all $j \neq i$ to $\Gamma(j)$. The image of $0 \in \mathbb{N}$ under a context Γ will, by convention, fulfil the role of *output*. If a computation e returns a value, then this value will be passed along the 'channel' 0. Formally, the sequent stating that a computation e returns a value of type $\mathtt{T}$ in a context Γ, will look like:

$$\Gamma \vdash e : [0 \mapsto \mathtt{T}]$$

with $0 \notin \mathrm{supp}(\Gamma)$ (since the 0 channel is reserved for outputs). Memory-manipulating operations like assignments will create non-output contexts on the right-hand side of sequents.

Our contexts are a dynamic version of the static contexts of [Kozen 1981] which consists of a constant map from $\mathbb{N}$ to a single type. They are in some respects similar to the heap models of separation logic, and for notational clarity we will require similar operations on contexts as on heaps: a notion of compatibility, of union and of difference. Given two contexts Γ_1, Γ_2 we will say that they are *compatible* if $\Gamma_1(i) = \Gamma_2(i)$ for all $i \in \mathrm{supp}(\Gamma_1) \cap \mathrm{supp}(\Gamma_2)$, and we will then write $\Gamma_1 \Downarrow \Gamma_2$. For any two compatible contexts $\Gamma_1 \Downarrow \Gamma_2$ we define the *union context* $\Gamma_1 \oplus \Gamma_2$ as the union of their graphs, which is a function by the compatibility assumption. We define the *difference context* $\Gamma_1 \ominus \Gamma_2$ as the map sending $i \mapsto \Gamma_1(i)$ if $i \notin \mathrm{supp}(\Gamma_2)$ and to $\mathtt{unit}$ otherwise. In particular $\Gamma_1 \ominus \Gamma_2 = \Gamma_1$ if the supports are disjoint. Finally, we write $\Gamma_1 \pitchfork \Gamma_2$ if $\mathrm{supp}(\Gamma_1) \cap \mathrm{supp}(\Gamma_2) = \emptyset$.

4.2 A Higher-Order Probabilistic Language

We now define the terms of our imperative probabilistic higher-order language.

4.2.1 Expressions. Terms are built according to the grammar of expressions given in Fig 2.

$$
\begin{array}{lr}
e ::= i \in m^k \mid n \in \mathbb{N}^k \mid r \in \mathbb{R}^k \mid m \in \mathrm{PosDef}(n) \mid & \text{Constants} \\
\quad \texttt{skip} \mid \texttt{op}(e, \ldots, e) \mid & \text{Built-in operations} \\
\quad \texttt{x\_}i \mid & 0 < i \in \mathbb{N},\ \text{Variables} \\
\quad \texttt{x\_}i := e \mid & \text{Assignment} \\
\quad \texttt{return(e)} \mid & \text{Return instruction} \\
\quad e; e \mid & \text{Sequential composition} \\
\quad \texttt{fn x\_}i\ .\ e \mid & \lambda\text{-abstraction} \\
\quad e(e) \mid & \text{Function application} \\
\quad \texttt{if}\ \ e\ \ \texttt{then}\ \ e\ \ \texttt{else}\ \ e \mid & \text{Conditional} \\
\quad \texttt{while}\ \ e\ \ \texttt{do}\ \ e \mid & \text{Iterations} \\
\quad \texttt{sample}(e) \mid & \text{Sampling} \\
\quad \texttt{sampler}(e) \mid & \text{Packages a program as a sampler} \\
\quad \texttt{observe}(e) & \text{Conditioning}
\end{array}
$$

Fig. 2. Expressions

Every built-in operation must come equipped with typing instruction which we will write as an $n+1$-tuple $(\mathsf{S}_1, \ldots, \mathsf{S}_n, \mathsf{T})$ where the first n components are ground types specifying the input types and the last component is a ground type or measures over a ground type specifying the output type. For example the boolean connective `or` would come with typing $(\texttt{bool}, \texttt{bool}, \texttt{bool})$, the sine function `sin` with typing $(\texttt{real}, \texttt{real})$ and the function `normal` constructing a normal distribution would come with typing $(\texttt{real}, \texttt{PosDef}(1), \mathsf{M}\ \texttt{real})$, where the first input is the mean, the second is the standard deviation and the output is a measure over the reals.

4.2.2 Well-Typed Expressions. The typing rules for our language are gathered in Fig. 3. We will discuss these rules in detail when we define the denotational semantics of our language in § 5, but we can already make some observations. The reader will notice that there are two types of rules: (1) the rules for computations which produce an output, which will conventionally be returned in a context of the shape $[0 \mapsto \mathsf{T}]$ on the right of the turnstile, and (2) rules for manipulating an internal store (like assignments, sequential compositions thereof, loops, etc.) which can have arbitrarily large contexts to the right of the turnstile. Since our variables are labelled $\texttt{x\_}i$ for $i > 0$, it is impossible to introduce a context on the left of the turnstile with 0 in its support.

The sequential composition rule looks daunting, but it is simply a version of the cut rule with the necessary bookkeeping to make sure contexts do not conflict with one another. The first side condition says that e_2 cannot rely on the part of the context Γ_1 which was consumed to produce the output of e_1, the second side condition says that the output context of e_1 must be compatible with the input context of e_2. Finally, the third side condition says that we can construct an output context for $e_1; e_2$ by combining output context of e_2 with the part of the output context of e_1 which was not consumed by e_2.

Notice that the only way to create a Bayesian type is through a variable assignment without free variables: a prior must contain definite information, not information which is parametric in variables. Only assignable types can form Bayesian types. Note finally that our `observe` statement applies to a term of type `T`, intuitively we observe a possibly random element of type `T`. This is slightly different from the syntax of `observe` in Anglican where a *distribution* is observed. Semantically, the difference disappears since a possibly random element is modelled by a distribution.

Constants

$$\frac{}{\emptyset \vdash i : [0 \mapsto n]}\ i \in n \qquad \frac{}{\emptyset \vdash n : [0 \mapsto \mathtt{int}]}\ n \in \mathbb{N}$$

$$\frac{}{\emptyset \vdash r : [0 \mapsto \mathtt{real}]}\ r \in \mathbb{R} \qquad \frac{}{\emptyset \vdash M : [0 \mapsto \mathtt{PosDef}(n)]}\ M \in \mathrm{PosDef}(n)$$

Variables, Bayesian types, Tensor rule

$$\frac{}{[i \mapsto \mathsf{T}] \vdash \mathtt{x\_}i : [0 \mapsto \mathsf{T}]} \qquad \frac{\emptyset \vdash e : [0 \mapsto \mathsf{T}]}{\emptyset \vdash e : [0 \mapsto (\mathsf{T}, e)]}\ \mathsf{T} \text{ measure type}$$

$$\frac{}{\Gamma \vdash \mathtt{skip} : \Gamma} \qquad \frac{\Gamma_1 \vdash e_1 : \Gamma_2}{\Gamma_1 \oplus \Delta \vdash e_1 : \Gamma_2 \oplus \Delta}\ \Gamma_1 \pitchfork \Delta, \Gamma_2 \pitchfork \Delta$$

Exponential

$$\frac{\Gamma[i \mapsto !\mathsf{S}, j \mapsto !\mathsf{S}] \vdash e : \Delta}{\Gamma[i \mapsto !\mathsf{S}] \vdash e : \Delta}\ i \neq j \qquad \frac{\Gamma[i \mapsto \mathsf{S}] \vdash e : \Delta}{\Gamma[i \mapsto !\mathsf{S}] \vdash e : \Delta}$$

Built-in operations

$$\frac{\Gamma_1 \vdash e_1 : [0 \mapsto \mathsf{S}_1] \qquad \cdots \qquad \Gamma_n \vdash e_n : [0 \mapsto \mathsf{S}_n]}{\Gamma_1, \ldots, \Gamma_n \vdash \mathtt{op}(e_1, \ldots, e_n) : [0 \mapsto \mathsf{T}]}\ \mathtt{op} : (\mathsf{S}_1, \ldots, \mathsf{S}_n, \mathsf{T}), \Gamma_i \pitchfork \Gamma_j, i \neq j$$

Assignment, Return

$$\frac{\Gamma \vdash e : [0 \mapsto \mathsf{T}]}{\Gamma[i \mapsto \mathsf{T}] \vdash \mathtt{x\_}i := e : [i \mapsto \mathsf{T}]}\ \mathsf{T} \text{ assignable}$$

$$\frac{}{\Gamma[i \mapsto \mathsf{T}] \vdash \mathtt{return}(\mathtt{x\_}i) : [0 \mapsto \mathsf{T}]}\ \Gamma(j), j \neq i \text{ assignable}$$

Sequential composition

$$\frac{\Gamma_1 \vdash e_1 : \Delta_1 \qquad \Gamma_2 \vdash e_2 : \Delta_2}{\Gamma_1 \oplus (\Gamma_2 \ominus \Delta_1) \vdash e_1; e_2 : (\Delta_1 \ominus \Gamma_2) \oplus \Delta_2}\ \Gamma_2 \pitchfork (\Gamma_1 \ominus \Delta_1),\ \Delta_1 \Downarrow \Gamma_2,\ (\Delta_1 \ominus \Gamma_2) \Downarrow \Delta_2$$

λ-abstraction, application

$$\frac{\Gamma[i \mapsto S] \vdash e : [0 \mapsto \mathsf{T}]}{\Gamma \vdash \mathtt{fn\ x\_}i\ .\ e : [0 \mapsto (\mathsf{S} \to \mathsf{T})]}\ i \notin \mathrm{supp}(\Gamma)$$

$$\frac{\Gamma \vdash e_1 : [0 \mapsto \mathsf{S}] \qquad \Delta \vdash e_2 : [0 \mapsto (\mathsf{S} \to \mathsf{T})]}{\Gamma, \Delta \vdash e_2(e_1) : [0 \mapsto \mathsf{T}]}$$

Imperative control flow

$$\frac{\Gamma \vdash e_1 : [0 \mapsto \mathtt{bool}] \qquad \Gamma \vdash e_2 : \Delta \qquad \Gamma \vdash e_3 : \Delta}{\Gamma \vdash \mathtt{if}\ \ e_1\ \ \mathtt{then}\ \ e_2\ \ \mathtt{else}\ \ e_3 : \Delta}\ \Gamma \text{ order-complete type}$$

$$\frac{\Gamma \vdash e_1 : [0 \mapsto \mathtt{bool}] \qquad \Gamma \vdash e_2 : \Gamma}{\Gamma \vdash \mathtt{while}\ \ e_1\ \ \mathtt{do}\ \ e_2 : \Gamma}\ \Gamma \text{ order-complete type}$$

Probabilistic operations

$$\frac{\Gamma \vdash e : [0 \mapsto \mathsf{T}]}{\Gamma \vdash \mathtt{sampler}(e) : [0 \mapsto \mathsf{MT}]}\ \mathsf{T} \text{ measure type} \qquad \frac{\Gamma \vdash e : [0 \mapsto \mathsf{MT}]}{\Gamma \vdash \mathtt{sample}(e) : [0 \mapsto \mathsf{T}]}\ \mathsf{T} \text{ measure type}$$

$$\frac{[i \mapsto (\mathsf{S}, \mu)] \vdash e : [0 \mapsto \mathsf{T}]}{[i \mapsto (\mathsf{S}, \mu)] \vdash \mathtt{observe}(e) : [0 \mapsto ((\mathsf{T}, e[\mathtt{x\_}i/\mu]) \to (\mathsf{S}, \mu))]}\ \mathsf{S}, \mathsf{T} \text{ measure types}$$

Fig. 3. Typing rules

4.2.3 A Simple Example. It is not hard (if notationally cumbersome) to type-check the simple Gaussian inference program of Fig. 4 against the inference rules of Fig. 3. In the context $[1 \mapsto (\mathtt{real}, \mathtt{sample}(\mathtt{normal}(\mathtt{0}, \mathtt{1})))]$, it evaluates to a function of type

$$(\mathtt{real}, \mathtt{sample}(\mathtt{normal}(\mathtt{sample}(\mathtt{normal}(\mathtt{0}, \mathtt{1})), \mathtt{1}))) \to (\mathtt{real}, \mathtt{sample}(\mathtt{normal}(\mathtt{0}, \mathtt{1}))) \tag{8}$$

which, as we will see in § 5.2.9, is what we want semantically.

```
x_1=sample(normal(0,1);
observe(sample(normal(x_1,1)))
```

Fig. 4. A small Gaussian inference program

5 DENOTATIONAL SEMANTICS

As the reader will have guessed we will now provide a denotational semantics for the language described in § 4 in the category **RoBan**: types will be denoted by regular ordered Banach spaces, and programs by regular (and thus continuous) operators between these spaces.

5.1 Semantics of Types

5.1.1 Ground Types. We define:

- $[\![m^k]\!] = \mathcal{M}\left(\{1,\dots,m\}^k\right)$ where $\{1,\dots,m\}^k$ is equipped with the discrete σ-algebra. Note that $[\![m]\!] \simeq \mathbb{R}^m$, and thus $[\![\mathtt{unit}]\!] \simeq \mathbb{R}$, the unit of the positive projective tensor.
- $[\![\mathtt{int}^k]\!] = \mathcal{M}\left(\mathbb{N}^k\right)$, where $\mathbb{N}^k$ is equipped with the discrete σ-algebra
- $[\![\mathtt{real}^k]\!] = \mathcal{M}\left(\mathbb{R}^k\right)$, where $\mathbb{R}^k$ is equipped with its usual Borel σ-algebra
- $[\![\mathtt{PosDef}(n)]\!] = \mathcal{M}\mathrm{PosDef}(n)$, where $\mathrm{PosDef}(n)$ is the space of positive semi-definite $n \times n$ matrices equipped with the Borel σ-algebra inherited from $\mathbb{R}^{n\times n}$

5.1.2 Elementary Type Constructors. As expected, the tensor and function type constructors are interpreted by the monoidal closed structure of **RoBan**, i.e.

$$[\![\mathtt{S} \otimes \mathtt{T}]\!] := [\![\mathtt{S}]\!] \mathbin{\widehat{\otimes}_{|\pi|}} [\![\mathtt{T}]\!] \qquad\qquad [\![\mathtt{S} \to \mathtt{T}]\!] := \left[[\![\mathtt{S}]\!], [\![\mathtt{T}]\!]\right]$$

The higher-order probability type constructor $\mathtt{M}$ is interpreted as follows. For any regular ordered Banach space V we consider the underlying set together with the Borel σ-algebra induced by the norm. We then apply the functor $\mathcal{M}$ to this measurable space. This construction is functorial and we overload $\mathcal{M}$ to denote the resulting regular ordered Banach space by $\mathcal{M}V$. Using this convenient notation we define

$$[\![\mathtt{M}\mathtt{T}]\!] := \mathcal{M}[\![\mathtt{T}]\!]$$

5.1.3 Bayesian Types. The type system in Fig. 3 can only produce a Bayesian type $(\mathtt{T}, \mu)$ if $\mathtt{T}$ is a measure type and μ has no free variables, i.e. if $\emptyset \vdash \mu : [0 \mapsto \mathtt{T}]$ is derivable. We will therefore only need to provide a semantics to Bayesian types of this shape. Our semantics of Bayesian types is in some respect similar to that of *pointed types* used in homotopy type theory [Licata and Finster 2014]. Indeed, at the type-theoretic level they are defined identically as a type together with a term inhabiting this type. However, the ordered vector space structure allows us to provide a semantics which is much richer than a space with a distinguished point. Given a measure type $\mathtt{T}$ and a sequent of the type $\emptyset \vdash \mu : [0 \mapsto \mathtt{T}]$, we will see in § 5.2 that μ is interpreted as an operator $[\![\mu]\!] : \mathbb{R} \to [\![\mathtt{T}]\!]$, which is uniquely determined by $[\![\mu]\!](1)$. For notational clarity we will often simply write μ for $[\![\mu]\!](1)$. We define the denotation of the Bayesian type $(\mathtt{T}, \mu)$ as the *principal band in* $[\![\mathtt{T}]\!]$ *(see § 3.1.3) generated by the measure* μ (i.e. $[\![\mu]\!](1)$). Formally:

$$[\![(\mathtt{T}, \mu)]\!] = [\![\mathtt{T}]\!]_\mu \tag{9}$$

For this semantics to be well-defined it is necessary that $[\![\mathtt{T}]\!]$ be at least a Riesz space, since bands are defined using the lattice structure. This is indeed the case:

Theorem 12. *The semantics of a measure type is a Banach lattice.*

5.1.4 Semantics of the Exponential. We follow the three steps of [Melliès et al. 2018] to compute the exponential objects in the category **RoBan** of regular ordered Banach spaces and regular operators. See also [Crubillé et al. 2017] for an application of the same construction in a similar context.

Step 1. Since **RoBan** has products, every object E generates a free pointed object $E_\bullet$ in the category $\mathbf{RoBan}_\bullet$ of pointed regular ordered Banach spaces. It can be described explicitly as the product

$$E_\bullet = E \times \mathbb{R} \qquad \text{together with} \qquad \varepsilon_E = \pi_1 : E_\bullet = E \times \mathbb{R} \to E.$$

Step 2. The second step consists in computing the equalizers (in **RoBan**, see § 3.3.5)

$$E^{\leq n} \xrightarrow{\quad e_n \quad} E_\bullet^{\otimes n} \underset{\sigma_{n!}}{\overset{\sigma_1}{\rightrightarrows}} E_\bullet^{\otimes n} \tag{10}$$

where $E_\bullet^{\otimes n}$ is short-hand for the n-fold positive projective tensor of $E_\bullet$ with itself, and $\sigma_1, \ldots, \sigma_{!n}$ are the elements of the group $\mathrm{Perm}(n)$ of permutations on n elements acting on n-tensors as follows:

$$\sigma\left((u_1, \lambda_1) \otimes \ldots \otimes (u_n, \lambda_n)\right) = \left((u_{\sigma(1)}, \lambda_{\sigma(1)}) \otimes \ldots \otimes (u_{\sigma(n)}, \lambda_{\sigma(n)})\right)$$

$u_i \in E, \lambda_i \in \mathbb{R}, 1 \leq i \leq n$. Since tensor products distribute over finite products (up to isomorphism) and $E \otimes \mathbb{R} \simeq E$ we have

$$E_\bullet^{\otimes n} = (E \times \mathbb{R})^{\otimes n} \simeq \prod_{i=0}^{n} \binom{n}{i} E^{\otimes i}$$

with $E^{\otimes 0} := \mathbb{R}$. In this sense $E_\bullet^{\otimes n}$ contains all tensors of rank $0 \leq i \leq n$, but it is simpler in what follows to simply view basic elements of $E^{\leq n}$ as symmetric tensors in $(E \times \mathbb{R})^{\otimes n}$. The unit $u : E_\bullet = E \times \mathbb{R} \to \mathbb{R}$ given by the second projection defines morphisms

$$\partial : E_\bullet^{\otimes n+1} \to E_\bullet^{\otimes n} : (u_1, \lambda_1) \otimes \ldots \otimes (u_n, \lambda_n) \otimes (u_{n+1}, \lambda_{n+1}) \mapsto (u_1, \lambda_1) \otimes \ldots \otimes (u_n, \lambda_n) \otimes \lambda_{n+1} \tag{11}$$

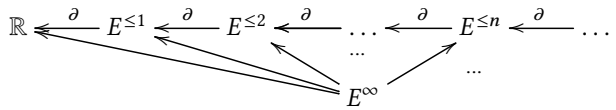
where the last tensor can be replaced by a scalar multiplication. By the universal property of equalizers, and the fact that (11) sends symmetric tensors to symmetric tensors we get a map

$$\partial : E^{\leq n+1} \to E^{\leq n}$$

acting in exactly the same way as (11). The following technical condition must be satisfied for the construction of [Melliès et al. 2018] to work. The proof relies on the linear structure of objects in **RoBan** which allows the construction of a symmetrization map $\mathfrak{S} : E_\bullet^{\otimes n} \to E^{\leq n}$.

Proposition 13. *The positive projective tensor product* $\widehat{\otimes}_{|\pi|}$ *commutes with the equalizers of* (10).

Step 3: The last step of the construction is to consider the projective limit E^∞ of the diagram



(12)

It follows from the construction of products and equalizers in **RoBan** that the projective limit E^∞ is completely determined by the positive cone

$$(E^\infty)^+ = \{(t_1, t_2, \ldots, t_n, \ldots) \mid t_n \in \left(E^{\leq n}\right)^+, \ \sup_n \|t_n\| < \infty, \ \partial(t_{n+1}) = t_n\}$$

which characterises the order on E^∞, generates the entire space via $E^\infty = (E^\infty)^+ - (E^\infty)^+$, and defines the normed structure via the regular norm (4). The condition that $\sup_n \|t_n\| < \infty$ is essential, and only occurs because we are taking limits in a category of normed spaces.

We can characterise precisely the elements $(t_n)_{n\in\mathbb{N}} \in E^\infty$. For notational clarity we now denote tensors using concatenation, i.e. $(u_1,\lambda_1)\otimes(u_2,\lambda_2) := (u_1,\lambda_1)(u_2,\lambda_2)$ in analogy with words over an alphabet. We start with the following lemma examining projections on the 2^{nd} approximant $E^{\leq 2}$.

Lemma 14. *If t_2 is of the form*

$$t_2 = c_0(a,\lambda_a)(a,\lambda_a) + c_1(a,\lambda_a)(b,\lambda_b) + c_2(b,\lambda_b)(a,\lambda_a) + c_3(b,\lambda_b)(b,\lambda_b) \tag{13}$$

then $c_1 = c_2$ and $t_n, n \geq 2$ is a linear combination of tensor products of (a,λ_a) and (b,λ_b) only.

Let $\mathbb{R}[a_1,\ldots,a_k]$ denote the graded noncommutative ring of homogeneous polynomials over k noncommuting variables $a_1,\ldots,a_k$ with real coefficients. Let $\mathbb{R}[\mathbb{N}^k]$ denote its commutative image; that is, its image under the map $\theta : \mathbb{R}[a_1,\ldots,a_k] \to \mathbb{R}[\mathbb{N}^k]$ defined by

$$\theta(\mathbf{m}) = (\#a_1(\mathbf{m}),\ldots,\#a_k(\mathbf{m})) \qquad\qquad \theta\left(\sum_{\mathbf{m}} c_{\mathbf{m}}\mathbf{m}\right) = \sum_{\mathbf{m}} c_{\mathbf{m}}\theta(\mathbf{m}),$$

where $\mathbf{m}$ denotes a monomial and $\#a_i(\mathbf{m})$ denotes the number of occurrences of a_i in $\mathbf{m}$.

Theorem 15. *If t_2 is of the shape (13), then t_n is of the shape*

$$t_n = \sum_{k=0}^{n} c_k^{(n)} \sum \theta^{-1}(k, n-k) \tag{14}$$

where $\theta : \mathbb{R}[(a,\lambda_a),(b,\lambda_b)] \to \mathbb{R}[\mathbb{N}^2]$. Moreover,

$$c_k^{(n-1)} = c_{k+1}^{(n)}\lambda_a + c_k^{(n)}\lambda_b \quad (0 \leq k \leq n-1). \tag{15}$$

Corollary 16. *Under the assumptions of Theorem 15, if $\lambda_a = \lambda_b = 0$ then $c_k^{(n)} = 0$ for each n and $0 \leq k \leq n$. If $\lambda_a \neq 0$ then the coefficients $c_k^{(n)}$ satisfy:*

$$c_k^{(n)} = \sum_{i=0}^{k} \binom{k}{i} c_0^{(n-k+i)} (-\lambda_b)^i \lambda_a^{-k} \tag{16}$$

If follows from Corollary (16) that given an element of t_2 of the shape (13) with either $\lambda_a \neq 0$ or $\lambda_b \neq 0$, each t_n is uniquely defined by the choice of a single real number $c_0^{(n)}$ at each level n. With this characterisation of the elements of E^∞ we can prove the final condition which must be met in the construction of [Melliès et al. 2018]. The proof of the following result relies heavily on the fact that objects in **RoBan** are *complete* spaces.

Theorem 17. *The positive projective tensor product $\widehat{\otimes}_{|\pi|}$ commutes with the projective limits (12).*

This completes the proof of existence, and the description of the exponential object. It allows us to define $[\![!\mathsf{T}]\!] := ![\![\mathsf{T}]\!] := [\![\mathsf{T}]\!]^\infty$.

5.1.5 Order-Complete Types. As shown in [Fremlin 1974, 4C] the product $\mathrm{L}_2([0,1]) \,\widehat{\otimes}_{|\pi|}\, \mathrm{L}_2([0,1])$ is not order-complete, even though $\mathrm{L}_2([0,1])$ is. It follows that order-completeness does not propagate through the grammar (5), even though the denotation of every ground type is order-complete. Since order-completeness will be crucial in defining the semantics of conditionals and `while` loops, it is important to identify a sub-grammar of types which are guaranteed to be order complete. This is the purpose of the order-complete types given by the grammar (7).

Theorem 18. *The semantics of an order-complete type is an order-complete space.*

5.1.6 Contexts. A context Γ will be interpreted as the positive projective tensor

$$[\![\Gamma]\!] = \bigotimes_{i \in \mathrm{supp}(\Gamma)}^{\widehat{}}{}_{|\pi|} [\![\Gamma(i)]\!].$$

and we put $[\![\emptyset]\!] := \mathbb{R}$. A typing rule $\Gamma \vdash e : \Delta$ will be interpreted as a regular (in fact positive, see Th. 26) operator $[\![e]\!] : [\![\Gamma]\!] \to [\![\Delta]\!]$.

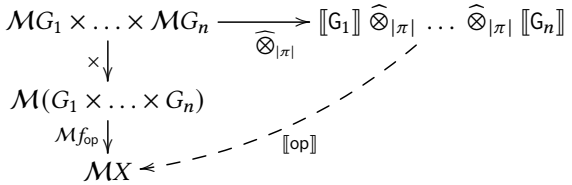
5.2 Semantics of well-formed expressions

Let us now turn to the semantics of terms.

5.2.1 Constants. A constant $c \in G$ whose ground type $\mathtt{G}$ is interpreted as the space $\mathcal{M}G$ will be interpreted as the operator

$$[\![c]\!] : [\![\emptyset]\!] = \mathbb{R} \longrightarrow [\![\mathtt{G}]\!] = \mathcal{M}G, \quad \lambda \mapsto \lambda \delta_c$$

5.2.2 `skip` *and Built-in Operations.* The denotation of $\Gamma \vdash \mathtt{skip} : \Gamma$ is $[\![\mathtt{skip}]\!] := \mathrm{Id}_{[\![\Gamma]\!]}$. Recall that every built-in operation `op` comes with typing information $(\mathtt{G}_1, \ldots, \mathtt{G}_n, \mathtt{T})$ where each $\mathtt{G}_i$ is of ground type and $\mathtt{T}$ is either of ground type or of type `MG`, for `G` a ground type. Each built-in operation is interpreted via a function $f_{\mathsf{op}} : G_1 \times \ldots \times G_n \to X$, with $X = G$ or $X = \mathcal{M}G$, as the unique regular operator which linearizes $\mathcal{M}f_{\mathsf{op}} \circ \times$ according to the universal property (3) of $\widehat{\otimes}_{|\pi|}$:



For example the boolean operator `or` of type $(\mathtt{bool}, \mathtt{bool}, \mathtt{bool})$ would be interpreted, via the function $f_{\mathsf{or}} : 2 \times 2 \to 2$ implementing the boolean join, as the linearisation of $\mathcal{M}f_{\mathsf{or}} \circ \times$ (which is bilinear). Similarly, the operation `normal` of type $(\mathtt{real}, \mathtt{PosDef}(1), \mathtt{Mreal})$ building a normal distributions would be interpreted, via the obvious function $f_{\mathtt{normal}} : \mathbb{R} \times \mathbb{R}^+ \to \mathcal{M}\mathbb{R}$, as the linearisation of $\mathcal{M}f_{\mathtt{normal}} \circ \times$. If the input is deterministic, i.e. a tensor $\delta_\mu \otimes \delta_\sigma$ for a mean $\mu \in \mathbb{R}$ and a standard deviation $\sigma \in \mathbb{R}^+$ (as would typically be the case), then $[\![\mathtt{normal}]\!](\delta_\mu \otimes \delta_\sigma)$ outputs *a Dirac delta* over the distribution $\mathcal{N}(\mu, \sigma)$. Note how we interpret the deterministic construction of a distribution over X differently from sampling an element of X according to this distribution: the former is a distribution over distributions, the latter just a distribution.

5.2.3 Variables, Assignments and Return. A variable on its own acts like a variable declaration and introduces a context (see Fig. 3). Its semantics is simply given by the identity operator on the type of the variable, formally the sequent $[i \mapsto \mathtt{T}] \vdash \mathtt{x\_}i : [0 \mapsto \mathtt{T}]$ is interpreted by $[\![\mathtt{x\_}i]\!] = \mathrm{Id}_{[\![\mathtt{T}]\!]}$ (where we use the fact that the denotation of the contexts $[i \mapsto \mathtt{T}]$ and $[0 \mapsto \mathtt{T}]$ are both $[\![\mathtt{T}]\!]$).

The idea behind our semantics of assignment is to generalize the semantics of [Kozen 1981] where an assignment $\mathtt{x\_}i := x$ to a variable of type, say, `real` is interpreted as $\mathcal{M}c_x : \mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}$ where $c_x : \mathbb{R} \to \mathbb{R}, y \mapsto x$ is the constant function to x. This interpretation sends any input measure μ to $ev_{\mathbb{R}}(\mu)\delta_x$, i.e. it coerces any input measure into the Dirac δ over x – which is what one would expected of a deterministic assignment – *up to a scalar factor* given by the total mass of the input measure $ev_{\mathbb{R}}(\mu)$. This last point is essential to make the semantics of assignment linear. In order to generalise this semantics to more types and to random assignments we need to formalise the role of the functional $ev_{\mathbb{R}} : \mathcal{M}\mathbb{R} \to \mathbb{R}$.

Theorem 19. *The denotation of any assignable type* T *admits a strictly positive functional* ϕ_{T} *with norm* $\|\phi_{\mathsf{T}}\| \leq 1$ *built inductively from the evaluation functionals* ev_G*, where* $\mathcal{M}G = [\![\mathsf{G}]\!]$ *ranges over ground types.*

The strictly positive functional ϕ_{T} built inductively by Theorem 19 can be thought of as a generalisation to all assignable types of the 'total mass' functional on spaces of measures. With this notion in place we can provide a semantics to assignments. Given a sequent $\Gamma \vdash e : [0 \mapsto \mathsf{T}]$, let ϕ_{T} be the strictly positive functional on $[\![\mathsf{T}]\!]$ constructed in Th. 19 and let us write $[\![\Gamma[i \mapsto \mathsf{T}]]\!]$ as $[\![\Gamma_1]\!] \mathbin{\widehat{\otimes}_{|\pi|}} [\![\mathsf{T}]\!] \mathbin{\widehat{\otimes}_{|\pi|}} [\![\Gamma_2]\!]$. We now define the multilinear map

$$[\![\Gamma_1]\!] \times [\![\mathsf{T}]\!] \times [\![\Gamma_2]\!] \longrightarrow [\![\mathsf{T}]\!] \qquad (\gamma_1, t, \gamma_2) \mapsto \begin{cases} \phi_{\mathsf{T}}(t)[\![e]\!](\gamma_1 \otimes t \otimes \gamma_2) & \text{if } \Gamma(i) = \mathsf{T} \\ \phi_{\mathsf{T}}(t)[\![e]\!](\gamma_1 \otimes \gamma_2) & \text{else} \end{cases}$$

This defines the unique linearizing operator[3]

$$[\![\mathtt{x\_}i := e]\!] : [\![\Gamma[i \mapsto \mathsf{T}]]\!] \longrightarrow [\![[i \mapsto \mathsf{T}]]\!]$$

As a simple example, it is easy to type-check the program $\mathtt{x\_}i := 3.5$ and see by unravelling the definition that it is interpreted as the operator $[\![\mathtt{x\_}i := 3.5]\!] : \mathcal{M}\mathbb{R} \to (\mathcal{M}\mathbb{R})_{\delta_{3.5}}, \mu \mapsto \mu(\mathbb{R})\delta_{3.5}$. In particular any probability distribution gets mapped to $\delta_{3.5}$. We thus recover the semantics of assignment of [Kozen 1981]. The type system of Fig. 3 also allows us to interpret $\mathtt{x\_}i := 3.5$ more finely as an assignment to a Bayesian type interpreted as the restriction of $[\![\mathtt{x\_}i := 3.5]\!]$ to the one-dimensional band $[\![(\mathtt{real}, 3.5)]\!] = \mathcal{M}\mathbb{R}_{\delta_{3.5}}$.

The denotation of the `return` rule exploits the fact that the functionals described in Th. 19 can be used as deletion operators (since they are of type $\mathsf{T} \to \mathtt{unit}$ with $[\![\mathtt{unit}]\!] = \mathbb{R}$). Writing again $[\![\Gamma[i \mapsto \mathsf{T}]]\!] = [\![\Gamma_1]\!] \mathbin{\widehat{\otimes}_{|\pi|}} [\![\mathsf{T}]\!] \mathbin{\widehat{\otimes}_{|\pi|}} [\![\Gamma_2]\!]$ we define $[\![\mathtt{return}(\mathtt{x\_}i)]\!] : [\![\Gamma[i \mapsto \mathsf{T}]]\!] \to [\![\mathsf{T}]\!]$

$$[\![\mathtt{return}(\mathtt{x\_}i)]\!] : \left(\bigotimes_{i \in \mathrm{supp}(\Gamma_1)} \phi_{\Gamma_1(i)} \otimes \mathrm{Id}_{[\![\mathsf{T}]\!]} \otimes \bigotimes_{i \in \mathrm{supp}(\Gamma_2)} \phi_{\Gamma_2(i)} \right)$$

as the operator which deletes (up to a scalar factor in order to remain linear) all registers apart from the one corresponding to $[i \mapsto \mathsf{T}]$ which is returned on the 'output channel' 0.

Remark 20. We do not know if it is possible to define a strictly positive functional on an arbitrary space of operators $[U, V]$ given strictly positive linear functionals on U and V. It is easy, using the Hahn-Banach theorem to find *positive* functionals of bounded norm, but they might vanish for some positive operators in $[U, V]$.

5.2.4 Exponential. The denotation of the exponential type constructor ensures that $[\![!\mathsf{T}]\!]$ always comes equipped with a comonoid and a comonad structure. In particular this provides morphisms

$$[\![!\mathsf{T}]\!] \to [\![!\mathsf{T}]\!] \otimes [\![!\mathsf{T}]\!] \qquad [\![!\mathsf{T}]\!] \to [\![\mathsf{T}]\!]$$

through which the exponential's rules of Fig. 3 are interpreted in the usual way [Melliès et al. 2018].

5.2.5 Sequential Composition. The semantics of sequential composition boils down to function composition and tensoring. The only difficulty lies in the bookkeeping of contexts. Given

$$[\![e_1]\!] : [\![\Gamma_1]\!] \to [\![\Delta_1]\!] \qquad \text{and} \qquad [\![e_2]\!] : [\![\Gamma_2]\!] \to [\![\Delta_2]\!], \tag{17}$$

the side conditions ensure that we can build the join context

$$[\![\Gamma_1 \oplus (\Gamma_2 \ominus \Delta_1)]\!] = \bigotimes_{i \in \mathrm{supp}(\Gamma_1) \cup \mathrm{supp}(\Gamma_2 \ominus \Delta_1)}^{\widehat{}}{}_{|\pi|} [\![\Gamma(i)]\!]$$

[3] In fact a *nuclear operator* [Abramsky et al. 1999].

where $\Gamma(i) = \Gamma_1(i)$ if $i \in \mathrm{supp}(\Gamma_1)$ and $\Gamma_2(i)$ else. Since **RoBan** is *symmetric* monoidal, there exists a linear operator permuting the factors in the context $[\![\Gamma_1 \oplus (\Gamma_2 \ominus \Delta_1)]\!]$ so that factors in Γ_1 appear first:

$$T_1 : [\![\Gamma_1 \oplus (\Gamma_2 \ominus \Delta_1)]\!] \to \left(\bigotimes_{i \in \mathrm{supp}(\Gamma_1)}^{\widehat{\otimes}_{|\pi|}} \Gamma_1(i)\right) \widehat{\otimes}_{|\pi|} \left(\bigotimes_{i \in \mathrm{supp}(\Gamma_2)\setminus\mathrm{supp}(\Gamma_1)}^{\widehat{\otimes}_{|\pi|}} \Gamma_2(i)\right)$$

The first side condition of the sequential composition rule in Fig. 3 ensures that $\mathrm{supp}(\Gamma_2)\setminus\mathrm{supp}(\Gamma_1) = \mathrm{supp}(\Gamma_2) \setminus \mathrm{supp}(\Delta_1)$. The second side condition guarantees that there exists a permutation operator T_2:

$$T_2 : \left(\bigotimes_{i \in \mathrm{supp}(\Delta_1)}^{\widehat{\otimes}_{|\pi|}} \Delta_1(i)\right) \widehat{\otimes}_{|\pi|} \left(\bigotimes_{i \in \mathrm{supp}(\Gamma_2)\setminus\mathrm{supp}(\Delta_1)}^{\widehat{\otimes}_{|\pi|}} \Gamma_2(i)\right) \to \left(\bigotimes_{i \in \mathrm{supp}(\Delta_1)\setminus\mathrm{supp}(\Gamma_2)}^{\widehat{\otimes}_{|\pi|}} \Delta_1(i)\right) \widehat{\otimes}_{|\pi|} \left(\bigotimes_{i \in \mathrm{supp}(\Gamma_2)}^{\widehat{\otimes}_{|\pi|}} \Gamma_2(i)\right)$$

Finally, there is a permutation operator T_3

$$T_3 : \left(\bigotimes_{i \in \mathrm{supp}(\Delta_1)\setminus\mathrm{supp}(\Gamma_2)}^{\widehat{\otimes}_{|\pi|}} \Delta_1(i)\right) \widehat{\otimes}_{|\pi|} \left(\bigotimes_{i \in \mathrm{supp}(\Delta_2)}^{\widehat{\otimes}_{|\pi|}} \Delta_2(i)\right) \to [\![(\Delta_1 \ominus \Gamma_2) \oplus \Delta_2]\!]$$

where $[\![(\Delta_1 \ominus \Gamma_2) \oplus \Delta_2]\!]$ is defined exactly as in (17). With this bookkeeping in place we can now define $[\![e_1; e_2]\!] : [\![\Gamma_1 \oplus (\Gamma_2 \ominus \Delta_1)]\!] \to [\![(\Delta_1 \ominus \Gamma_2) \oplus \Delta_2]\!]$ as

$$[\![e_1; e_2]\!] = T_3 \circ (\mathrm{Id} \otimes e_2) \circ T_2 \circ (e_1 \otimes \mathrm{Id}) \circ T_1 \tag{18}$$

As a simple example consider the program `x_1 := 3.5 ; x_2 := x_1 + 7.3`. Using the sequential composition rule we can derive the following typing-checking proof:

$$\dfrac{\dfrac{\emptyset \vdash \mathtt{3.5} : [0 \mapsto \mathtt{real}]}{[1 \mapsto \mathtt{real}] \vdash \mathtt{x\_1 := 3.5} : [1 \mapsto \mathtt{real}]} \qquad \dfrac{\dfrac{[1 \mapsto \mathtt{real}] \vdash \mathtt{x\_1} : [0 \mapsto \mathtt{real}] \qquad \emptyset \vdash \mathtt{7.3} : [0 \mapsto \mathtt{real}]}{[1 \mapsto \mathtt{real}] \vdash \mathtt{x\_1 + 7.3} : [0 \mapsto \mathtt{real}]}}{[1 \mapsto \mathtt{real}, 2 \mapsto \mathtt{real}] \vdash \mathtt{x\_2 := x\_1 + 7.3} : [2 \mapsto \mathtt{real}]}}{[1 \mapsto \mathtt{real}, 2 \mapsto \mathtt{real}] \vdash \mathtt{x\_1 := 3.5 \ ; \ x\_2 := x\_1 + 7.3} : [1 \mapsto \mathtt{real}, 2 \mapsto \mathtt{real}]}$$

The semantics of the program is computed step-by-step as follows

$[\![\mathtt{x\_1 := 3.5}]\!] : \mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}, \mu \mapsto ev_{\mathbb{R}}(\mu)\delta_{3.5}$

$[\![\mathtt{x\_1 + 7.3}]\!] : \mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}, \mu \mapsto (+)_*(\mu \times \delta_{7.3})$

$[\![\mathtt{x\_2 := x\_1 + 7.3}]\!] : \mathcal{M}\mathbb{R} \,\widehat{\otimes}_{|\pi|}\, \mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}, \mu \otimes \nu \mapsto ev_{\mathbb{R}}(\nu)(+)_*(\mu \times \delta_{7.3})$

$[\![\mathtt{x\_1 := 3.5 \ ; \ x\_2 := x\_1 + 7.3}]\!] : \mathcal{M}\mathbb{R} \,\widehat{\otimes}_{|\pi|}\, \mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R} \,\widehat{\otimes}_{|\pi|}\, \mathcal{M}\mathbb{R}, \mu \otimes \nu \mapsto ev_{\mathbb{R}}(\mu)\delta_{3.5} \otimes ev_{\mathbb{R}}(\nu)\delta_{10.8}$

since $(+)_*(\delta_{3.5} \times \delta_{7.8}) = \delta_{10.8}$. Note that if we had replaced the instruction `x_2 := x_1 + 7.3` by `x_2 := x_1 * sin(x_1)`, we would not have been able to type-check the problem. The linearity of our system would prevent us from using twice `x_1` unless it was explicitly declared as duplicable. The tensor rule of Fig. 3 is interpreted in a very similar way by tensoring with the identity $\mathrm{Id}_{[\![\Delta]\!]}$ up to permutations.

5.2.6 λ-abstraction and Function Application. These are interpreted exactly as expected in a monoidal closed category, namely via the adjunction $- \,\widehat{\otimes}_{|\pi|}\, [\![\mathtt{S}]\!] \dashv [[\![\mathtt{S}]\!], -]$ and ordinary function application.

Remark 21. While the denotation of λ-abstraction is immediately given by the monoidal closed structure of **RoBan**, the following point is worth making. Assume a context of ground types only. In the system of [Kozen 1981], such a context is of the shape $\mathcal{M}(X_1 \times \ldots \times X_n)$, i.e. *any* joint distribution over the variables can be considered as an input to the program. If we were Cartesian

closed, such a context would be of the shape $\mathcal{M}X_1 \times \ldots \times \mathcal{M}X_n$, i.e. *only product distributions* would be considered as potential inputs to the program. Our semantics lies between these two possibilities since the context is now $\mathcal{M}X_1 \mathbin{\widehat{\otimes}_{|\pi|}} \ldots \mathbin{\widehat{\otimes}_{|\pi|}} \mathcal{M}X_n$. This means that not all joint probabilities can be λ-abstracted on, only those which live in the tensor product (i.e. limits of Cauchy sequences of linear combinations of product measures). Put differently, our system only allows λ-abstraction if the probabilistic state of the machine is prepared (to use a quantum analogy) to a distribution in the positive projective tensor product. This is typically much larger than the set of product distributions, but is smaller than the set of all joint distribution (unless the underlying sets are finite, in which case all joint distribution can be expressed as elements of the tensor product).

5.2.7 Semantics of `sampler` *and* `sample`. The semantics of these commands are extensions of the unit and co-unit of the Giry monad [Giry 1982] respectively. First we need the following easy result.

Theorem 22. *The semantics of every measure type is isometrically and monotonically embedded in a space of measures $\mathcal{M}X$.*

We can now define the semantics of `sampler`. Suppose we have $[\![e]\!] : [\![\Gamma]\!] \to [\![\mathtt{T}]\!]$ and, by Th. 22, that $[\![\mathtt{T}]\!]$ is isometrically and monotonically embedded in the space $\mathcal{M}X$. Now consider the map $\eta : X \to \mathcal{M}X, x \mapsto \delta_x$ (which is *not* an operator) and define denotation of $\mathtt{sampler}(e)$ as the positive operator $[\![\Gamma]\!] \to \mathcal{M}[\![\mathtt{T}]\!]$

$$[\![\mathtt{sampler}(e)]\!] := \mathcal{M}\eta \circ [\![e]\!]. \tag{19}$$

The semantics of `sample` works in the opposite direction. Suppose we have $[\![e]\!] : [\![\Gamma]\!] \to \mathcal{M}[\![\mathtt{T}]\!]$ with $[\![\mathtt{T}]\!]$ isometrically and monotonically embedded in $\mathcal{M}X$, then each element of $\mathcal{M}[\![\mathtt{T}]\!]$ is also an element of $\mathcal{M}\mathcal{M}X$. We can define a map

$$m_X : \mathcal{M}\mathcal{M}X \to \mathcal{M}X \qquad \rho \mapsto \lambda B. \int_{B^+(\mathcal{M}X)} ev_B(\mu)\, d\rho \tag{20}$$

where we recall that $B^+(\mathcal{M}X)$ is the positive unit ball of the space $\mathcal{M}X$. To see that this map is well-defined, recall first that $\mathcal{M}\mathcal{M}X$ is equipped with the Borel σ-algebra generated by the total variation norm. Since the evaluation maps $ev_B : \mathcal{M}\mathcal{M}X \to \mathcal{M}X$ are always continuous w.r.t. the total variation norm, they are in particular also measurable. Moreover, they are also tautologically bounded on $B^+(\mathcal{M}X)$ and this set has finite measure since ρ is of bounded variation. It follows that the integral in eq. (20) is well-defined. The map m_X clearly defines a positive operator, and in the case where ρ is supported by the set of probability distributions, i.e. the shell $\{\mu \in (\mathcal{M}X)^+ : \|\mu\| = 1\}$ of the positive unit ball, m_X coincides with the multiplication of the Giry monad. We are now ready to define $[\![\mathtt{sample}(e)]\!] : [\![\Gamma]\!] \to [\![\mathtt{T}]\!]$

$$[\![\mathtt{sample}(e)]\!] := m_X \circ [\![e]\!].$$

We can now interpret the type of the Gaussian inference program in Fig. 4. In defining the semantics of built-in operations we saw that the semantics of $\emptyset \vdash \mathtt{normal}(0,1) : [0 \mapsto \mathtt{Mreal}]$ is the linear map $\mathbb{R} \to \mathcal{M}\mathcal{M}\mathbb{R}$ mapping 1 to the Dirac delta over the normal distribution $\mathcal{N}(0,1)$. It follows that

$$[\![\mathtt{sample}(\mathtt{normal}(0,1))]\!](1) = \mathcal{N}(0,1)$$

and by unravelling the definition we similarly find that

$$[\![\mathtt{sample}(\mathtt{normal}(\mathtt{sample}(\mathtt{normal}(0,1)),1)]\!](1) = \lambda B. \int_{x\in\mathbb{R}} \mathcal{N}(x,1)(B)\, d\mathcal{N}(0,1) = \mathcal{N}(0,\sqrt{2})$$

which is the pushforward of $\mathcal{N}(0,1)$ by the kernel $\lambda x.\, \mathcal{N}(x,1)$ as described in (1). It follows that the output of the Gaussian inference program is interpreted as a linear operator

$$(\mathcal{M}\mathbb{R})_{\mathcal{N}(0,\sqrt{2})} \longrightarrow (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,1)}.$$

We will describe in § 5.2.9 what this operator actually is.

Remark 23. Although definitions eq. (19) and eq. (20) look a lot like the unit and multiplication of the Giry monad, it is worth emphasising that we are not defining a monad. We do not need to, since we are working at the 'lifted' level where all inputs are distributions and all programs are interpreted as linear operators on these distributions. This means in particular that questions of measurability, which are crucial to the definition of a Giry-like monad on quasi-Borel spaces [Heunen et al. 2017] and of sequencing of stable functions [Ehrhard et al. 2017] (in each case via an integral of the same form as in eq. (20)), do not apply here. We are only interested in linear operators. Some of these will arise from measurable maps via the pushforward operation and others will not, but non-measurable maps never enter into the picture.

5.2.8 Conditionals and `while` *Loops.* Given a boolean test $\Gamma \vdash e : [0 \mapsto \texttt{bool}]$ interpreted as an operator $[\![e]\!] : [\![\Gamma]\!] \to \mathcal{M}2$, the order-completeness of $[\![\Gamma]\!]$ allows us to define the maps

$$T_e \colon [\![\Gamma]\!]^+ \to [\![\Gamma]\!]^+, \gamma \mapsto \bigwedge \{0 \leq \gamma' \leq \gamma : [\![e]\!](\gamma')(1) = [\![e]\!](\gamma)(1)\}$$
$$F_e \colon [\![\Gamma]\!]^+ \to [\![\Gamma]\!]^+, \gamma \mapsto \bigwedge \{0 \leq \gamma' \leq \gamma : [\![e]\!](\gamma')(0) = [\![e]\!](\gamma)(0)\}$$

Proposition 24. *The maps T_e and F_e are positive, additive and $\mathbb{R}^+$-homogeneous.*

Since regular ordered Banach spaces have a generating cone, we can uniquely extend T_e and F_e to the entire space $[\![\Gamma]\!]$. The way to understand T_e and F_e is as operators computing a kind of 'weakest pre-conditions'. Given a state $\gamma \in [\![\Gamma]\!]$, $T_e(\gamma)$ describes the smallest (positive) state on which the test e has the same probability of success as on γ. Similarly, $F_e(\gamma)$ describes the smallest state below γ on which the test e has the same probability of failure as on γ.

Using the operators T_e and F_e we define the semantics of $\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \Delta$ as the operator:

$$[\![\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3]\!] : [\![\Gamma]\!] \to [\![\Delta]\!], \gamma \mapsto [\![e_2]\!] \circ T_{e_1}(\gamma) + [\![e_3]\!] \circ F_{e_1}(\gamma) \tag{21}$$

In the case of deterministic tests, the semantics defined by (21) recovers precisely the semantics of [Kozen 1981]. There, $[\![\Gamma]\!]$ is a measure space $\mathcal{M}X$ and $[\![e_1]\!] : \mathcal{M}X \to \mathcal{M}2$ is of the shape $\mathcal{M}b$ for a measurable map $b : X \to 2$ which specifies a measurable subset B of X. We claim that $T_e : \mathcal{M}X \to \mathcal{M}X$ sends a probability measure μ to the measure μ_B defined by $\mu_B(A) = \mu(A \cap B)$, exactly as the operator e_B of [Kozen 1981, 3.3.4]. By unravelling the definition we want to show:

$$\mu_B = \bigwedge \{0 \leq \nu \leq \mu : \nu(B) = \mu(B)\}$$

Note first that μ_B belongs to the set above, so it remains to show that it is its minimal element. Let ν also belong to this set, and let A be a measurable set. We decompose A as $A = (A \cap B) \uplus (A \cap B^c)$. By definition

$$\mu_B(A \cap B^c) = 0 \leq \nu(A \cap B^c) \text{ since } 0 \leq \nu.$$

Moreover we have

$$\mu_B(A \cap B) = \mu(A \cap B) = \nu(A \cap B).$$

For if we had $\nu(A \cap B) < \mu(A \cap B)$, then in order to keep $\mu(B) = \nu(B)$ we would need $\nu(A^c \cap B) > \mu(A^c \cap B)$, a contradiction with $\nu \leq \mu$. Thus $\mu_B \leq \nu$ as claimed and the semantics of $\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \mathsf{T}$ becomes the operator

$$\mu \mapsto [\![e_2]\!](\mu_B) + [\![e_3]\!](\mu_{B^c})$$

exactly as in [Kozen 1981]. However, the semantics (21) also covers the case of probabilistic tests. As an example consider the small program given in Fig. 5. For notational clarity let us write e for the program `sample(normal(x_1, 1)) > 0`. As we saw in § 5.2.3, the denotation of

```
x_1:=sample(normal(0,1);
if sample(normal(x_1,1))>0 then
        x_1:=x_1+1
else
        x_1:=x_1-1
```

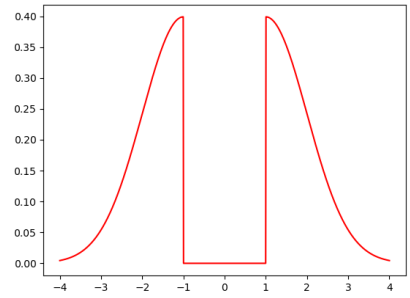


Fig. 5. `if then else` program with a probabilistic guard and its output distribution (up to a scalar).

`x_1:= sample(normal(0,1))` is the operator $\mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}, \mu \mapsto ev_{\mathbb{R}}(\mu)\mathcal{N}(0,1)$, thus it is enough to see how T_e and F_e operate on inputs of the shape $\lambda\mathcal{N}(0,1)$. Following the same computation as in § 5.2.7, one gets that

$$[\![e]\!](\lambda\mathcal{N}(0,1))(1) = \lambda\mathcal{N}(0,\sqrt{2})([0,+\infty[) = \frac{\lambda}{2}$$

since $\lambda\mathcal{N}(0,\sqrt{2})$ is symmetric and centred at 0. We thus have

$$T_e(\lambda\mathcal{N}(0,1)) = \bigwedge \left\{ 0 \le \mu \le \lambda\mathcal{N}(0,1) : [\![e]\!](\mu)(1) = \frac{\lambda}{2} \right\} = \lambda\mathcal{N}(0,1)_{[0,+\infty[}$$

where $\lambda\mathcal{N}(0,1)_{[0,+\infty[}$ is the measure defined by $\lambda\mathcal{N}(0,1)_{[0,+\infty[}(B) = \lambda\mathcal{N}(0,1)(B \cap [0,+\infty[)$. Similarly, $T_e(\lambda\mathcal{N}(0,1)) = \lambda\mathcal{N}(0,1)_{]-\infty,0]}$. We can now compute the denotation of the entire program as the operator $\mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}$ depicted graphically in Fig. 5 and defined by

$$\mu \mapsto ev_{\mathbb{R}}(\mu)\mathcal{N}(1,1)_{[1,+\infty[} + ev_{\mathbb{R}}(\mu)\mathcal{N}(-1,1)_{]-\infty,-1]}$$

The semantics of `while` loops is given precisely as in [Kozen 1981]: we want equivalence between the programs

$$\Gamma \vdash \texttt{while}\ e_1\ \texttt{do}\ e_2 : \Gamma \qquad \text{and} \qquad \Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ ;\ \texttt{while}\ e_1\ \texttt{do}\ e_2\ \texttt{else skip} : \Gamma.$$

Using (21), the equivalence above means that the operator $W : [\![\Gamma]\!] \to [\![\Gamma]\!]$ denoting `while` e_1 `do` e_2 should be a solution of the equation

$$W = W \circ [\![e_2]\!] \circ T_{e_1} + F_{e_1} \tag{22}$$

Following [Kozen 1981, 3.3.5], the least fixed point of the operator $W \mapsto W \circ [\![e_2]\!] \circ T_{e_1} + F_{e_1}$ exists by order-completeness of $[\![\Gamma]\!]$.

5.2.9 Semantics of `observe`. Following the typing rule of `observe` given in Fig. 3, let us assume that we have $[\![e]\!] : [\![(\mathsf{S},\mu)]\!] \to [\![\mathsf{T}]\!]$ with S,T of measure type. We also make the assumption, which we will justify in Th. 26 below, that $[\![e]\!]$ is not just regular, but positive. Under this assumption if $\nu \le K\mu$, then $[\![e]\!](\nu) \le K[\![e]\!](\mu)$, i.e. $[\![e]\!]$ restricts to an operator

$$[\![e]\!] : (\mathcal{M}X)^{UB}_{\mu} \to (\mathcal{M}Y)^{UB}_{[\![e]\!](\mu)}$$

where $(\mathcal{M}X)^{UB}_{\mu}$ is the set of measures uniformly bounded by a multiple of μ (see [Chaput et al. 2014]). The semantics of `observe`(e) is then fundamentally contained in the Köthe dual operator

$$[\![e]\!]^{\sigma} : \left((\mathcal{M}Y)^{UB}_{[\![e]\!](\mu)}\right)^{\sigma} \to \left((\mathcal{M}X)^{UB}_{\mu}\right)^{\sigma}.$$

It is not hard to check by using the Riesz Representation and Functional Representations natural transformations (RR and FR in Diagram (2)) that $\left((\mathcal{M}Y)_\nu^{UB}\right)^\sigma \simeq (\mathrm{L}_p(Y,\nu))^\sigma \simeq (\mathcal{M}Y)_\nu$, and thus $[\![e]\!]^\sigma$ can, modulo these isomorphisms, be typed as an operator

$$[\![e]\!]^\sigma : (\mathcal{M}Y)_{[\![e]\!](\mu)} \to (\mathcal{M}X)_\mu \tag{23}$$

which is what the typing rule for `observe` requires.

To illustrate how this semantics really implements the Bayesian inversion described in § 3.2.2, let us again consider the simple Gaussian inference program given in Fig. 4. The underlying Bayesian model is given by the probability kernel $\mathcal{N}(-,1) : \mathbb{R} \to \mathscr{M}\mathbb{R}$ and the prior $\mathcal{N}(0,1)$ on $\mathbb{R}$. Together these define a **Krn**-arrow $(\mathbb{R}, \mathcal{N}(0,1)) \to (\mathbb{R}, \mathcal{N}(0,\sqrt{2}))$ which is implemented by the program

$$[\mathtt{x} \mapsto (\mathtt{real}, \mathtt{normal}(\mathtt{0},\mathtt{1}))] \vdash \mathtt{sample}(\mathtt{normal}(\mathtt{x},\mathtt{1})) : [\mathtt{0} \mapsto \mathtt{real}]$$

whose denotation is the positive operator

$$\mathcal{M}_-(\mathcal{N}(-,1)) : (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,1)} \to \mathcal{M}\mathbb{R}.$$

Using the same argument as above, we can restrict this operator as follows

$$\mathcal{M}_-(\mathcal{N}(-,1)) : (\mathcal{M}\mathbb{R})^{UB}_{\mathcal{N}(0,1)} \to (\mathcal{M}\mathbb{R})^{UB}_{\mathcal{N}(0,\sqrt{2})}.$$

As stated above, all the information about the semantics of `observe(sample(normal(x,1)))` is contained in the Köthe dual of this operator, which, through the Riesz Representation and Functional Representations natural transformation, can be typed modulo isomorphism as

$$(\mathcal{M}_-(\mathcal{N}(-,1)))^\sigma : (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,\sqrt{2})} \to (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,1)}.$$

Using the other half of diagram (2), that is to say the Radon-Nikodym and Measure Representation natural transformations (RN and MR in diagram 2), this operator is equal, modulo isomorphism, to the operator

$$\mathcal{M}_-(\mathcal{N}(-,1)^\dagger) : (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,\sqrt{2})} \to (\mathcal{M}\mathbb{R})_{\mathcal{N}(0,1)}.$$

Here the Bayesian inverse of our original probability kernel appears explicitly, showing that our semantics indeed captures the notion of Bayesian inverse.

There is one final subtlety which we need to account for. Given $[\![e]\!] : [\![(\mathsf{S},\mu)]\!] \to [\![\mathsf{T}]\!]$, the typing rule for `observe` in fact makes the whole semantics described above parametric in a choice of measure absolutely continuous w.r.t. the prior μ (see Fig. 3). This is a simple technicality: morally and practically the parameter will always be set to the prior itself, in which case we get as output of the program the Köthe dual described by (23). Mathematically however, we can choose as input any $\nu \ll \mu$; the output operator is then defined by the following tortuous path (similar to constructions in [Chaput et al. 2014])

$$(\mathcal{M}Y)_{[\![e]\!](\mu)} \xrightarrow{\simeq} \left((\mathcal{M}Y)_-[\![e]\!](\mu)^{UB}\right)^\sigma \xrightarrow{i^\sigma} \left((\mathcal{M}Y)^{UB}_{[\![e]\!](\nu)}\right)^\sigma \xrightarrow{[\![e]\!]^\sigma} \left((\mathcal{M}X)^{UB}_\nu\right)^\sigma \xrightarrow{\simeq} (\mathcal{M}X)_\nu \xrightarrow{j} (\mathcal{M}X)_\mu$$

where i, j are the obvious inclusions.

Remark 25. The semantics of `observe` in term of Köthe duals is more general than a semantics in terms of Bayesian inversion/disintegration. Nothing prevents the introduction of ground types which stand for measurable spaces in which disintegrations do not exist. However, the Köthe dual will still exist. Thus our semantics is free of some of the 'pointful' technicalities surrounding the existence of disintegrations, and follow the 'pointless' perspective advocated in [Clerc et al. 2017]. Similarly, we do not have to worry about the ambiguity caused by the fact that disintegrations are only defined up to a null set: the Köthe dual of an operator between regular ordered Banach spaces exists completely unambiguously.

5.3 Properties of the Semantics

5.3.1 Norm. We can extend [Kozen 1981, Th. 3.3.8] by a straightforward induction and show that:

Theorem 26. *The semantics of any program is a positive operator of norm at most* 1.

However another result [Kozen 1981, Th. 6.1], namely that the denotation of a program is entirely determined by its action on point masses does *not* hold any more. The reason is interesting and is worth a few words. It is immediate from the type system (Fig. 3) and the denotation of Bayesian types that the domain of the interpretation of an `observe` statement may *not contain any point masses at all*. For example in the case of the Gaussian inference program of 4.2.3, this domain is $(\mathcal{M}\mathbb{R})_{\mathcal{N}(0,1)}$ which contains no point masses at all.

5.3.2 Commutativity. The symmetric monoidal structure of **RoBan** and the denotation of sequential compositions given by (18) provide an easy proof of the commutativity of our semantics.

Theorem 27. *Suppose* $\Gamma \vdash e_1 : [0 \mapsto \mathsf{T}_1]$ *and* $\Delta \vdash e_2 : [0 \mapsto \mathsf{T}_2], \mathrm{supp}(\Gamma) \cap \mathrm{supp}(\Delta) = \emptyset$ *are derivable, then for* $j \notin \mathrm{supp}(\Gamma), i \notin \mathrm{supp}(\Delta)$

$$[\![\mathtt{x\_}i \coloneqq e_1;\ \mathtt{x\_}j \coloneqq e_2]\!] = [\![\mathtt{x\_}j \coloneqq e_2;\ \mathtt{x\_}i \coloneqq e_1]\!]$$

In [Staton 2017], a probabilistic language is given a semantics in terms of (s-finite) Markov kernels composed using Kleisli composition. Commutativity is therefore shown via a Fubini theorem. Here we are dealing with linear operators composing in the usual way. The connection between the two approaches is of course that each kernel gives rise to a linear operator given by the pushforward under a kernel defined in (1). The key difference is that our system is linear, and the assignments $\mathtt{x\_}i \coloneqq e_1$ and $\mathtt{x\_}j \coloneqq e_2$ are therefore tensored rather than composed, which explains why commutativity boils down to the symmetric monoidal structure of **RoBan** rather than a Fubini theorem.

5.3.3 Comparison with Semantics à la Scott. There are interesting parallels to be drawn between our semantics and the Scott-Strachey semantics in terms of domains.

Looking at ground types first, it is worth noting that just like the *flat domain* functor turns a set (of integers for example) into a valid semantic object (a domain), so the functor $\mathcal{M}$ turns a measurable set into a valid semantic object (a regular ordered Banach space). Similarly, just like the flat domain functor can turn a partial map between sets into a total map between domains, so the functor $\mathcal{M}$ turns a partial measurable map into a linear operator. Partiality is encoded by the presence of the bottom element in the case of domain, and by the possibility to lose mass (i.e. get subdistributions) in the case of spaces of measures.

We do not know yet if the semantics of every program in our language is σ-order continuous, which would be the equivalent of Scott-continuous in our setting. The fundamental difference however is that our semantic category is not Cartesian closed, but monoidal closed.

5.3.4 A Note on Operational Semantics and Adequacy. Defining a small-step operational semantics for probabilistic languages is far more intricate than in the non-probabilistic case; see e.g. [Borgström et al. 2016; Staton et al. 2016]. The difficulty is compounded in the case of our system by the combined presence of continuously-parametrized families of distributions over a range of types, higher-order types, and the Bayesian inference command `observe`.

The presence of continuously-parametrized families of distributions, whose parameters might only be fixed at sample time and which can be called arbitrarily many times from a `while` loop, means that the operational semantics of `sample` will require a carefully designed global source of randomness, perhaps in the form of a countable product of abstract measured spaces.

The presence of higher-order types also introduces subtle issues. As mentioned, a natural approach is to provide a fixed global source of randomness that can be sampled as necessary during the computation. However, care must be taken: the naive approach of packaging the random source with a function at the site of the function's definition can in some circumstances lead to the reuse of samples at different locations in the program, thereby breaking independence. For example, in the evaluation of $(\lambda x.xa(xb))(\lambda y.\text{if flip=heads then } cy \text{ else } dy)$, the same coin would be used twice in the resolution of two flips when the body of the first expression is evaluated. To achieve adequacy with respect to an operational semantics, it would be necessary that the source be dynamically scoped to ensure that samples are not reused. Thus functions must allow the random source to be supplied as a parameter at the call site. The difficulty here is that the form of the randomness required by the function is not known to the calling context, so some conversion or approximation would be required.

Finally, the operational interpretation of the `observe` command is at the moment very unclear. One may be tempted to implement a rejection-sampling scheme, but this favours a (not terribly good) Bayesian inference algorithm, on the sole basis that it has an easily understandable operational behaviour. In fact the principle of separation of concerns in languages like Anglican, where the programmer writes an `observe` statement in the model part of the code and then specifies *separately* which inference algorithm will implement it, suggests an operational semantics that is parametric in the choice of an inference algorithm.

With so many interesting but difficult questions, we believe that an operational semantics and a proof of adequacy for the kind of language described in this work deserve a detailed treatment in a future work.

6 CORRECTNESS OF GIBBS SAMPLING

We have now all the tools necessary to prove the denotational correctness of the Gibbs sampling algorithm given in Fig. 1. We assume that `xyz[i]` and `xyz[i, j]`, $i \neq j$ are syntactic sugar for built-in functions associated with the obvious projections from $[\![\mathsf{X} \times \mathsf{Y} \times \mathsf{Z}]\!]$, e.g.

$$[\![\texttt{xyz[1]}]\!] = \mathcal{M}\pi_{\mathsf{X}} : \mathcal{M}[\![\mathsf{X} \times \mathsf{Y} \times \mathsf{Z}]\!] \to \mathcal{M}[\![\mathsf{X}]\!], \quad [\![\texttt{xyz[1, 3]}]\!] = \mathcal{M}\pi_{\mathsf{X}\times\mathsf{Z}} : \mathcal{M}[\![\mathsf{X} \times \mathsf{Y} \times \mathsf{Z}]\!] \to \mathcal{M}[\![\mathsf{X} \times \mathsf{Z}]\!]$$

The semantics of the built-in function `write_X` : $(\mathsf{X} \times \mathsf{Y} \times \mathsf{Z}, \mathsf{X}, \mathsf{X} \times \mathsf{Y} \times \mathsf{Z})$ is given by the function $((x, y, z), x_{\text{new}}) \mapsto (x_{\text{new}}, y, z)$ in the way described in § 5.2.2, and similarly for `write_Y` and `write_Z`. For notational convenience we denote by `mc` the body of the loop, by P the product $\mathsf{X} \times \mathsf{Y} \times \mathsf{Z}$ and by P_{X} (resp. $\mathsf{P}_{\mathsf{Y}}, \mathsf{P}_{\mathsf{Z}}$) the product $\mathsf{Y} \times \mathsf{Z}$ (resp. $\mathsf{X} \times \mathsf{Z}$, $\mathsf{Y} \times \mathsf{Y}$).

The pre-condition that the inputs `cond_`i, $i = 1, 2, 3$ are the conditional distributions of the joint distribution μ can be formalized as saying that:

$$[\![\texttt{sample(cond\_1)}]\!] := \mathcal{M}\pi^{\dagger}_{\mathsf{P}_{\mathsf{Z}}} : \mathcal{M}[\![\mathsf{P}_{\mathsf{Z}}]\!]_{(\pi_{\mathsf{P}_{\mathsf{Z}}})_*\mu} \to \mathcal{M}[\![\mathsf{P}]\!]_{\mu}$$
$$[\![\texttt{sample(cond\_2)}]\!] := \mathcal{M}\pi^{\dagger}_{\mathsf{P}_{\mathsf{Y}}} : \mathcal{M}[\![\mathsf{P}_{\mathsf{Y}}]\!]_{(\pi_{\mathsf{P}_{\mathsf{Y}}})_*\mu} \to \mathcal{M}[\![\mathsf{P}]\!]_{\mu}$$
$$[\![\texttt{sample(cond\_3)}]\!] := \mathcal{M}\pi^{\dagger}_{\mathsf{P}_{\mathsf{X}}} : \mathcal{M}[\![\mathsf{P}_{\mathsf{X}}]\!]_{(\pi_{\mathsf{P}_{\mathsf{X}}})_*\mu} \to \mathcal{M}[\![\mathsf{P}]\!]_{\mu}$$

where $(\pi_{\mathsf{P}_{\mathsf{Z}}})_*\mu$ is the pushforward of the measure μ under the projection $\pi_{\mathsf{P}_{\mathsf{Z}}}$, and similarly for the other permutations. Using the typing rules of Fig. 3 we can type-check the `while` loop of the program

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{[1\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 2\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 3\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P}), 4\mapsto\mathsf{P}] \vdash \texttt{mc} : [4\mapsto\mathsf{P}]}}{[1, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 2, 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 3, 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P}), 4\mapsto\mathsf{P}] \vdash \texttt{mc} : [4\mapsto\mathsf{P}, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P})]}}{[4\mapsto\mathsf{P}, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P})] \vdash \texttt{mc} : [4\mapsto\mathsf{P}, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P})]}}{[4\mapsto\mathsf{P}, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P})] \vdash \texttt{while true do mc} : [4\mapsto\mathsf{P}, 5\mapsto!(\mathsf{P}_{\mathsf{Z}}\to\mathsf{P}), 6\mapsto!(\mathsf{P}_{\mathsf{Y}}\to\mathsf{P}), 7\mapsto!(\mathsf{P}_{\mathsf{X}}\to\mathsf{P})]}$$

and we can therefore interpret `while true do mc` as the linear operator

$$!\llbracket \mathsf{P}_Z \to \mathsf{P} \rrbracket \mathbin{\widehat{\otimes}_{|\pi|}} !\llbracket \mathsf{P}_Y \to \mathsf{P} \rrbracket \mathbin{\widehat{\otimes}_{|\pi|}} !\llbracket \mathsf{P}_X \to \mathsf{P} \rrbracket ! \mathbin{\widehat{\otimes}_{|\pi|}} \llbracket \mathsf{P} \rrbracket \longrightarrow !\llbracket \mathsf{P}_Z \to \mathsf{P} \rrbracket \mathbin{\widehat{\otimes}_{|\pi|}} !\llbracket \mathsf{P}_Y \to \mathsf{P} \rrbracket \mathbin{\widehat{\otimes}_{|\pi|}} !\llbracket \mathsf{P}_X \to \mathsf{P} \rrbracket ! \mathbin{\widehat{\otimes}_{|\pi|}} \llbracket \mathsf{P} \rrbracket$$

defined by the fixpoint equation (22), that is to say satisfying

$$\llbracket \texttt{while true do mc} \rrbracket (f_1 \otimes f_2 \otimes f_3 \otimes \nu) = \llbracket \texttt{while true do mc} \rrbracket (f_1 \otimes f_2 \otimes f_3 \otimes T(\nu)) \tag{24}$$

where $T = \llbracket \mathsf{mc} \rrbracket : \mathcal{M}\llbracket \mathsf{P} \rrbracket_\mu \to \mathcal{M}\llbracket \mathsf{P} \rrbracket_\mu$ is the Markov operator representing the chain built by the program and given by:

$$T = \mathcal{M}\pi^\dagger_{\mathsf{P}_X} \circ \mathcal{M}\pi_{\mathsf{P}_X} \circ \mathcal{M}\pi^\dagger_{\mathsf{P}_Y} \circ \mathcal{M}\pi_{\mathsf{P}_Y} \circ \mathcal{M}\pi^\dagger_{\mathsf{P}_Z} \circ \mathcal{M}\pi_{\mathsf{P}_Z}$$

We now show that as long as the pre-conditions are satisfied the operator defined by (24) sends $\llbracket \mathsf{cond\_1} \rrbracket \otimes \llbracket \mathsf{cond\_2} \rrbracket \otimes \llbracket \mathsf{cond\_3} \rrbracket \otimes \nu$ to $\llbracket \mathsf{cond\_1} \rrbracket \otimes \llbracket \mathsf{cond\_2} \rrbracket \otimes \llbracket \mathsf{cond\_3} \rrbracket \otimes \mu$, where μ is the desired distribution. We start with the following, almost immediate, fact:

Theorem 28. $\mu \in \mathrm{fix}(T)$

The second observation is that T is a Markov operator and that Markov operators satisfy the *mean ergodic theorem* (see [Dunford et al. 1971]).

Theorem 29 (Lem. 8.3, Th. 8.24 [Eisner et al. 2015])**.** *The limit* $P_T := \lim_{n\to\infty} \frac{1}{n} \sum_{j=0}^{n-1} T^j$ *exists in the strong operator topology. Moreover,* P_T *is a projection operator onto* $\mathrm{fix}(T)$ *and* $P_T \circ T = P_T$.

The last step is to show that T is ergodic/irreducible. By [Prop. 7.15, [Eisner et al. 2015]] this amounts to proving the following result which is easily shown by using the naturality the Radon-Nikodym natural transformation of Diagram (10) and the functorial properties of $(-)^\dagger$.

Theorem 30. $\dim \mathrm{fix}(T) = 1$.

We can now conclude by combining Theorems 28, 29 and 30 to get that the operator

$$(f_1, f_2, f_3, \nu) \mapsto (f_1, f_2, f_3, P_{T(f_1,f_2,f_3)}(\nu))$$

(where the copying of f_1, f_2, f_3 is taken care of by the comonoidal structure of the exponential) is a solution of the fixpoint equation (24) when f_1, f_2, f_3 satisfy the pre-conditions. Moreover, if ν satisfies the pre-condition $\nu \ll \mu$, we get that the 'output' of the program is given by

$$\llbracket \texttt{while true do mc} \rrbracket (f_1, f_2, f_3, \nu) = (f_1, f_2, f_3, \mu)$$

as desired. If the return command was reached it would return the expected distribution μ, and we therefore have a proof of correctness of Gibbs sampling *in the limit of an infinite number of iterations*. This mirrors the operational picture where sampling from μ is only guaranteed in the limit of an infinite number of transitions through the chain described by T.

Of course, it would be interesting to quantify how many iterations of the loop are necessary to approach μ up to some pre-specified error, i.e. how quickly the sum $\frac{1}{n}\sum_i^n T^i$ converges to P_T. This is beyond the scope of the present work, but given the wealth of result in ergodic theory (results on mixing times for example) it is a question which lends itself perfectly to our semantics. In other words our semantics paves the way not just for correctness proofs, but also for more quantitative results with obvious practical applications.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1637532. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The first author also wishes to acknowledge support from the ERC grant ProfoundNet and the EPSRC platform grant EP/P010040/1.

REFERENCES

S. Abramsky, R. Blute, and P. Panangaden. 1999. Nuclear and trace ideals in tensored ⋆-categories. *Journal of Pure and Applied Algebra* 143, 1-3 (1999), 3–47.

C. Aliprantis and K. Border. 1999. *Infinite dimensional analysis*. Vol. 32006. Springer.

C. D. Aliprantis and O. Burkinshaw. 2006. *Positive operators*. Vol. 119. Springer Science & Business Media.

J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 33–46.

J. T. Chang and D. Pollard. 1997. Conditioning as disintegration. *Statistica Neerlandica* 51, 3 (1997), 287–317.

P. Chaput, V. Danos, P. Panangaden, and G. Plotkin. 2014. Approximating Markov Processes by averaging. *J. ACM* 61, 1 (Jan. 2014).

F. Clerc, F. Dahlqvist, V. Danos, and I. Garnier. 2017. Pointless Learning. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017. Proceedings*.

R. Crubillé, T. Ehrhard, M. Pagani, and C. Tasson. 2017. The free exponential modality of probabilistic coherence spaces. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 20–35.

F. Dahlqvist, V. Danos, I. Garnier, and A. Silva. 2018. Borel kernels and their Approximations, Categorically. In *Mathematical Foundations of Programming Semantics (MFPS)*.

E. B. Davies. 1968. The structure and ideal theory of the predual of a Banach lattice. *Trans. Amer. Math. Soc.* 131, 2 (1968), 544–555.

J. Dieudonné. 1951. Sur les espaces de Köthe. *Jour. d'Analyse Math.* 1, 1 (1951), 81–115.

N. Dunford, J. T. Schwartz, W. G Bade, and R. G. Bartle. 1971. *Linear operators I.* Wiley-interscience New York.

T. Ehrhard, M. Pagani, and C. Tasson. 2017. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 59.

T. Ehrhard, C. Tasson, and M. Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 309–320.

T. Eisner, B. Farkas, M. Haase, and R. Nagel. 2015. *Operator theoretic aspects of ergodic theory*. Vol. 272. Springer.

D. H. Fremlin. 1972. Tensor products of Archimedean vector lattices. *American Journal of Mathematics* 94, 3 (1972), 777–798.

D. H. Fremlin. 1974. Tensor products of Banach lattices. *Math. Ann.* 211, 2 (1974), 87–106.

S. Geman and D. Geman. 1987. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. In *Readings in computer vision*. Elsevier, 564–584.

M. Giry. 1982. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*. Springer, 68–85.

A. Grothendieck. 1955. *Produits tensoriels topologiques et espaces nucléaires*. Vol. 16. American Mathematical Soc.

B. Hayes. 2001. Computing Science: Randomness as a Resource. *American Scientist* 89, 4 (2001), 300–304.

O. Heunen, C.and Kammar, S. Staton, and H. Yang. 2017. A convenient category for higher-order probability theory. In *LICS*. IEEE, 1–12.

A. S. Kechris. 1995. *Classical descriptive set theory*. Graduate Text in Mathematics, Vol. 156. Springer.

D. Kozen. 1981. Semantics of probabilistic programs. *J. Comput. Syst. Sci.* 22, 3 (June 1981), 328–350.

D. Kozen. 1985. A probabilistic *PDL*. *J. Comput. Syst. Sci.* 30, 2 (April 1985), 162–178.

D. Licata and E. Finster. 2014. Eilenberg-MacLane spaces in homotopy type theory. In *LICS*. ACM, 66.

P. Mellies. 2009. Categorical semantics of linear logic. *Panoramas et syntheses* 27 (2009), 15–215.

P. Melliès, N. Tabareau, and C. Tasson. 2018. An explicit formula for the free exponential modality of linear logic. *Mathematical Structures in Computer Science* 28, 7 (2018), 1253–1286.

K. C. Min. 1983. An exponential law for regular ordered Banach spaces. *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 24, 3 (1983), 279–298.

R. A. Ryan. 2013. *Introduction to tensor products of Banach spaces*. Springer Science & Business Media.

A. Ścibior, O. Kammar, M. Vákár, S. Staton, H. Yang, Y. Cai, K. Ostermann, S. K. Moss, C. Heunen, and Z. Ghahramani. 2017. Denotational validation of higher-order Bayesian inference. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 60.

P. Selinger and B. Valiron. 2008. A linear-non-linear model for a computational call-by-value lambda calculus. In *International Conference on Foundations of Software Science and Computational Structures*. Springer, 81–96.

S. Staton. 2017. Commutative semantics for probabilistic programming. In *ESOP*. Springer, 855–879.

S. Staton, F. Wood, H. Yang, C. Heunen, and O. Kammar. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–10.

M. Vákár, O. Kammar, and S. Staton. 2019. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 36.

A. W. Wickstead. 2007. Regular operators between Banach lattices. In *Positivity*. Springer, 255–279.

G. Wittstock. 1974. Ordered normed tensor products. In *Foundations of quantum mechanics and ordered linear spaces*. Springer, 67–84.
Y. Wong and K. Ng. 1973. *Partially ordered topological vector spaces*. Oxford University Press.
A. C. Zaanen. 2012. *Introduction to operator theory in Riesz spaces*. Springer Science & Business Media.