Deterministic stream-sampling for probabilistic programming: semantics and verification

Fredrik Dahlqvist
Queen Mary University of London
and *University College London*
London, United Kingdom
f.dahlqvist@qmul.ac.uk

Alexandra Silva
Department of Computer Science
Cornell University
Ithaca, USA
alexandra.silva@cornell.edu

William Smith
Department of Computer Science
University College London
London, United Kingdom
william.smith.19@ucl.ac.uk

***Abstract*—Probabilistic programming languages rely fundamentally on some notion of sampling, and this is doubly true for probabilistic programming languages which perform Bayesian inference using Monte Carlo techniques. Verifying samplers—proving that they generate samples from the correct distribution—is crucial to the use of probabilistic programming languages for statistical modelling and inference. However, the typical denotational semantics of probabilistic programs is incompatible with deterministic notions of sampling. This is problematic, considering that most statistical inference is performed using pseudorandom number generators.**

We present a higher-order probabilistic programming language centred on the notion of *samplers* and *sampler operations*. We give this language an operational and denotational semantics in terms of continuous maps between topological spaces. Our language also supports discontinuous operations, such as comparisons between reals, by using the type system to track discontinuities. This feature might be of independent interest, for example in the context of differentiable programming.

Using this language, we develop tools for the formal verification of sampler correctness. We present an equational calculus to reason about equivalence of samplers, and a sound calculus to prove semantic correctness of samplers, i.e. that a sampler correctly targets a given measure by construction.

***Index Terms*—Probabilistic programming, operational and denotational semantics, verification**

I. Introduction

Probabilistic programming languages without conditioning – that is to say, programming languages capable of drawing random samples – and the concepts of Monte Carlo methods and randomized algorithms have been around as long as true computers have[1]; however, the introduction of languages with conditioning, higher-order features, continuous variables, recursion, and their application to statistical modelling and machine learning, is a product of the twenty-first century [36], [26], [15], [27], [40], [7], [3]. Since a probabilistic programming language with conditioning must come equipped with a range of inference algorithms and sampling methods, and since the rate of introduction of these has increased in recent years, new formal methods must be developed for the *verification* of these algorithms.

We aim to make the verification of inference algorithms straightforward by introducing a language endowed with a sampler type, featuring many of the sampler operations used in these inference algorithms, and a calculus for reasoning about correctness of these samplers relative to intended 'target' distributions. The semantics of our language is fully deterministic, in order to allow the use of deterministic – pseudorandom – samplers, in a simple manner and without paradox.

When assigning operational and denotational semantics to probabilistic programs, an interesting asymmetry emerges: the simplest reasonable denotational semantics of a first-order language with continuous datatypes is in terms of probability measures [21], while the simplest reasonable operational semantics is in terms of sampled *values* – the latter being much closer to the intuitions used by programmers of languages with the ability to draw samples.

The denotational semantics of probabilistic programming languages, in terms of measures (broadly construed) is well-understood [21], [16], [12], [38], [8]. The most common approaches to the operational semantics of such a language, as highlighted by [12], are *trace semantics* and *Markov chain semantics*. The latter is the chosen operational semantics for a number of probabilistic λ-calculi [11], [22], [5], [13], [12], [14] and probabilistic languages [34], [38], but does not speak of sampled values, only of distributions on execution paths. From the perspective of the asymmetry described above, it is thus closer to a denotational semantics.

Trace semantics, originally developed in [21] and later applied in [32], [5], [9], [2], assumes that for each distribution in the language, an infinite set of samples has been produced ahead-of-time. When a sample is requested, the head of this sequence is popped and used in the computation, and the tail of the sequence is kept available for further sampling. This perspective models samplers, such as `rand()`, as functions with hidden side effects on the state of the machine – in line with a programmer's intuition on the nature of sequential calls to `rand()`. The natural notion of adequacy with respect to the denotational semantics is to show that subject to the assumption that each element of these sequences is sampled independently from its corresponding distribution, the resulting pushforward

This work was supported by the Leverhulme Project Grant "Verification of Machine Learning Algorithms".

[1]See [4] for an overview of probabilistic programming languages, [6] for a historical overview of the Monte Carlo method, and [23] for a history of pseudorandom number generation.

979-8-3503-3587-3/23/$31.00 ©2023 IEEE

through the program is identical to the program's denotational semantics. We see two issues with this approach.

First, the supposition that all samples are pre-computed ahead-of-time is incompatible with pseudorandom generation of 'random' values, since computationally-generated samples and truly-random samples are in fact distinguishable. For example, if $(x_0, x_1, \ldots)$ is a sequence of samples targeting the distribution P which was produced via iteration of a computable map $x_{n+1} = T(x_n)$, then the program $T(x) - x$ will behave differently if x is a 'truly random' sample from P than if x is produced by the aforementioned iterative procedure, and so the operational and denotational semantics no longer cohere; similar counterexamples exist for any pseudorandom number generator. If $(x_0, x_1, \ldots)$ is a deterministic sequence, meaningful coherence between the operational and denotational sequence can only be assured if it is assumed that this sequence is *Martin-Löf random* (first defined in [29], later generalised to computable metric spaces in [17]); unfortunately, all Martin-Löf random sequences are uncomputable. Pseudorandom numbers are in fact used in simulation far more commonly than 'true' physical randomness, as they are typically faster to obtain, have the advantage of being reproducible given a particular seed, and, subject to certain assumptions, can even have better convergence properties [24]. We find the inability of trace semantics to describe pseudorandom number generation to be a significant weakness.

Second, from the trace semantics perspective, the notion of a sampler type is inextricably bound up in issues regarding side-effects, which makes verification challenging. In order to properly assign the syntax `rand`, without parentheses, a meaningful semantics as a function, we must give it a monadic interpretation as in [32], accounting for its hidden effect on the trace. The correctness of code which inputs and outputs samplers – sampler operations – is then subject to the state of the trace when computation is started, which is not contained within either the code of the sampler operation in question or the code of the samplers it inputs. This pattern, of using sampler operations to create composite samplers which make use of other 'primitive' samplers, is a central theme in computational statistics. In particular, inference algorithms within Bayesian statistics, to which probabilistic programming languages with conditioning compile, make heavy use of this technique. Commonly-used sampling methods, such as importance sampling, rejection sampling, and particle Markov chain Monte Carlo methods, are naturally understood as composite samplers of this type [1], [33]. We prefer a side-effect-free perspective from which the correctness of sampler operations can be demonstrated subject to assumptions about the samplers input to these programs, as opposed to a perspective in which their correctness depends also on the state of the machine on which these operations are run.

Contributions. We develop a language based around the idea of sampler types and sampler operations, which allows reasoning about deterministic and real-valued samplers, and which is designed to make verification of these samplers natural. A syntax, operational semantics, and denotational semantics for our language are introduced in § III, and an adequacy result relating them is shown. § IV lays out a notion of equivalence of samplers, which will be applied to simplify programs. Finally, in § V, we discuss methods for proving that samplers target the desired probability measure (i.e. verifying samplers), and introduce a sound calculus for verifying the correctness of composite samplers which is capable of demonstrating the soundness of common Monte Carlo techniques such as importance sampling and rejection sampling.

II. Examples

The purpose of this section is twofold. First, we present examples of how samplers are transformed in order to create new samplers. We use this opportunity to informally introduce a language with a sampler type constructor Σ and operations for constructing and manipulating samplers. Second, we present techniques to reason about the correctness of sampling algorithms. These come in two flavours. We reason *equationally* about the equivalence between samplers (see § IV), and we reason *semantically* about whether a sampler does the job it is designed to do – namely, generate deviates from a target distribution (see § V).

A. Von Neumann extractor

We begin with a simple family of discrete samplers known as von Neumann extractors. This example will illustrate the concept of a sampler's *self-product*, a central concept for a language featuring sampler types. The von Neumann extractor [37] is a simple procedure which, given flips from a *biased* coin on $\{\texttt{True}, \texttt{False}\}$ with probability $p \in (0, 1)$ of landing `True`, produces flips from an *unbiased* coin with probability $1/2$ of landing `True`. We view this as a sampler `v` of Boolean type – notation $\texttt{v} : \Sigma\,\texttt{B}$ – which, given another Boolean-valued sampler $\texttt{flip} : \Sigma\,\texttt{B}$ representing our biased coin, constructs an unbiased Boolean-valued sampler. A simple implementation of the von Neumann extractor is given in Listing 1.

```
let choice = λb : B × B .
  if (fst(b) and snd(b)) or (not fst(b) and
    not snd(b)) then 0 else 1
  in let proj = λb : B × B . fst(b)
    in map(proj, reweight(choice, flip²))
```

Listing 1: von Neumann extractor

The idea behind this algorithm is that if (b_1, b_2) are sampled independently from a Bernoulli distribution with parameter p, then the probabilities of the outcomes $(b_1, b_2) = (\texttt{False}, \texttt{True})$ and $(b_1, b_2) = (\texttt{True}, \texttt{False})$ are both $p(1-p)$, and so the first element b_1 of each pair is an unbiased flip. In Listing 1, samples (b_1, b_2) where $b_1 = b_2$ are removed by the `reweight` operation, which sets the *weight* of such pairs to zero; then, the command `map` applies the function `proj` to each pair (b_1, b_2), returning the sampler whose outputs are only the first element b_1.

Note the appearance of $\texttt{flip}^2$ in the von Neumann extractor; this is the 'self-product' of the sampler `flip`. Where $\texttt{flip} : \Sigma\,\texttt{B}$ produces Boolean-valued samples, $\texttt{flip}^2 : \Sigma\,(\texttt{B} \times \texttt{B})$ produces samples which are pairs of

$$\dfrac{\dfrac{\vdash \texttt{flip}^2 : \Sigma\,(\texttt{B}\times\texttt{B}) \rightsquigarrow \mathrm{Ber}(p)^2 \quad \vdash \texttt{choice} : \texttt{B}\times\texttt{B} \to \texttt{R}^+}{\vdash \texttt{reweight}(\texttt{choice},\texttt{flip}^2) : \Sigma\,(\texttt{B}\times\texttt{B}) \rightsquigarrow [\![\texttt{choice}]\!]\cdot \mathrm{Ber}(p)^2} \quad \vdash \texttt{proj} : \texttt{B}\times\texttt{B}\to\texttt{B}}{\vdash \texttt{map}(\texttt{proj},\texttt{reweight}(\texttt{choice},\texttt{flip}^2)) : \Sigma\,\texttt{B} \rightsquigarrow [\![\texttt{proj}]\!]_* \,([\![\texttt{choice}]\!]\cdot \mathrm{Ber}(p)^2) = \mathrm{Ber}(1/2)}$$

Fig. 1: Validity of von Neumann extractor

$$\dfrac{\dfrac{\vdash \texttt{rand}^2 : \Sigma\,(\texttt{R}\times\texttt{R}) \rightsquigarrow U^2 \quad \vdash \texttt{plus} : \texttt{R}\times\texttt{R}\to\texttt{R}}{\vdash \texttt{map}(\texttt{plus},\texttt{rand}^2) : \Sigma\,\texttt{R} \rightsquigarrow [\![\texttt{plus}]\!]_*\, U^2} \quad \vdash \texttt{phi} : \texttt{R}\to\texttt{R}^+}{\vdash \texttt{reweight}(\texttt{phi},\texttt{map}(\texttt{plus},\texttt{rand}^2)) : \Sigma\,\texttt{R} \rightsquigarrow [\![\texttt{phi}]\!]\cdot([\![\texttt{plus}]\!]_*\, U^2) = P}$$

Fig. 2: Validity of importance sampling in Listing 2

Booleans. Given a sampler $t : \Sigma\,\texttt{T}$ of type $\texttt{T}$, the self-product $t^2 : \Sigma\,(\texttt{T}\times\texttt{T})$ is a sampler whose elements are adjacent samples from $\texttt{T}$; the same construction, detailed in § III, is easily extended to arbitrary self-powers $t^K : \Sigma\,(\texttt{T}^K)$.

The main application of our language is to serve as a setting for the formal verification of its samplers. Let $\texttt{v} : \Sigma\,\texttt{B}$ be the von Neumann extractor defined in Listing 1; the task of verifying $\texttt{v}$ is the task of showing that $\texttt{v}$ 'targets' the uniform distribution $\mathrm{Ber}\,(1/2)$ – meaning, informally, that in the limit of increasing sample size, $\texttt{v}$ generates unbiased flips. In § V-B, we define a relation $\rightsquigarrow$ between samplers and distributions which reifies this notion: we read $\vdash \texttt{v} : \Sigma\,\texttt{B} \rightsquigarrow \mathrm{Ber}\,(1/2)$ as 'the sampler $\texttt{v}$ targets the measure $\mathrm{Ber}\,(1/2)$'. The aforementioned self-product operation plays a crucial role in sampler verification, as it does *not* suffice, in order to conclude that $\texttt{v}$ targets $\mathrm{Ber}\,(1/2)$, to assume that $\texttt{flip}$ targets $\mathrm{Ber}(p)$ for some $p \in (0,1)$. Instead, we are required to make the stronger assumption that $\texttt{flip}^2$ targets $\mathrm{Ber}(p)^2$: in essence, that adjacent samples from $\texttt{flip}$ act as if they are independent. Following the literature on pseudorandom number generation, we refer to this property as *K-equidistribution* (in this case, for $K = 2$): we will say that a sampler s is K-equidistributed with respect to the distribution P if s^K targets P^K. For example, the assertion that a pseudorandom number generator which takes values in $\{0,\ldots,N-1\}$ is K-equidistributed with respect to the uniform distribution is the assertion that all K-length words $w \in \{0,\ldots,N-1\}^K$ are produced in equal proportion. Several commonly-used discrete PRNGs, such as xorshift and the Mersenne twister, have well-known K-equidistribution guarantees; see [39], [30].

In our calculus for asymptotic targeting, the validity of the von Neumann extractor is shown in Fig. 1. Before we begin this derivation, it must be shown that the von Neumann extractor $\texttt{v}$ is equivalent to the simplified sampler $\texttt{map}(\texttt{proj},\texttt{reweight}(\texttt{choice},\texttt{flip}^2))$ in a context in which access to a Boolean-typed sampler $\texttt{flip}$ is assumed, where $\texttt{proj}$ and $\texttt{choice}$ are defined as in Listing 1. We write this equivalence as $\texttt{flip} : \Sigma\,\texttt{B} \vdash \texttt{v} \approx \texttt{map}(\texttt{proj},\texttt{reweight}(\texttt{choice},\texttt{flip}^2)) : \Sigma\,\texttt{B}$; this equivalence relation is discussed in § IV, and is shown in this particular case using the let-binding rule of Table IV.

Having rewritten $\texttt{v}$ in this way, and using the hypothesis of 2-equidistribution $\vdash \texttt{flip}^2 : \Sigma\,\texttt{B} \rightsquigarrow \mathrm{Ber}(p)^2$, we derive our conclusion in Fig. 1 by applying the rules from § V-B corresponding to the sampler operations $\texttt{reweight}$ and $\texttt{map}$. These rules show us that $\texttt{v}$ targets the measure $[\![\texttt{proj}]\!]_*\,([\![\texttt{choice}]\!]\cdot \mathrm{Ber}(p)^2)$. (Here, as we will discuss in § V-B, $[\![f]\!]_*\,\mu$ denotes the pushforward of the measure μ through the function f, and the notation $f\cdot\mu$ denotes the measure μ reweighted by the density f.) To complete the proof, we show that this measure is identical to $\mathrm{Ber}(1/2)$, the uniform measure on $\texttt{B}$; this is straightforward. It is easily seen first that $[\![\texttt{choice}]\!]\cdot\mathrm{Ber}(p)^2$ assigns probability $1/2$ to the samples $(\texttt{True},\texttt{False})$ and $(\texttt{False},\texttt{True})$ and zero probability to all other samples; the desired result then follows by observing that the function $\texttt{proj}$ simply drops the second sample.

B. Importance sampling

A central application of the reweighting operation is its role in *importance sampling*. This is a commonly used technique [33], [6] in Bayesian learning and statistical inference, which transforms samples from a 'proposal' distribution Q on the latent space X into approximate samples from a 'target' distribution P, where P is absolutely continuous with respect to Q with Radon-Nikodym derivative $\frac{dP}{dQ}(x)$. Its operation is straightforward: for each sample $x_n \sim Q$, compute the sample's weight $w_n = \frac{dP}{dQ}(x)$, and then the *weighted sample* (x_n, w_n) is informally understood as an approximate sample from the target P. Formally, the normalised empirical measure $\sum_{n=1}^{N} \frac{w_n}{\sum_{i=1}^{N} w_i}\delta_{x_n}$, where δ_x is the Dirac measure at $x \in X$, converges weakly as $N\to\infty$ to the target measure P.

For example, consider the Bayesian inference problem in which the prior P_0 is the triangular distribution on $[0,2]$ and the likelihood of the observation $y = 3$ given the latent value x is a standard Gaussian $L(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(3-x)^2}{2}\right)$. Let P represent the corresponding posterior distribution, whose density is proportional to the pointwise product of the triangular and Gaussian densities; a simple importance-sampling procedure for sampling from P in our language is shown in Listing 2. Here, we assume access to a sampler $\texttt{rand}$ which targets the uniform distribution on $[0,1]$; recalling that a triangular random variable is the sum of two independent uniform random variables, and assuming $\texttt{rand}$ has the necessary independence property of 2-equidistribution, we sum two draws from $\texttt{rand}$ to produce a triangular random variable. Finally, we *reweight* the result according to the likelihood $L(x)$, yielding a sampler which targets the posterior distribution P corresponding to the observed datum $y = 3$.

$$\dfrac{\dfrac{\vdash \texttt{tri} \otimes \texttt{rand} : \Sigma\,\mathtt{T} \rightsquigarrow \mathrm{Tri} \otimes U \quad \vdash \texttt{accept} : \mathtt{T} \to \mathtt{R}^+}{\vdash \texttt{reweight}(\texttt{accept}, \texttt{tri} \otimes \texttt{rand}) : \Sigma\,\mathtt{T} \rightsquigarrow [\![\texttt{accept}]\!] \cdot (\mathrm{Tri} \otimes U)} \quad \vdash \texttt{proj} : \mathtt{T} \to \mathtt{R}}{\vdash \texttt{map}(\texttt{proj}, \texttt{reweight}(\texttt{accept}, \texttt{tri} \otimes \texttt{rand})) : \Sigma\,\mathtt{R} \rightsquigarrow [\![\texttt{proj}]\!]_* \left([\![\texttt{accept}]\!] \cdot (\mathrm{Tri} \otimes U)\right) = P}$$

Fig. 3: Validity of rejection sampling in Listing 3

```
let phi = λx : R . 1/sqrt(2*pi) * exp(-1/2*(3-
    x)*(3-x))
  in let plus = λu : R × R . fst(u) + snd(u)
    in reweight(phi, map(plus, rand²))
```

Listing 2: Importance sampling

The validity of this sampler – i.e. the fact that it targets the correct posterior distribution – follows easily in our targeting calculus. Under the hypothesis that `rand` produces 2-equidistributed samples with respect to the uniform distribution U, the derivation Fig. 2 proves that the sampler defined in Listing 2 targets the measure $[\![\texttt{phi}]\!] \cdot ([\![\texttt{plus}]\!]_* U^2)$. It remains to show that this measure is the desired P; once one shows that the sum of two independent uniform variates is triangular, this follows by definition of the reweighting operation $\cdot$. The same argument suffices for any observation y.

C. Rejection sampling

Our final example of sampler verification is an instance of the technique known as *rejection sampling*. Listing 3 applies rejection sampling from the prior to the same Bayesian inference problem discussed in § II-B to yield a sampler which targets the same posterior distribution P. Of particular importance is the *discontinuity* of the accept-reject step, which significantly complicates the argument of sampler verification in the presence of pseudorandom number generation.

```
let phi = λx : R . 1/sqrt(2*pi) * exp(-1/2*(3-
    x)*(3-x))
  in let accept = λ(u,v) : T .
    if v ≤ phi(u)*sqrt(2*pi) then 1 else 0
    in let proj = λz : T. fst(cast⟨R×R⟩(z)) in
      map(proj, reweight(accept, tri ⊗ rand))
```

Listing 3: Rejection sampling

In order to show the validity of rejection sampling from the prior, we must assume access to a sampler on the prior distribution (here `tri`), an independent standard uniform random sampler (here `rand`), and an upper bound $\sup_{x\in\mathbb{R}} L(x) = \frac{1}{2\pi}\sup_{x\in\mathbb{R}} \exp(-\frac{(3-x)^2}{2}) = \frac{1}{2\pi}$ on the likelihood, which is used in the acceptance condition. Fig. 3 proves that, subject to the natural independence assumption for the samplers `tri` and `rand`, the rejection sampler defined by Listing 3 targets the measure $[\![\texttt{proj}]\!]_* ([\![\texttt{accept}]\!] \cdot (\mathrm{Tri} \otimes U))$. We can then show, using standard methods, that this measure is identical to P, the posterior distribution also targeted by Listing 2.

We have omitted, for the moment, one crucial part of the proof. Note that, in both Listing 3 and Fig. 3, the function `accept`, which one might expect to have type $\mathtt{R} \times \mathtt{R} \to \mathtt{R}^+$, instead has type $\mathtt{T} \to \mathtt{R}^+$. Correspondingly, the product $\texttt{tri} \otimes \texttt{rand}$ must be assumed to produce samples of type `T`, rather than $\mathtt{R} \times \mathtt{R}$, and `proj` must accepts inputs of type `T` rather than pairs $\mathtt{R} \times \mathtt{R}$. The nature of this type `T`, a *subtype* of $\mathtt{R} \times \mathtt{R}$, will be explained in § III, but it encodes the fact that `accept` is discontinuous when viewed as a function on the standard topologies, as well as where those discontinuities are allowed to lie. The type-inference of Listing 3, detailing the structure of `T`, is given in [10, Fig. 6 and Fig. 7].

III. Language

A. Syntax

We use a λ-calculus with a notion of subtype and a type constructor Σ for samplers.

1) Types: Types are generated by the mostly standard grammar in Fig. 4a, where the set Ground of ground types is

$$\{\mathtt{N}, \mathtt{R}, \mathtt{R}^+\} \cup \{f^{-1}(i) \mid f \in \{\leq, <, \geq, >, =, \neq\}, i = 0, 1\}.$$

Our ground types include the natural, real and nonnegative real numbers, as well as important sets of pairs of reals: for example, $<^{-1}(1)$ will be denoted, as the notation suggests, by the pairs of reals whose first component is strictly smaller than the second. The boolean type $\mathtt{B} \triangleq 1 + 1$ will be treated as a ground type.

The only unusual type constructors are the *pullback types* ${}_s\mathtt{T}_t$ which – as the name suggests – will be interpreted as pullbacks (in fact inverse images), and the *sampler types* $\Sigma\,\mathtt{T}$ which will be defined as the coinductive (stream) types defined by the (syntactic) functors $\mathtt{T} \times \mathtt{R}^+ \times -$. In other words, we assume that samplers can be weighted; this covers the special case of unweighted samplers, in which every weight is set to 1. As these are the only coinductive types we need, and to highlight the central role played by samplers, we choose not to add generic coinductive types to the language.

The *subtyping relation* $\triangleleft$ on types is the reflexive transitive closure of the relation generated by the rules of Fig. 4b.

2) Terms: Fig. 4c presents the grammar generating the set Expr of terms in our language. We assume the existence of a set Func of built-in functions which come equipped with typing information $f : \mathtt{T} \to \mathtt{G}$, where `G` is a ground type. Some built-in functions will be continuous w.r.t. to the usual topologies, such as the addition operation $+ : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$, but others will be discontinuous, such as the comparison operators $\{\leq, <, \geq, >, =, \neq\} : \mathtt{R} \times \mathtt{R} \to \mathtt{B}$. Dealing with such functions is the main reason for adding coproducts to the grammar, as we will discuss in § III-C. We also employ the syntactic sugar

$$\texttt{if}\ b\ \texttt{then}\ s_{\mathtt{True}}\ \texttt{else}\ s_{\mathtt{False}} \triangleq \texttt{case}\ (b, \_)\ \texttt{of}\ \{(i, \_) \Rightarrow s_i\}_{i\in\mathbb{B}}.$$

Most of our language constructs are standard for a typed functional language without recursion, but we endow our language with several nonstandard (sampler) operations:

$$\mathtt{S},\mathtt{T} ::= \mathtt{G} \in \text{Ground} \mid 1 \mid \mathtt{S} \times \mathtt{T} \mid \mathtt{S} + \mathtt{T} \mid {}_s\mathtt{T}_t \mid \mathtt{S} \to \mathtt{T} \mid \Sigma\,\mathtt{T} \qquad s,t : \mathtt{T}$$

(a) Type grammar

$$\frac{}{f^{-1}(0) + f^{-1}(1) \lhd \mathtt{R} \times \mathtt{R}}\ f \in \{<,\leq,>,\geq,=,\neq\} \qquad \frac{\mathtt{S}_1 \lhd \mathtt{S}_2 \qquad \mathtt{T}_1 \lhd \mathtt{T}_2}{\mathtt{S}_1 \times \mathtt{T}_1 \lhd \mathtt{S}_2 \times \mathtt{T}_2} \qquad \frac{\mathtt{S}_1 \lhd \mathtt{S}_2 \qquad \mathtt{T}_1 \lhd \mathtt{T}_2}{\mathtt{S}_1 + \mathtt{T}_1 \lhd \mathtt{S} + \mathtt{T}_2}$$

$$\frac{\mathtt{S} \lhd \mathtt{T}}{\Sigma\,\mathtt{S} \lhd \Sigma\,\mathtt{T}} \qquad \frac{\sum_{i\in n} \mathtt{S}_i \lhd \mathtt{S} \quad \sum_{j\in m} \mathtt{S}'_i \lhd \mathtt{S}}{\sum_{i\in n, j\in m} \mathtt{S}_i \cap \mathtt{S}'_j \lhd \sum_{i\in n} \mathtt{S}_i} \qquad \frac{\sum_{i\in n} \mathtt{S}_i \lhd \mathtt{S} \quad \sum_{j\in m} \mathtt{S}'_j \lhd \mathtt{S}}{\sum_{i\in n, j\in m} \mathtt{S}_i \cap \mathtt{S}'_j \lhd \sum_{j\in m} \mathtt{S}_j}$$

(b) Subtyping rules

$$\begin{aligned}
t ::= {} & x \in \text{Var} \mid b \in \{\mathtt{True},\mathtt{False}\} \mid n \in \mathbb{N} \mid r \in \mathbb{R} \mid && \textit{Variables and constants}\\
& f(t,\ldots,t), f \in \text{Func} \mid \mathtt{cast}\langle \mathtt{T}\rangle t \mid && \textit{Built-in functions}\\
& \mathtt{case}\ t\ \mathtt{of}\ \{(i,x_i) \Rightarrow s_i\}_{i\in n} \mid \mathtt{in}_i\,(t) \mid \lambda x\colon \mathtt{T}.t \mid t(t) \mid \mathtt{let}\ x = t\ \mathtt{in}\ t \mid && \textit{Programming constructs}\\
& (t,t) \mid \mathtt{fst}(t) \mid \mathtt{snd}(t) \mid && \textit{Products}\\
& \mathtt{prng}(t,t) \mid t \otimes t \mid \mathtt{map}(t,t) \mid \mathtt{reweight}(t,t) \mid \mathtt{hd}(t) \mid \mathtt{wt}(t) \mid \mathtt{tl}(t) \mid \mathtt{thin}(t,t) && \textit{Sampler operations}
\end{aligned}$$

(c) Term grammar

Fig. 4: Grammars and subtyping rules

- The operation $\mathtt{prng}(f,t)$ is used to construct a sampler as a pseudo-random number generator, using an initial value t and a deterministic endomap f.
- $s \otimes t$ represents the product of samplers s,t.
- The syntax $\mathtt{map}(f,t)$ maps the function f over the elements produced by the sampler t to produce a new sampler, in analogy to the pushforward of a measure.
- The operation $\mathtt{reweight}(f,t)$ applies the reweighting scheme f to the sampler t to form a new sampler.
- Given a sampler t, the operation $\mathtt{hd}(t)$ returns the first sample produced by t, $\mathtt{wt}(t)$ the weight of the first sample produced by t, and $\mathtt{tl}(t)$ returns the sampler t but with its first sample-weight pair dropped.
- The operation $\mathtt{thin}(n,t)$, given a natural number n and a sampler t, returns the sampler which includes only those elements of t whose index is a multiple of n.

The intuition and purposes of most of these language constructs was explained in § II, and their precise meaning will be made clear when we introduce their semantics.

3) Well-formed terms: Our typing system is mostly standard and presented in Table I. The only non-standard rules are the *context-restriction rule* on the second line of Table I, and the typing rules for the sampler operations, which should be straightforward given their descriptions above. The purpose of the context-restriction rule is, in a nutshell, to be able to pass the result of a computation of type $\mathtt{T}$ which is continuous w.r.t. a topology τ on the denotation of $\mathtt{T}$, to a computation using a variable of type $\mathtt{T}$ but which is continuous w.r.t. to a finer topology $\tau' \supset \tau$ on the denotation of $\mathtt{T}$. After application of this rule, it is no longer possible to λ-abstract on the individual variables of the context. There are good semantic reasons for this feature, which we discuss in § III-C. For readability and intuition's sake, the rule is written using the syntactic sugar

$$t^{-1}(\mathtt{T}_i) \triangleq {}_{\mathtt{cast}\langle \mathtt{T}\rangle \mathtt{in}_i(x)}\mathtt{T}_t \qquad \text{where } x : \mathtt{T}_i \tag{1}$$

For the subtyping rules Fig. 4b, we use the syntactic sugar

$$\mathtt{S}_i \cap \mathtt{S}'_j \triangleq {}_{\mathtt{cast}\langle \mathtt{S}\rangle \mathtt{in}_i(x_i)}\mathtt{S}_{\mathtt{cast}\langle \mathtt{S}\rangle \mathtt{in}_j(x'_j)} \quad \text{where } x_i : \mathtt{S}_i, x'_j : \mathtt{S}'_j$$

Our typed lambda calculus does not feature recursion for two reasons. First, it is not necessary: as any computable probability measure can be obtained as a computable pushforward of the uniform measure on the unit interval [18], [17], any sampler language which features the sampler operation `map` can, given a uniform sampler, target any computable probability measure. In particular, many rejection samplers, which are commonly implemented recursively, can alternatively be implemented using the operation `reweight`, as shown in Listing 3. Second, the categorical semantics of a typed, probabilistic, higher-order lambda calculus with recursion are a very recent area of investigation [38]; we consider the inclusion of recursive samplers to be further work.

B. Operational semantics

In practice, in order to evaluate a program containing a sampler, one must specify a finite number of samples $N \in \mathbb{N}$ which are to be produced. Our (big-step) operational semantics correspondingly takes the form of a reduction relation $(t,N) \to v$, where the left side consists of a well-typed closed term $t \in \text{Expr}$ and a number of samples $N \in \mathbb{N}$, and the right side is a *value* $v \in \text{Value}$, i.e. a term generated by the grammar

$$v ::= x \in \text{Var} \mid g \in \mathtt{G} \mid (v,v) \mid \mathtt{in}_i\,(v) \mid \lambda x\colon \mathtt{T}.\ v \tag{2}$$

The rules of this big-step operational semantics, shown in full in [10, Table VI], are the usual rules for the standard language constructs, together with additional rules for our implemented sampler operations; these are given in Table II. For notational simplicity, these operations make use of lists (a,b,c,d), which are in fact interpreted within our language as nested pairs $(a,(b,(c,d)))$. In order to keep the rules readable, we also introduce the shorthand $(t,N) \to ((v_1,w_1),\ldots,(v_N,w_N))$ to denote the N reductions

$$\begin{aligned}
&(\mathtt{hd}(t), \mathtt{wt}(t)) \to (v_1,w_1),\\
&(\mathtt{hd}(\mathtt{tl}(t)),\ \mathtt{wt}(\mathtt{tl}(t))) \to (v_2,w_2),\ldots,\\
&(\mathtt{hd}(\mathtt{tl}^{N-1}(t)),\ \mathtt{wt}(\mathtt{tl}^{N-1}(t))) \to (v_N,w_N).
\end{aligned}$$

$$\frac{}{\Gamma \vdash g : \mathtt{G}}\ g \in [\![\mathtt{G}]\!] \qquad \frac{}{\Gamma, x : \mathtt{T}, \Delta \vdash x : \mathtt{T}} \qquad \frac{\Gamma \vdash t : \mathtt{T}}{\Gamma \vdash f(t) : \mathtt{G}}\ \mathrm{Func} \ni f : \mathtt{T} \to \mathtt{G} \qquad \frac{\Delta \vdash t : \mathtt{S}}{\Gamma \vdash \mathtt{cast}\langle \mathtt{T} \rangle t : \mathtt{T}}\ \mathtt{S} \lhd \mathtt{T}, \Gamma \lhd \Delta$$

$$\frac{\Gamma \vdash t : \mathtt{T},}{(x_1, \ldots, x_n) : \sum_{i \in m} t^{-1}(\mathtt{T}_i) \vdash t : \sum_{i \in m} \mathtt{T}_i}\ \sum_{i \in m} \mathtt{T}_i \lhd \mathtt{T}, \Gamma = x_1 : \mathtt{S}_1, \ldots, x_n : \mathtt{S}_n$$

$$\frac{\Gamma \vdash s : \mathtt{S} \quad \Gamma \vdash t : \mathtt{T}}{\Gamma \vdash (s,t) : \mathtt{S} \times \mathtt{T}} \qquad \frac{\Gamma \vdash t : \mathtt{S} \times \mathtt{T}}{\Gamma \vdash \mathtt{fst}(t) : \mathtt{S}} \qquad \frac{\Gamma \vdash t : \mathtt{S} \times \mathtt{T}}{\Gamma \vdash \mathtt{snd}(t) : \mathtt{T}} \qquad \frac{\Gamma, \mathtt{x} : \mathtt{S} \vdash t : \mathtt{T} \quad \Gamma \vdash s : \mathtt{S}}{\Gamma \vdash \mathtt{let}\ x = s\ \mathtt{in}\ t : \mathtt{T}}$$

$$\frac{\Gamma, x : \mathtt{S} \vdash t : \mathtt{T}}{\Gamma \vdash \lambda x : \mathtt{S}\ .\ t : \mathtt{S} \to \mathtt{T}} \qquad \frac{\Gamma \vdash s : \mathtt{S} \quad \Gamma \vdash t : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash t(s) : \mathtt{T}} \qquad \frac{\Gamma \vdash t : \mathtt{T}_j}{\Gamma \vdash \mathtt{in}_j\,(t) : \sum_{i \in n} \mathtt{T}_i}\ j \in n \qquad \frac{\Gamma \vdash t : \sum_{i \in I} \mathtt{T}_i \quad \Gamma, x_i : \mathtt{T}_i \vdash s_i : \mathtt{T}}{\Gamma \vdash \mathtt{case}\ t\ \mathtt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i \in I} : \mathtt{T}}$$

$$\frac{\Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash \mathtt{hd}(t) : \mathtt{T}} \qquad \frac{\Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash \mathtt{wt}(t) : \mathtt{R}^+} \qquad \frac{\Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash \mathtt{tl}(t) : \Sigma\,\mathtt{T}} \qquad \frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \quad \Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash s \otimes t : \Sigma\,(\mathtt{S} \times \mathtt{T})}$$

$$\frac{\Gamma \vdash s : \mathtt{T} \to \mathtt{T} \quad \Gamma \vdash t : \mathtt{T}}{\Gamma \vdash \mathtt{prng}(s,t) : \Sigma\,\mathtt{T}} \qquad \frac{\Gamma \vdash t : \Sigma\,\mathtt{T} \quad \Gamma \vdash n : \mathtt{N}}{\Gamma \vdash \mathtt{thin}(t,n) : \Sigma\,\mathtt{T}} \qquad \frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \quad \Gamma \vdash t : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash \mathtt{map}(t,s) : \Sigma\,\mathtt{T}} \qquad \frac{\Gamma \vdash s : \mathtt{T} \to \mathtt{R}^+ \quad \Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash \mathtt{reweight}(s,t) : \Sigma\,\mathtt{T}}$$

TABLE I: Typing rules

Note that the product of two weighted samplers has as its weights the product of its factors' weights. The product and the operation `reweight` are the only operations modifying the weights of samplers.

The following proposition shows that the operational semantics is well-formed in that for any $N \in \mathbb{N}$, samplers can only reduce to weighted lists of length N.

Proposition III.1. *[10, Appendix A] If $\vdash s : \Sigma\,\mathtt{S}$ is a closed sampler, then for any $N \in \mathbb{N}$, if $(s, N) \to v$, then v has the form $((v_1, w_1), \ldots, (v_N, w_N))$, where v_n are values and $w_n \in \mathbb{R}_{\geq 0}$ are weights. If $\mathtt{S}$ is not a sampler type, then $v_n : \mathtt{S}$; more generally, each v_n might be a weighted list itself.*

The self-product operation: Having clarified the meaning of the product and of the `thin` operation, we are now in a position to formally justify the operation which we referred to, in § II, as the 'self-product' of a sampler. To motivate it, consider a sampler $t : \Sigma\,\mathtt{T}$ which evaluates as $(\mathtt{t}, 2N) \to (x_1, \ldots, x_{2N})$, where for notational clarity we have omitted the weights. From the above operational semantics, the lagged sampler $\mathtt{thin}(2, \mathtt{t} \otimes \mathtt{tl}(\mathtt{t})) : \Sigma\,(\mathtt{T} \times \mathtt{T})$ evaluates to

$$(\mathtt{thin}(2, \mathtt{t} \otimes \mathtt{tl}(\mathtt{t})), N) \to ((x_1, x_2), (x_3, x_4), \ldots, (x_{2N-1}, x_{2N})).$$

This is the 'self-product' which was denoted $\mathtt{t}^2$ in § II. This notion is important because it is the construction which allows us to generate pairs of independent samples from a given sampler. Note that simply taking $t \otimes t$ will produce pairs of perfectly correlated samples: the operational semantics gives $(t \otimes t, N) \to ((x_1, x_1), \ldots (x_N, x_N))$. More generally, for any $K \in \mathbb{N}$, we define the K-fold self-product of a sampler as

$$t^K \triangleq \mathtt{thin}(K, t \otimes \mathtt{tl}(t) \otimes \ldots \otimes \mathtt{tl}^{K-1}(t)). \tag{3}$$

Sampling from t^K is intended to allow the sampling of K-tuples of independent deviates generated by the sampler K. Ultimately, it is only to define this self-product operation that the sampler operation `thin` is included at all, it being somewhat of an unnatural construct.

C. Denotational semantics

1) Denotational universe: We will see in § V that continuous maps play a special role in the verification of sampler properties. We therefore need a denotational domain in which continuity is a meaningful concept. We also need a Cartesian closed model, as we want to interpret the lambda-abstraction operation of our calculus. A standard solution is to consider the category of *compactly generated topological spaces* [35], [31], [25] (henceforth *CG-spaces*). A topological space X is compactly generated if it is Hausdorff and has the property that $C \subseteq X$ is closed iff $C \cap K$ is closed in K for every compact K in X [35, §1]. We need not worry about the theory of these spaces, but the following facts are essential in what follows.

Proposition III.2 ([35], [25])**.** *1) The category $\mathbf{CG}$ of CG-spaces and continuous functions is Cartesian closed.*
2) The category $\mathbf{CG}$ is complete and cocomplete.
3) Every metrizable topological space is CG.
4) Locally closed subsets (i.e. intersections of an open and a closed subset) of CG-spaces are compactly generated.

It is worth briefly describing the Cartesian closed structure of $\mathbf{CG}$. The product is in general different from the product in $\mathbf{Top}$, the category of topological spaces: if the usual product topology is not already compactly generated, then it needs to be modified to enforce compact generation [35, §4]. However, in most practical instances the usual product topology is already compactly generated – for example, any countable product of metrizable spaces is metrizable, and thus compactly generated by Prop. III.2. The internal hom $[X, Y]$ between CG-spaces X, Y is given by the set of continuous maps $X \to Y$ together with the topology of uniform convergence on compact sets, also known as the compact-open topology [35, §5].

2) Semantics of types: With this categorical model in place we define the semantics of types. The semantics of ground types is as expected: $[\![\mathtt{N}]\!] = \mathbb{N}$, equipped with the discrete topology, and $[\![\mathtt{R}]\!] = \mathbb{R}, [\![\mathtt{R}^+]\!] = [0, \infty)$ with the usual topology. The spaces $f^{-1}(i), f \in \{\leq, <, \geq, >, =, \neq\}, i \in 2$ are interpreted precisely as the notation suggests, e.g.

$$[\![<^{-1}(0)]\!] = \{(x, y) \mid x, y \in \mathbb{R} \wedge x \geq y\},$$
$$[\![=^{-1}(1)]\!] = \{(x, x) \mid x \in \mathbb{R}\}$$

together with the subspace topology inherited from $\mathbb{R} \times \mathbb{R}$. Since all these spaces are metrizable, our ground types are

$$\frac{((s(\mathtt{hd}(t)),\mathtt{wt}(t)),N)\to(v_1,w_1)\quad\dots\quad((s(\mathtt{hd}(\mathtt{tl}^{N-1}(t)),\mathtt{wt}(\mathtt{tl}^{N-1}(t))),N)\to(v_N,w_N)}{(\mathtt{map}(s,t),N)\to((v_1,w_1),\dots,(v_N,w_N))}$$

$$\frac{((\mathtt{hd}(t),s(\mathtt{hd}(t))\cdot\mathtt{wt}(t)),N)\to(v_1,w_1)\quad\dots\quad((\mathtt{hd}(\mathtt{tl}^{N-1}(t)),s(\mathtt{hd}(\mathtt{tl}^{N-1}(t)))\cdot\mathtt{wt}(\mathtt{tl}^{N-1}(t))),N)\to(v_N,w_N)}{(\mathtt{reweight}(s,t),N)\to((v_1,w_1),\dots,(v_N,w_N))}$$

$$\frac{(s,N)\to((v_1,w_1),\dots,(v_N,w_N))\quad(t,N)\to((v'_1,w'_1),\dots,(v'_N,w'_N))}{(s\otimes t,N)\to(((v_1,v'_1),w_1\cdot w'_1),\dots,((v_N,v'_N),w_N\cdot w'_N))}$$

$$\frac{(t,N)\to((v_1,w_1),\dots,(v_N,w_N))}{(\mathtt{hd}(t),N)\to v_1}\qquad\frac{(t,N)\to((v_1,w_1),\dots,(v_N,w_N))}{(\mathtt{tl}(t),N-1)\to((v_2,w_2),\dots,(v_N,w_N))}\qquad\frac{(t,N)\to((v_1,w_1),\dots,(v_N,w_N))}{(\mathtt{wt}(t),N)\to w_1}$$

$$\frac{(s,N)\to i\quad(t,Ni)\to((v_1,w_1),\dots,(v_{Ni},w_{Ni}))}{(\mathtt{thin}(s,t),N)\to((v_1,w_1),(v_{i+1},w_{i+1}),(v_{2i+1},w_{2i+1}),\dots,(v_{(N-1)i+1},w_{(N-1)i+1}))}$$

$$\frac{(t,N)\to v_1\quad(s(t),N)\to v_2\quad\dots\quad(s^{N-1}(t),N)\to v_N}{(\mathtt{prng}(s,t),N)\to((v_1,1),\dots,(v_N,1))}$$

TABLE II: Big-step operational semantics of sampler operations

interpreted in $\mathbf{CG}$ by Prop. III.2.

Products (including the unit type) and function types are interpreted in the obvious way using the Cartesian closed structure of $\mathbf{CG}$. Coproduct types are interpreted by coproducts in $\mathbf{CG}$, and given two terms $s,t:\mathtt{T}$ interpreted as $\mathbf{CG}$-morphisms $[\![s]\!]:A\to[\![\mathtt{T}]\!],[\![t]\!]:B\to[\![\mathtt{T}]\!]$, the pullback type ${}_s\mathtt{T}_t$ is interpreted as the pullback $A\times_{[\![\mathtt{T}]\!]}B$ of $[\![s]\!]$ along $[\![t]\!]$. All these spaces live in $\mathbf{CG}$ by Prop. III.2.

Since sampler types are coinductive types, their semantics will hinge on the existence of terminal coalgebras.

Theorem III.1 (Adámek)**.** *Let $\mathscr{C}$ be a category with terminal object 1, and $F:\mathscr{C}\to\mathscr{C}$ be a functor. If $\mathscr{C}$ has and F preserves ω^{op}-indexed limits, then the limit νF of $1\xleftarrow{!}F1\xleftarrow{F!}FF1\xleftarrow{FF!}\dots$ is the terminal coalgebra of F.*

Since $\mathbf{CG}$ is complete, it has ω^{op}-indexed limits. Recall that we want to interpret $\Sigma\,\mathtt{T}$ as the coinductive type defined by the 'functor' $\mathtt{T}\times\mathtt{R}^+\times-$. Formally, given a type $\mathtt{T}$ we want

$$[\![\Sigma\,\mathtt{T}]\!]\triangleq\nu([\![\mathtt{T}]\!]\times\mathbb{R}^+\times\mathrm{Id}).\tag{4}$$

Since products are limits, and limits commute with limits, it is clear that the functor $[\![\mathtt{T}]\!]\times\mathbb{R}^+\times\mathrm{Id}$ preserves limits, and in particular ω^{op}-indexed ones. Adámek's theorem thus guarantees the existence of an object satisfying (4). More concretely, since the termimal object 1 is trivially metrizable, and since $\mathbb{R}^+$ is metrizable, each object in the terminal sequence will be metrizable provided $[\![\mathtt{T}]\!]$ is, and thus $\prod_n([\![\mathtt{T}]\!]\times\mathbb{R}^+)^n$ will be metrizable whenever $[\![\mathtt{T}]\!]$ is, and will therefore be equipped with the usual product topology. The limit defining (4) is a closed subspace of this product, which means that the limit in $\mathbf{CG}$ defining $[\![\Sigma\,\mathtt{T}]\!]$ is the same as in $\mathbf{Top}$ when $[\![\mathtt{T}]\!]$ is metrizable (for example, if $\mathtt{T}$ is a ground type or a product of ground types). However, by defining $[\![\Sigma\,\mathtt{T}]\!]$ coinductively rather than simply as $([\![\mathtt{T}]\!]\times\mathbb{R}^+)^\omega$, we obtain a terminal coalgebra structure on $[\![\Sigma\,\mathtt{T}]\!]$, and therefore the ability to define sampler operations coinductively.

3) Semantics of the subtyping relation: Our language contains the predicates $f\in\{\leq,<,\geq,>,=,\neq\}$ (essential for rejection sampling § II-C) and yet is meant to be interpreted in a universe of topological spaces and continuous maps. These predicates are not continuous maps $\mathbb{R}\times\mathbb{R}\to 2$ for the usual topology on $\mathbb{R}\times\mathbb{R}$. However, for each such predicate f, the sets $[\![f^{-1}(0)]\!]$ and $[\![f^{-1}(1)]\!]$ are *locally closed sets*, that is to say the intersection of an open set and a closed set (for the usual topology on $\mathbb{R}\times\mathbb{R}$), and therefore CG-spaces by Prop. III.2, e.g. $[\![<^{-1}(0)]\!]$ is closed and $[\![<^{-1}(1)]\!]$ open.

Our central idea for dealing with discontinuities is that since $\mathbf{CG}$ is cocomplete, the space $[\![f^{-1}(0)]\!]+[\![f^{-1}(1)]\!]$ is a CG-space. This space has the nice property that f *is continuous* as a map $f:[\![f^{-1}(0)+f^{-1}(1)]\!]\to 2$. Since each $f^{-1}(i)$ is a type, we can enforce this semantics by simply *typing* these built-in functions in Func as $f:f^{-1}(0)+f^{-1}(1)\to\mathtt{B}$.

The topology on $[\![f^{-1}(0)+f^{-1}(1)]\!]$ is finer than the usual topology on $\mathbb{R}\times\mathbb{R}$, which means that the identity map $\mathrm{Id}:[\![f^{-1}(0)+f^{-1}(1)]\!]\to\mathbb{R}\times\mathbb{R}$ is continuous. This is the semantic basis for the axiom in Fig. 4b. From the other rules it is easy to see by induction that the subtyping relation is always between spaces *sharing the same carrier set* and is semantically given by coarsening the topology. In other words, if $\mathtt{S}\triangleleft\mathtt{T}$, then $[\![\mathtt{S}]\!]$ and $[\![\mathtt{T}]\!]$ share the same carrier and the corresponding identity map $\mathrm{Id}:[\![\mathtt{S}]\!]\to[\![\mathtt{T}]\!]$ is continuous.

Example III.1. *Let* $\mathtt{p}\triangleq\mathtt{if}\ x=0\ \mathtt{then}\ 1\ \mathtt{else}\ -1$*; we will first show how the context-restriction rule allows us to type-check this program. For readability's sake, let* $\mathtt{Eq}\triangleq{=}^{-1}(1)$ *and* $\mathtt{Neq}\triangleq{=}^{-1}(0)$*. We now derive, using* $=:\mathtt{Neq}+\mathtt{Eq}\to\mathtt{R}$*,*

$$\dfrac{\dfrac{x:\mathtt{R}\vdash x:\mathtt{R}\qquad\vdash 0:\mathtt{R}}{x:\mathtt{R}\vdash(x,0):\mathtt{R}\times\mathtt{R}}\ \mathtt{Neq}+\mathtt{Eq}\triangleleft\mathtt{R}\times\mathtt{R}}{\dfrac{x:(x,0)^{-1}\mathtt{Neq}+(x,0)^{-1}\mathtt{Eq}\vdash(x,0):\mathtt{Neq}+\mathtt{Eq}\qquad\vdash 1:\mathtt{R}\qquad\vdash -1:\mathtt{R}}{x:(x,0)^{-1}\mathtt{Neq}+(x,0)^{-1}\mathtt{Eq}\vdash x=0:\mathtt{B}}}{x:(x,0)^{-1}\mathtt{Neq}+(x,0)^{-1}\mathtt{Eq}\vdash\mathtt{if}\ x=0\ \mathtt{then}\ 1\ \mathtt{else}\ -1:\mathtt{R}}$$

Anticipating the semantics on terms discussed shortly, it can easily be shown that

$$[\![(x,0)^{-1}\mathtt{Neq}+(x,0)^{-1}\mathtt{Eq}]\!]=((-\infty,0)\cup(0,\infty))+\{0\}$$

and thus $[\![\mathtt{p}]\!]$ *is the* continuous *map*

$$[\![\mathtt{p}]\!] : ((-\infty, 0) \cup (0, \infty)) + \{0\} \to \mathbb{R}, x \mapsto \begin{cases} 1 & \text{if } x = 0 \\ -1 & \text{else} \end{cases}$$

4) Semantics of well-formed terms: Axioms, weakening, subtyping, product, projections, `let`-binding, λ-abstraction, function application, injections and pattern matching are interpreted in the expected way (given that **CG** is a Cartesian closed category with coproducts).

Continuous built-in functions, for example $+ : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$ or $\mathtt{exp} : \mathtt{R} \to \mathtt{R}$, are interpreted in the obvious way. As explained above, discontinuous built-in functions $\{\leq, <, \geq, >, =, \neq\}$ are typed in such a way that their natural interpretations are tautologically continuous.

We can now describe the semantics of the context-restriction rule. From the premise, our observations in § III-C3, and the side-conditions, we have morphisms

$$[\![t]\!] : \prod_{j \in n} [\![\mathtt{S}_j]\!] \to [\![\mathtt{T}]\!], \quad \text{and} \quad \mathrm{Id} : \coprod_{i \in m} [\![\mathtt{T}_i]\!] \to [\![\mathtt{T}]\!].$$

By Eq. (1) we interpret each 'inverse image type' $t^{-1}(\mathtt{T}_i)$ as the pullback (inverse image) of $[\![t]\!]$ along the inclusion $[\![\mathtt{T}]\!]_i \hookrightarrow \coprod_{i \in m} [\![\mathtt{T}_i]\!]$ which is, as the notation implies, simply given by $[\![t]\!]^{-1}([\![\mathtt{T}_i]\!])$. Since $\coprod_{i \in m} [\![\mathtt{T}_i]\!]$ and $[\![\mathtt{T}]\!]$ share the same carrier, it is clear that this defines a partition of $[\![\Gamma]\!]$, and we can thus retype t as a continuous map $\coprod_{i \in m} [\![t^{-1}(\mathtt{T}_i)]\!] \to \coprod_{i \in m} [\![\mathtt{T}_i]\!]$, interpreting the rule.

As mentioned earlier in this section, context-restriction prevents λ-abstraction; the following example illustrates why this must be the case.

Example III.2. *Consider the program* $x < y$ *derived by:*

$$\dfrac{\dfrac{x : \mathtt{R}, y : \mathtt{R} \vdash (x, y) : \mathtt{R} \times \mathtt{R}}{(x,y):(x,y)^{-1}(<^{-1}(0)) + (x,y)^{-1}(<^{-1}(1)) \vdash (x,y):<^{-1}(0) + <^{-1}(1)}}{(x,y) : (x,y)^{-1}(<^{-1}(0)) + (x,y)^{-1}(<^{-1}(1)) \vdash x < y : \mathtt{B}}$$

The interpretation of $x < y$ *is given by the continuous function*

$$[\![<]\!] : \{(x, y) \mid x < y\} + \{(x, y) \mid x \geq y\} \to 2.$$

Although it has the same carrier $\mathbb{R} \times \mathbb{R}$*, the domain of this map is no longer of product of topological spaces; it is now a coproduct of topological spaces. This means that it is no longer possible to* λ*-abstract over one of the variables of this function using the Cartesian closed structure of* **CG**.

In order to be able to λ*-abstract the map* $<$*, we would need a topology on* $\mathbb{R} \times \mathbb{R}$ *with the property that for any given* $x_0 \in \mathbb{R}$ *the function* $x_0 < - : \mathbb{R} \to 2$ *is continuous. This would introduce the open sets* $[x_0, \infty)$ *to the topology of* $\mathbb{R}$ *for each* $x_0 \in \mathbb{R}$*, meaning that we must equip* $\mathbb{R}$ *with the notoriously problematic lower limit topology (a.k.a. the Sorgenfrey line). Whether or not this is a CG-space seems to be a thorny question, possibly independent of ZF [20].*

Finally, we define the denotational semantics of sampler operations using the coinductive nature of sampler types. Recall that for a type $\mathtt{T}$, $[\![\Sigma\,\mathtt{T}]\!] \triangleq \nu([\![\mathtt{T}]\!] \times \mathbb{R}^+ \times \mathrm{Id})$. In particular, $[\![\Sigma\,\mathtt{T}]\!]$ comes equipped with a coalgebra structure map

$$\mathrm{unfold}_{\mathtt{T}} : [\![\Sigma\,\mathtt{T}]\!] \to [\![\mathtt{T}]\!] \times \mathbb{R}^+ \times [\![\Sigma\,\mathtt{T}]\!].$$

Moreover, for any other (continuous) coalgebra structure map $\gamma : X \to [\![\mathtt{T}]\!] \times \mathbb{R}^+ \times X$, the terminal nature of $[\![\Sigma\,\mathtt{T}]\!]$ provides a unique $[\![\mathtt{T}]\!] \times \mathbb{R}^+ \times \mathrm{Id}$-coalgebra morphism

$$\mathrm{beh}(\gamma) : X \to [\![\Sigma\,\mathtt{T}]\!].$$

Since $[\![\Sigma\,\mathtt{T}]\!]$ is interpreted in **CG**, it follows automatically that both $\mathrm{unfold}_{\mathtt{T}}$ and $\mathrm{beh}(\gamma)$ are continuous. However, what is not immediately clear is that beh is in fact continuous in γ.

Proposition III.3. *[10, Appendix A] Let* $F : \mathbf{CG} \to \mathbf{CG}$ *satisfy the condition of Thm. III.1 as well as the condition that* $\mathrm{int}(\nu F) \neq \emptyset$ *in* $\prod_i F^i 1$*, and let* $\mathrm{beh}_X : [X, FX] \to [X, \nu F]$ *be the (behaviour) map associating to any* F*-coalgebra structure on* X *the unique coalgebra morphism into the terminal coalgebra. The map* beh_X *is continuous, i.e. is a* **CG***-morphism.*

Using unfold and beh we define the denotational semantics of all the sampler operations in Table III. These definitions are precisely the infinite (coinductive) versions of the finitary transformations defined in the operational semantics of Table II. All the maps involved in these definitions are continuous; this follows from Prop. III.3 and the fact that evaluation and function composition are continuous operations on the internal hom sets of **CG** ([35, 5.2,5.9]).

D. Adequacy

This language features an interesting asymmetry in that its denotational semantics is written in terms of the coinductive sampler type $[\![\Sigma\,\mathtt{T}]\!]$, while its operational semantics is written in terms of finitary operations on finite sequences of samples. Moreover, the operational semantics is given in terms of reductions to *values*, i.e. terms whose types are constructed without the type constructor Σ, whereas the denotational semantics does not make this distinction. To establish a connection, we start by defining a generic way to convert terms of arbitrary types into values, following the idea behind the operational semantics. Given a type $\mathtt{T}$ and an integer N we inductively define its associated value type $\mathrm{val}^N(\mathtt{T}) \in \mathrm{Value}$ by:

$$\begin{array}{ll} \mathrm{val}^N(\mathtt{G}) = \mathtt{G} & \mathrm{val}^N(\Sigma\,\mathtt{T}) = \left(\mathrm{val}^N \mathtt{T}\right)^N \\ \mathrm{val}^N(\mathtt{S} * \mathtt{T}) = \mathrm{val}^N(\mathtt{S}) * \mathrm{val}^N(\mathtt{T}), & * \in \{\times, +, \to\} \end{array}$$

where $\mathtt{G} \in \mathrm{Ground}$.[2] We now define the *generalized projection* maps $p^N_{\mathtt{T}} : [\![\mathtt{T}]\!] \to \left[\!\left[\mathrm{val}^N(\mathtt{T})\right]\!\right]$ recursively via

$$\begin{array}{ll} p^N_{\mathtt{G}} = \mathrm{id}_{[\![\mathtt{G}]\!]}, & p^N_{\mathtt{S}*\mathtt{T}} = p^N_{\mathtt{S}} * p^N_{\mathtt{T}}, * \in \{\times, +\} \\ p^N_{\mathtt{S}\to\mathtt{T}} = \mathrm{id}_{[\![\mathtt{S}\to\mathtt{T}]\!]} & p^N_{\Sigma\mathtt{T}} = \pi_{1:N} \circ (p^N_{\mathtt{T}\times\mathtt{R}^+})^\omega \end{array}$$

The reader will have noticed that we have defined $p^N_{\mathtt{S}\to\mathtt{T}}$ trivially. The reason is that, as a quick examination of the rules of Table II will reveal, there is no conclusion and no premise

[2] Since we're only interested in closed samplers here, and since pullback types can only occur in a context, we need not define val^N on pullback types.

$$\frac{[\![\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] = f}{[\![\Gamma \vdash \mathtt{hd}(t) : \mathtt{T}]\!] = \pi_1 \circ \mathrm{unfold}_{\mathtt{T}} \circ f} \quad \frac{[\![\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] = f}{[\![\Gamma \vdash \mathtt{wt}(t) : \mathtt{R}^+]\!] = \pi_2 \circ \mathrm{unfold}_{\mathtt{T}} \circ f} \quad \frac{[\![\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] = f}{[\![\Gamma \vdash \mathtt{tl}(t) : \Sigma\,\mathtt{T}]\!] = \pi_3 \circ \mathrm{unfold}_{\mathtt{T}} \circ f}$$

$$\frac{[\![\Gamma \vdash s : \mathtt{N}]\!] = f \quad [\![\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] = g}{[\![\Gamma \vdash \mathtt{thin}(s,t) : \Sigma\,(\mathtt{T})]\!] = \mathrm{ev}_{\Sigma\mathtt{T},\Sigma\mathtt{T}} \circ (\mathrm{id}_{\Sigma\mathtt{T}} \times \mathrm{beh}_{\Sigma\mathtt{T}}) \circ \left(\mathrm{id}_{\Sigma\mathtt{T}} \times \left(\mathrm{unfold}_{\mathtt{T}} \circ (\pi_3 \circ \mathrm{unfold}_{\mathtt{T}})^{(\cdot\,-1)}\right)\right) \circ \langle f, g\rangle}$$

$$\frac{[\![\Gamma \vdash s : \Sigma\,\mathtt{S}]\!] = f \quad [\![\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] = g}{[\![\Gamma \vdash s \otimes t : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!] = \mathrm{beh}_{\Sigma\mathtt{S},\Sigma\mathtt{T}}\,(\pi_1 \times \pi_4 \times (\pi_2 \cdot \pi_5) \times \pi_3 \times \pi_6 \circ \mathrm{unfold}_{\mathtt{S}} \times \mathrm{unfold}_{\mathtt{T}}) \circ \langle f, g\rangle}$$

$$\frac{[\![\Gamma \vdash s : \Sigma\,\mathtt{S}]\!] = f \quad [\![\Gamma \vdash t : \mathtt{S} \to \mathtt{T}]\!] = g}{[\![\Gamma \vdash \mathtt{map}(t,s) : \Sigma\,\mathtt{T}]\!] = \mathrm{ev}_{\Sigma\mathtt{S},\Sigma\mathtt{T}} \circ (\mathrm{id}_{\Sigma\mathtt{S}} \times \mathrm{beh}_{\Sigma\mathtt{S}}) \circ (\mathrm{id}_{\Sigma\mathtt{S}} \times ((- \times \mathrm{id}_{\mathtt{R}^+} \times \mathrm{id}_{\Sigma\mathtt{S}}) \circ \mathrm{unfold}_{\mathtt{S}})) \circ \langle f, g\rangle}$$

$$\frac{[\![\Gamma \vdash s : \Sigma\,\mathtt{T}]\!] = f \quad [\![\Gamma \vdash t : \mathtt{T} \to \mathtt{R}^+]\!] = g}{[\![\Gamma \vdash \mathtt{reweight}(t,s)]\!] = \mathrm{ev}_{\Sigma\mathtt{T},\Sigma\mathtt{T}} \circ (\mathrm{id}_{\Sigma\mathtt{T}} \times \mathrm{beh}_{\Sigma\mathtt{T}}) \circ (\mathrm{id}_{\Sigma\mathtt{T}} \times ((\mathrm{id}_{\mathtt{T}} \times - \times \mathrm{id}_{\Sigma\mathtt{T}}) \circ \mathrm{unfold}_{\mathtt{T}})) \circ \langle f, g\rangle}$$

$$\frac{[\![\Gamma \vdash t : \mathtt{T}]\!] = f \quad [\![\Gamma \vdash s : \mathtt{T} \to \mathtt{T}]\!] = g}{[\![\Gamma \vdash \mathtt{prng}(s,t) : \Sigma\,\mathtt{T}]\!] = \mathrm{ev}_{\mathtt{T},\Sigma\mathtt{T}} \circ (\mathrm{id}_{\mathtt{T}} \times \mathrm{beh}_{\mathtt{T}}) \circ (\mathrm{id}_{\mathtt{T}} \times (\mathrm{id}_{\mathtt{T}} \times 1 \times -)) \circ \langle f, g\rangle}$$

TABLE III: Denotational semantics of sampler operations

of the type $(t, N) \to v$ where t is of function type. The only occurrence of terms of function types are within an evaluation, or are *values*, i.e. terms trivially reducing to themselves.

Theorem III.2. *[10, Appendix A] For any program* $\vdash t : \mathtt{T}$*, we have*

$$(t, N) \to v \Leftrightarrow p_{\mathtt{T}}^{N}([\![t]\!]) = [\![v]\!]\,.$$

IV. Equivalence of samplers

In order to implement a system for reasoning about whether a deterministic sampler targets a particular probability distribution, it is necessary to first define a notion of equivalence between samplers. Having such a system gives a natural path towards verifying a sampler: first rewrite a given sampler s in an equivalent but simpler form, and then show that this simplified form targets the correct distribution. This is the approach taken in the derivations in § II, which implicitly used several equivalence results – in particular, `let`-reduction and the equivalence of the nested self-product $(s^m)^n$ to the self-product s^{m*n} for any sampler s. In this section, we introduce a relation $\approx$ on programs which justifies this type of reasoning.

Definition IV.1. *We say that two programs* $\Gamma \vdash s : \mathtt{T}$ *and* $\Gamma \vdash t : \mathtt{T}$ *are* equivalent*, notation* $\Gamma \vdash s \approx t : \mathtt{T}$*, if they are related by the smallest congruence relation on well-typed terms containing the rules of Table IV.*[3]

The rules of Table IV employ a number of shorthand conventions for a more concise presentation. We introduce identity functions $\mathtt{id}_{\mathtt{S}} \triangleq \lambda x : \mathtt{S}.\ x : \mathtt{S} \to \mathtt{S}$, constant functions $\mathtt{1}_{\mathtt{S}} \triangleq \lambda x : \mathtt{S}.\ 1 : \mathtt{S} \to \mathtt{R}^+$, function composition $t \circ s \triangleq \lambda x : \mathtt{S}.\ t(s(x)) : \mathtt{S} \to \mathtt{U}$ where $s : \mathtt{S} \to \mathtt{T}, t : \mathtt{T} \to \mathtt{U}$, compositions $\mathtt{f}^{\mathtt{0}} \triangleq \mathtt{id}_{\mathtt{S}} : \mathtt{S} \to \mathtt{S}, \mathtt{f}^{\mathtt{n}} \triangleq \mathtt{f} \circ \mathtt{f}^{\mathtt{n-1}}$ for any $n \in \mathbb{N}$, pointwise products $s \cdot t \triangleq \lambda x : \mathtt{S}, y : \mathtt{T}.\ s(x) * t(y) : \mathtt{S} \times \mathtt{T} \to \mathtt{R}^+$ of real-valued functions $s : \mathtt{S} \to \mathtt{R}^+, t : \mathtt{T} \to \mathtt{R}^+$, and finally Cartesian products $s \times t \triangleq \lambda x : \mathtt{S}, y : \mathtt{T}.\ (s(x), t(y)) : \mathtt{S} \times \mathtt{T} \to \mathtt{S}' \times \mathtt{T}'$ of functions $s : \mathtt{S} \to \mathtt{S}', t : \mathtt{T} \to \mathtt{T}'$.

[3]By *congruence relation*, we mean that $\approx$ is an equivalence relation preserved by all operations in the language. For example, if $\Gamma \vdash s \approx t : \Sigma\,\mathtt{T}$ holds, then $\Gamma \vdash \mathtt{tl}(s) \approx \mathtt{tl}(t) : \Sigma\,\mathtt{T}$ must hold as well, and the same for all operations in the language.

Theorem IV.1. *[10, Appendix A] The rules of Table IV are sound: if* $\Gamma \vdash s \approx t : \mathtt{T}$*, then* $[\![\Gamma \vdash s : \mathtt{T}]\!] = [\![\Gamma \vdash t : \mathtt{T}]\!]$.

The proof is a straightforward exercise in coinductive reasoning and can be found in the Appendix, along with the full list of equivalence rules. The soundness of these rules with respect to operational equivalence then follows from abstraction, though it is also straightforward to show directly.

Recall that an important application of our sampler operations is to provide a formal definition of the *self-product* of samplers, given in (3). It is crucial that our equivalence rules should show that this self-product is well-defined.

Proposition IV.1. *[10, Appendix A] For any* $\Gamma \vdash s : \Sigma\,\mathtt{S}$*,* $m, n \in \mathbb{N}$*, the self-product satisfies* $\Gamma \vdash (s^m)^n \approx s^{mn} : \Sigma\,(\mathtt{S}^{mn})$.

The equivalence rules in Table IV suggest a procedure for simplifying samplers. Consider samplers which have no occurrences of the operation `prng`. Of our remaining sampler operations, we identify two groups: $\{\mathtt{tl}, \mathtt{hd}, \mathtt{wt}, \mathtt{thin}, \otimes\}$ and $\{\mathtt{map}, \mathtt{reweight}\}$. Applying the rules of Table IV, we see that for each combination of operations in the first and second group, there is a rule which enables us to pull the first operation into the body of the second. Therefore, any sampler with no instances of `prng` can be written so that the sampler operations $\mathtt{tl}, \mathtt{hd}, \mathtt{wt}, \mathtt{thin}, \otimes$ are pulled all the way inwards.

Proposition IV.2. *Let* $\Gamma \vdash t : \Sigma\,\mathtt{T}$ *be a sampler which contains no instances of* `prng`*. Applying the rules of Table IV, it follows that we can equivalently rewrite such a sampler in either the form* $\Gamma \vdash \mathtt{map}(f, \mathtt{reweight}(g, \mathtt{map}(f', \ldots, s))) : \Sigma\,\mathtt{T}$ *or* $\Gamma \vdash \mathtt{reweight}(g, \mathtt{map}(f, \mathtt{reweight}(g', \ldots, s))) : \Sigma\,\mathtt{T}$*, i.e. a composition of invocations of* `map` *and* `reweight` *(including the trivial case of zero occurrences of either), where crucially the sampler* s *does not contain the sampler operations* `map` *or* `reweight`.

We can also show that the self-product distributes over the operations `map` self-and `reweight`; such operations are useful for representing the self-product of a composite sampler in a

$$\Gamma, s : \mathtt{S} \vdash (\lambda x : \mathtt{S}.t)(s) \approx t[x \leftarrow s] : \mathtt{T} \qquad \Gamma \vdash \lambda x : \mathtt{S}.t(x) \approx t : \mathtt{S} \to \mathtt{T}$$
$$\Gamma, s : \mathtt{S} \vdash \mathtt{let}\ x = s\ \mathtt{in}\ t \approx (\lambda x : \mathtt{S}.\ t)(s) : \mathtt{T}$$
$$\Gamma \vdash \mathtt{if\ True\ then}\ s\ \mathtt{else}\ t \approx s : \mathtt{T} \qquad \Gamma \vdash \mathtt{if\ False\ then}\ s\ \mathtt{else}\ t \approx t : \mathtt{T}$$
$$\Gamma \vdash \mathtt{fst}((s,t)) \approx s : \mathtt{S} \qquad \Gamma \vdash \mathtt{snd}((s,t)) \approx t : \mathtt{T}$$

$$\Gamma \vdash \mathtt{hd}(\mathtt{map}(s,t)) \approx s(\mathtt{hd}(t)) : \mathtt{T} \qquad \Gamma \vdash \mathtt{wt}(\mathtt{map}(s,t)) \approx \mathtt{wt}(t) : \mathtt{R}^+ \qquad \Gamma \vdash \mathtt{tl}(\mathtt{map}(s,t)) \approx \mathtt{map}(s, \mathtt{tl}(t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash (\mathtt{hd}(s), \mathtt{hd}(t)) \approx \mathtt{hd}(s \otimes t) : \mathtt{S} \times \mathtt{T} \qquad \Gamma \vdash \mathtt{wt}(s) * \mathtt{wt}(t) \approx \mathtt{wt}(s \otimes t) : \mathtt{R}^+ \qquad \Gamma \vdash \mathtt{tl}(s) \otimes \mathtt{tl}(t) \approx \mathtt{tl}(s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{hd}(\mathtt{thin}(n,t)) \approx \mathtt{hd}(t) : T \qquad \Gamma \vdash \mathtt{wt}(\mathtt{thin}(n,t)) \approx \mathtt{wt}(t) : \mathtt{R}^+ \qquad \{\Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,t)) \approx \mathtt{thin}(n, \mathtt{tl}^n(t)) : \Sigma\,\mathtt{T} \mid n \in \mathbb{N}\}$$
$$\Gamma \vdash \mathtt{thin}(1,t) \approx t : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{hd}(\mathtt{prng}(s,t)) \approx t : \mathtt{T} \qquad \Gamma \vdash \mathtt{wt}(\mathtt{prng}(s,t)) \approx 1 : \mathtt{R}^+ \qquad \Gamma \vdash \mathtt{tl}(\mathtt{prng}(s,t)) \approx \mathtt{prng}(s, s(t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{hd}(\mathtt{reweight}(s,t)) \approx \mathtt{hd}(t) : \mathtt{T} \qquad \Gamma \vdash \mathtt{wt}(\mathtt{reweight}(s,t)) \approx s(\mathtt{hd}(t)) * \mathtt{wt}(t) : \mathtt{R}^+ \qquad \Gamma \vdash \mathtt{tl}(\mathtt{reweight}(s,t)) \approx \mathtt{reweight}(s, \mathtt{tl}(t)) : \Sigma\,\mathtt{T}$$

$$\Gamma \vdash \mathtt{thin}(n, \mathtt{thin}(m,t)) \approx \mathtt{thin}(n * m, t) \approx \Sigma\,\mathtt{T} \qquad \Gamma \vdash \mathtt{map}(g, \mathtt{map}(f,t)) \approx \mathtt{map}(g \circ f, t) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{reweight}(g, \mathtt{reweight}(f,t)) \approx \mathtt{reweight}(f \cdot g, t) : \Sigma\,\mathtt{T}$$
$$\{\Gamma \vdash \mathtt{thin}(n, \mathtt{prng}(s,t)) \approx \mathtt{prng}(s^n, t) : \Sigma\,\mathtt{T} \mid n \in \mathbb{N}\} \qquad \Gamma \vdash \mathtt{thin}(n, \mathtt{map}(s,t)) \approx \mathtt{map}(s, \mathtt{thin}(n,t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash s \otimes \mathtt{map}(f,t) \approx \mathtt{map}(\mathtt{id}_\mathtt{S} \times f, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T}) \qquad \Gamma \vdash \mathtt{map}(f,s) \otimes t \approx \mathtt{map}(f \times \mathtt{id}_\mathtt{T}, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash s \otimes \mathtt{reweight}(g,t) \approx \mathtt{reweight}(1_\mathtt{S} \cdot g, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T}) \qquad \Gamma \vdash \mathtt{reweight}(f,s) \otimes t \approx \mathtt{reweight}(f \cdot 1_\mathtt{T}, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{prng}(f,a) \otimes \mathtt{prng}(g,b) \approx \mathtt{prng}(f \times g, (a,b)) : \Sigma\,(\mathtt{S} \times \mathtt{T}) \qquad \Gamma \vdash \mathtt{thin}(n,s) \otimes \mathtt{thin}(n,t) \approx \mathtt{thin}(n, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$

TABLE IV: Rules for sampler equivalence

simpler form whose correctness can then be verified.

Proposition IV.3. *[10, Appendix A] For any mapped sampler* $\Gamma \vdash \mathtt{map}(f,s) : \Sigma\,\mathtt{T}$ *and any* $n \in \mathbb{N}$*, it follows that* $\Gamma \vdash \mathtt{map}(f,s)^n \approx \mathtt{map}(f \times \ldots \times f, s^n) : \Sigma\,(\mathtt{T}^n)$*; for a reweighted sampler* $\Gamma \vdash \mathtt{reweight}(f,s) : \Sigma\,\mathtt{S}$*, it follows that* $\Gamma \vdash \mathtt{reweight}(f,s)^n \approx \mathtt{reweight}(f \cdot \ldots \cdot f, s^n) : \Sigma\,(\mathtt{S}^n)$.

V. Semantic correctness of samplers

The fundamental correctness criterion for a sampler is that it should produce samples which are distributed according to the desired target distribution. This section aims to sketch a simple 'targeting calculus' to compositionally verify this property. We frame this correctness in terms of weak convergence of measures; while other notions of convergence could be used, weak convergence is standard and will suffice for our purposes.

A. The empirical transformation

First, we need to formalise what we mean when we say that a sampler $s : \Sigma\,\mathtt{T}$ *targets* a probability distribution on $[\![\mathtt{T}]\!]$. Given a topological space X, let us write $\mathcal{P}X$ for the space of probability measures on (the Borel σ-algebra generated by) X, equipped with the topology of weak convergence, i.e. $\lim_{n\to\infty} \mu_n = \mu$ in $\mathcal{P}X$ if for any *bounded continuous* map $f : X \to \mathbb{R}$, $\lim_{n\to\infty} \int f\ d\mu_n = \int f\ d\mu$. In fact, $\mathcal{P}$ defines a functor $\mathbf{Top} \to \mathbf{Top}$: if $f : X \to Y$ is a continuous map, then $\mathcal{P}(f) \triangleq f_* : \mathcal{P}X \to \mathcal{P}Y$ is the pushforward map, which is easily shown to be continuous. We do not know if $\mathcal{P}X$ is a CG-space when X is, and in particular we do not know if $\mathcal{P}$ can be given a monad structure on $\mathbf{CG}$. These questions are, however, orthogonal to this work since $\mathcal{P}$ plays no role in the semantics of § III-C.

For any stream $\sigma : \mathbb{N} \to X \times \mathbb{R}^+$ we define $\hat{\sigma}_n \in \mathcal{P}X$ as

$$\hat{\sigma}_n \triangleq \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_2(\sigma(i))}{\sum_{j=1}^{n} \pi_2(\sigma(j))} \delta_{\pi_1(\sigma(i))},$$

the *empirical distribution* based on the first n (weighted) samples of σ. We also define $\mathcal{P}_\perp X \triangleq \mathcal{P}X + 1$, where $1 = \{\perp\}$ is the terminal object and $+$ the coproduct in $\mathbf{Top}$.

Definition V.1. *The* empirical measure transformation *is the* $\mathbf{Top}^{\mathrm{obj}}$*-collection of maps*

$$\varepsilon_X : (X \times \mathbb{R}^+)^{\mathbb{N}} \to \mathcal{P}_\perp X, \sigma \mapsto \begin{cases} \lim_{n\to\infty} \hat{\sigma}_n & \textit{if it exists} \\ \perp \in 1 & \textit{else} \end{cases}$$

The empirical measure transformation cannot be natural, as the following example shows.

Example V.1. *Let* $\sigma : \mathbb{N} \to X$ *be a diverging unweighted sampler on* X*, i.e.* $\varepsilon_X(\sigma) = \perp$*, and consider the map to the terminal object* $! : X \to 1$*. Then* $\varepsilon_1(! \circ \sigma) = \varepsilon_1(\perp, \perp, \ldots) = \delta_\perp$*, but* $\mathcal{P}_\perp(!)(\varepsilon_X(\sigma)) = \mathcal{P}_\perp(!)(\perp) = \perp$.

However, if a sampler does define a probability measure via ε, this is preserved by continuous maps.

Proposition V.1. *[10, Appendix A] Let* $\sigma : \mathbb{N} \to X \times \mathbb{R}^+$ *and* $f : X \times \mathbb{R}^+ \to Y \times \mathbb{R}^+$ *be continuous. If* $\varepsilon_X(\sigma) = \mu$*, then* $\varepsilon_Y(f \circ \sigma) = f_*(\mu)$.

It is tempting to try to generalise this nice property of continuous maps to more general maps – for example, measurable maps. The following example shows that this is not possible.

Example V.2. *Let* $X = [0,1]$ *and* $\sigma : \mathbb{N} \to [0,1]$ *denote any unweighted sampler such that* $\varepsilon(\sigma)$ *is the Lebesgue measure on* $[0,1]$*. Now consider the map* $f : [0,1] \to \{0,1\}$ *defined by* $f(x) = 1$ *if* $x = \sigma(i)$ *for some* i *and 0 else. This function is the indicator function of a countable, therefore closed, set, and so is Borel-measurable. On the one hand we have that* $\varepsilon(f \circ \sigma) = \delta_1$ *since* $f \circ \sigma$ *is the constant stream on ones, but on the other we have* $f_*(\varepsilon(\sigma))(1) = \varepsilon(\sigma)(f^{-1}(1)) = 0$ *since only a countable set is mapped onto 1 by* f.

Even for functions with finitely many discontinuities, it is impossible to extend the class of functions for which Prop. V.1

holds. However, the semantic framework adopted in § III-C allows us to bypass this problem altogether. We illustrate these two points by revisiting Ex. III.1.

Example V.3. *Consider the sampler* $\mathtt{s} \triangleq \mathtt{prng}(\lambda x : \mathtt{R} \,.\, x/2, 1)$ *and the term* $\mathtt{p} \triangleq \mathtt{if}\ x = 0\ \mathtt{then}\ 1\ \mathtt{else}\ -1$ *of Ex. III.1. Assume first that* $\mathbb{R}$ *is equipped with its standard topology, i.e. that* $[\![\mathtt{p}]\!]$ *is not continuous at 0. Since* $\mathbb{R}$ *is a metric space we can use the Portmanteau Lemma and rephrase weak convergence by limiting ourselves to bounded Lipschitz functions. It is then easy to show that* $\varepsilon([\![\mathtt{s}]\!]) = \delta_0$*: letting* $f : \mathbb{R} \to \mathbb{R}$ *be bounded Lipschitz, we have*

$$\lim_{n\to\infty} \left| \int f \; d\widehat{[\![\mathtt{s}]\!]}_n - \int f \; d\delta_0 \right| = \lim_{n\to\infty} \left| \frac{1}{n} \sum_{i=1}^{n} f\left(\frac{1}{2^i}\right) - f(0) \right| \leq \lim_{n\to\infty} \left| \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2^i} \right| \leq \lim_{n\to\infty} \frac{2}{n} = 0$$

Prop. V.1 now fails on $[\![\mathtt{p}]\!]$*, since* $\varepsilon([\![\mathtt{p}]\!] \circ [\![\mathtt{s}]\!]) = \varepsilon(-1, -1, \ldots) = \delta_{-1} \neq \mathcal{P}([\![\mathtt{p}]\!])(\varepsilon([\![\mathtt{s}]\!])) = [\![\mathtt{p}]\!]_* (\delta_0) = \delta_1$.

Let us now equip $\mathbb{R}$ *with the topology given by type-checking* p *as described in Ex. III.1. This makes* $[\![\mathtt{p}]\!]$ *bounded and continuous, and we therefore no longer have* $\varepsilon([\![\mathtt{s}]\!]) = \delta_0$*; indeed* $\lim_n \int [\![\mathtt{p}]\!] \; d\widehat{[\![\mathtt{s}]\!]}_n = -1 \neq [\![\mathtt{p}]\!](0) = 1$*. In fact we now have* $\varepsilon([\![\mathtt{s}]\!]) = \bot$*, i.e.* s *is no longer a sampler targeting anything for this topology, which prevents the failure of Prop. V.1 on* $[\![\mathtt{p}]\!]$.

This example also shows that our semantics has provided us with many more morphisms satisfying Prop. V.1 than would have been the case had we only considered programs which are continuous w.r.t. the usual topology on the denotation of types. Our semantics allows us to push forward a sampler s through any piecewise continuous function, except in the narrow case where this function has a point of discontinuity which is asymptotically assigned positive mass by s. We illustrate this further in the next example.

Example V.4. *Consider the sampler of Ex. V.3, but now let* $\mathtt{p} \triangleq \mathtt{if}\ x = 2\ \mathtt{then}\ 1\ \mathtt{else}\ -1$ *instead. To make this function continuous, our semantics adds the open set* $\{2\}$ *to the usual topology of* $\mathbb{R}$*. This does not interfere with the derivation that* $\varepsilon([\![\mathtt{s}]\!]) = \delta_0$*, since we can write* $\int f \; d\widehat{[\![\mathtt{s}]\!]}_n = \int_{\{2\}^c} f \; d\widehat{[\![\mathtt{s}]\!]}_n + \int_{\{2\}} f \; d\widehat{[\![\mathtt{s}]\!]}_n = \int_{\{2\}^c} f \; d\widehat{[\![\mathtt{s}]\!]}_n$*, and similarly for* δ_0*. Because the discontinuity of* $[\![\mathtt{p}]\!]$ *is not assigned any mass by* δ_0*, the topology on* $\mathbb{R}$ *making* $[\![\mathtt{p}]\!]$ *continuous no longer prevents* $\varepsilon([\![\mathtt{s}]\!])$ *from converging, and we can therefore safely push* s *forward through* p *using* map.

B. Calculus for asymptotic targeting

Definition V.2. *We will say that the sampler* $\Gamma \vdash s : \Sigma\,\mathtt{S}$ asymptotically targets*, or simply* targets*, the continuous map* $\mu : [\![\Gamma]\!] \to \mathcal{P}\,[\![\mathtt{S}]\!]$ *if for every* $\gamma \in [\![\Gamma]\!]$*,*

$$\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s]\!]\,(\gamma) = \mu(\gamma).$$

In particular, $\widehat{[\![s]\!]\,(\gamma)}_n$ *always converges as* $n \to \infty$*; diverging samplers do not target anything.*

We will say that $\Gamma \vdash s : \Sigma\,\mathtt{S}$ *is* K-equidistributed *with respect to the morphism* $\mu : [\![\Gamma]\!] \to \mathcal{P}\,[\![\mathtt{S}]\!]$ *if for every* $\gamma \in [\![\Gamma]\!]$*,* $\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s^K]\!]\,(\gamma) = \mu^K(\gamma)$*, where the self-product* s^K *is defined in* (3) *and* $\mu^K(\gamma) \in \mathcal{P}([\![\mathtt{S}]\!]^K)$ *is the* K*-fold product of the measure* $\mu(\gamma)$ *with itself.*

We introduce in Table V a relation $\rightsquigarrow$ which is sound with respect to asymptotic targeting; this is the relation which was used in the proofs of § II. That is, if $\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$, then s is a parametrised sampler on S which asymptotically targets a parametrised distribution μ on $[\![\mathtt{S}]\!]$. Here, we use Greek lower case letters μ, ν to represent (parametrised) distributions in order to emphasise their role as meta-variables, used only in the context of the targeting calculus, and not within the language itself. In the rule for `reweight`, we abbreviate the operation of reweighting a measure μ on X by $f : X \to \mathbb{R}_{\geq 0}$ as the product $(f \cdot \mu)(A) = \frac{\int_A f(x)\, d\mu(x)}{\int_X f(x)\, d\mu(x)}$, assuming that the integral in question is finite and nonzero.

Table V incorporates a rule for building samplers from scratch as pseudo-random number generators defined by a deterministic endomap $t : \mathtt{T} \to \mathtt{T}$ and an initial value $x : \mathtt{T}$ via $\mathtt{prng}(t, x) : \Sigma\,\mathtt{T}$.[4] However, Table V also incorporates a set of 'axioms' for built-in samplers $\mathtt{rand}_i$, each targeting distributions $\mu_i \in \mathcal{P}\,[\![\mathtt{T}_i]\!]$; in some settings, it may be defensible to assume access to 'truly random' samplers which generate samples using a physical process.

The reader might wonder why Table V does not have a rule for the `thin` operation: after all, if σ is a sampler targeting a distribution μ, then only keeping every n samples should produce a good sampler as well. Whilst this is true of the sequences produced by 'true' i.i.d. samplers (for example physical samplers) with probability 1, this rule is in general not sound, as the following simple example shows.

Example V.5. *Consider the sampler on* $\{0, 1\}$ *defined by the program* $\mathtt{prng}(\lambda x : \mathtt{R} \,.\, 1 - x, 0)$*. This sampler, which generates the unweighted samples* $(0, 1, 0, 1, \ldots)$*, targets the uniform Bernoulli distribution; however, applying* $\mathtt{thin}(2, -)$ *to it yields a sampler which targets the Dirac measure* δ_0*.*

This example highlights the fact that samplers can be manifestly non-random, and yet from the perspective of inference – that is to say, from the perspective of the topology of weak convergence – target bona fide probability distributions.

Theorem V.1. *[10, Appendix A] Targeting* $\rightsquigarrow$ *is sound: if* $\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$*, then* $\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s]\!] = \mu$.

We saw in § V-A how our (sub-)typing system can be used to safely pushforward samplers through maps which are only piecewise continuous. Our typing system also allows us to add additional constraints to samplers. Specifically, we can ensure that a sampler visits certain subsets infinitely often.

[4] Applying this rule requires showing that the initial point of the sampler is *typical*; a point $x \in [\![\mathtt{T}]\!]$ is called *typical* if it belongs to the μ-mass 1 subset $X \subseteq [\![\mathtt{T}]\!]$ in which the ergodic theorem holds [19, Theorem 9.6].

$$\frac{}{\Gamma_i \vdash \mathtt{rand}_i : \Sigma\,\mathtt{T}_i \rightsquigarrow \mu_i}\ i \in I \qquad \frac{\Gamma \vdash s \approx t : \Sigma\,\mathtt{T} \quad \Gamma \vdash s : \Sigma\,\mathtt{T} \rightsquigarrow \mu}{\Gamma \vdash t : \Sigma\,\mathtt{T} \rightsquigarrow \mu} \qquad \frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu}{\Gamma \vdash \mathtt{tl}(s) : \Sigma\,\mathtt{S} \rightsquigarrow \mu}$$

$$\frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu \quad \Gamma \vdash f : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash \mathtt{map}(f, s) : \Sigma\,\mathtt{T} \rightsquigarrow \gamma \mapsto (\llbracket f \rrbracket(\gamma))_* \mu(\gamma)} \qquad \frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu \quad \Gamma \vdash f : \mathtt{S} \to \mathtt{R}^+}{\Gamma \vdash \mathtt{reweight}(f, s) : \Sigma\,\mathtt{S} \rightsquigarrow \gamma \mapsto \llbracket f \rrbracket(\gamma) \cdot \mu(\gamma)}\ \int_{\llbracket S \rrbracket} \llbracket f \rrbracket(\gamma)\, d\mu(\gamma) \in (0, \infty)$$

$$\frac{}{\Gamma \vdash \mathtt{prng}(f, x) \rightsquigarrow \mu}\ \llbracket f \rrbracket : \llbracket \mathtt{T} \rrbracket \to \llbracket \mathtt{T} \rrbracket \text{ ergodic w.r.t. } \mu, x \text{ typical}$$

TABLE V: Rules for asymptotic targeting

$$\dfrac{\dfrac{\dfrac{\vdash \mathtt{rand}^3 : \Sigma\,\mathtt{S} \rightsquigarrow U^3 \quad \vdash \mathtt{id_R} \times \mathtt{box} : \mathtt{S} \to \mathtt{T}}{\vdash \mathtt{map}(\mathtt{id_R} \times \mathtt{box}, \mathtt{rand}^3) : \Sigma\,\mathtt{T} \rightsquigarrow \llbracket \mathtt{id_R} \times \mathtt{box} \rrbracket_* (U^3) = U \otimes \mathrm{N}(0,1)} \quad \vdash \mathtt{accept} : \mathtt{T} \to \mathtt{R}^+}{\vdash \mathtt{reweight}(\mathtt{accept}, \mathtt{joint}) : \Sigma\,\mathtt{T} \rightsquigarrow \llbracket \mathtt{accept} \rrbracket \cdot (U \otimes \mathrm{N}(0,1))} \quad \vdash \mathtt{produce} : \mathtt{T} \to \mathtt{R}}{\vdash \mathtt{map}(\mathtt{produce}, \mathtt{reweight}(\mathtt{accept}, \mathtt{joint})) : \Sigma\,\mathtt{R}^+ \rightsquigarrow \llbracket \mathtt{produce} \rrbracket_* (\llbracket \mathtt{accept} \rrbracket \cdot (U \otimes \mathrm{N}(0,1))) = \Gamma(\alpha, 1)}$$

Fig. 5: Validity of Marsaglia sampler in Listing 4

Proposition V.2. *[10, Appendix A] Assume* $\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$, $\mathtt{S} \triangleleft \mathtt{T}$ *and* $\llbracket \mathtt{T} \rrbracket$ *second-countable; then* $\Gamma \vdash \mathtt{map}(\lambda x : \mathtt{S}.\mathtt{cast}\langle \mathtt{T} \rangle x, s) : \Sigma\,\mathtt{T}$ *targets the same measure* μ *on* $\mathtt{T}$*. Moreover, if* $\llbracket \mathtt{T} \rrbracket$ *is metrizable, if* U *is in the topology of* $\llbracket \mathtt{S} \rrbracket$ *but not* $\llbracket \mathtt{T} \rrbracket$ *and* $\mu(\partial_{\mathtt{T}} U) > 0$ *(where* $\partial_{\mathtt{T}}$ *denotes the boundary in* $\llbracket \mathtt{T} \rrbracket$*) then* s *must visit* $\partial_{\mathtt{T}} U$ *i.o. (infinitely often).*

Example V.6. *Suppose we want* $s : \Sigma\,\mathtt{R} \rightsquigarrow \mathrm{Bern}(1/2)$*. A sampler alternating between the sampler* $z \triangleq \mathtt{prng}(\lambda x : \mathtt{R} \,.\, x/2, 1)$ *of Ex. V.3 and its shifted version* $\mathtt{map}(\lambda x : \mathtt{R} \,.\, 1 + x, z)$ *will satisfy the condition, but will never visit* 0 *or* 1*! We can use the previous result to enforce that a sampler* s *targeting* $\mathrm{Bern}(1/2)$ *should visit* 0 *i.o. by constructing* s *in such a way that it has type* $\Sigma\big((x,0)^{-1}\mathtt{Neq} + (x,0)^{-1}\mathtt{Eq}\big)$ *(see Ex. III.1). We can, in the same manner, enforce that a sampler* s' *targeting* $\mathrm{Bern}(1/2)$ *visits* 1 *i.o. Finally, using the last two rules of Fig. 4b which build the coarsest common refinement of two topologies, we can combine* s *and* s' *to create a sampler targeting* $\mathrm{Bern}(1/2)$ *and guaranteed to visit* $0, 1$ *i.o.*

Example V.7. *We conclude with an example highlighting sampler compositionality by chaining two well-known sampling algorithms. Consider the following program:*

```
let box = λu : R⁺×R⁺ . sqrt(-2*log(fst(u))) *
    cos(2*pi*snd(u)) in
let joint = map(id_R×box, rand³) in
let d = α - 1/3 in
let c = 1/(3*sqrt(d)) in
let accept = λ(u, x) : T .
  let v = (1+c*x)^3 in
    if v > 0 and log(u) < x^2 + d - d*v + d*
    log(v) then 1 else 0 in
let produce = λz : T . d*(1+c*fst(cast⟨R×R⟩z))
    ^3 in
  map(proj, reweight(accept, joint))
```

Listing 4: Marsaglia sampler for gamma random variables

First, the Box-Muller technique is a well-known technique for generating standard normal random variates using two independent uniform samples; its verification using the `map` *rule of Table V is straightforward. We then form a joint sampler consisting of independent uniform and Gaussian samples, and then consume both of these samples to generate gamma-distributed random variables* $z \sim \Gamma(\alpha, 1)$*, for shape* $\alpha \geq 1$*, using a well-known rejection sampling technique; see [28] for a proof. The validity of this sampler is sketched in Fig. 5; we omit some types for brevity.*

VI. Discussion

We have presented a 'probabilistic' language designed to compositionally construct *samplers*. We have given this language an intuitive operational semantics and a denotational semantics in the category of CG-spaces, and shown that the two are equivalent for closed samplers. This denotational universe is sufficiently rich to interpret sampler types coinductively, and to interpret functions which are only piecewise-continuous on the standard topologies given by the type system (§ III-C).

With the support of this language, we have shown how to compositionally reason about the validity of sampler constructions, an essential aspect in the practice of probabilistic programming. Our approach draws on a sound equational system to reason about equivalent ways of constructing the same sampler (§ IV) and a sound system for reasoning about semantic correctness (§ V).

What distinguishes our approach is that we are in effect providing a purely deterministic semantics for probabilistic programs. This approach is much closer to the practice of probabilistic programming, in which samples and samplers are the most important concrete entities; this distinction between samplers and the measures they target is necessary in order to support pseudo-random number generation. Measure-theoretic entities, which have typically been a part of the denotational semantics of probabilistic languages, e.g. [21], [16], [12], [38], [8], instead take a meta-theoretic role as verification criteria.

Two commonly-used schemes for producing samplers are missing from our calculus: Markov chain Monte Carlo methods and resampling techniques, as applied in e.g. particle filters. We consider adding these constructions to our language, together with the corresponding correctness proofs in our targeting calculus, to be future work.

References

[1] Andrieu, C., Doucet, A., and Holenstein, R. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72*, 3 (2010), 269–342.

[2] Bagnall, A., Stewart, G., and Banerjee, A. Formally verified samplers from probabilistic programs with loops and conditioning, 2023.

[3] Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P. A., Horsfall, P., and Goodman, N. D. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res. 20* (2019), 28:1–28:6.

[4] Blackwell, A., Kohn, T., Erwig, M., Baydin, A. G., Church, L., Geddes, J., Gordon, A., Gorinova, M., Gram-Hansen, B., Lawrence, N., Mansinghka, V., Paige, B., Petricek, T., Robinson, D., Sarkar, A., and Strickson, O. Usability of probabilistic programming languages. In *Psychology of Programming Interest Group Annual Workshop (PPIG 2019), Newcastle, UK, 28–30 August 2019* (2019).

[5] Borgström, J., Dal Lago, U., Gordon, A. D., and Szymczak, M. A lambda-calculus foundation for universal probabilistic programming. *ACM SIGPLAN Notices 51*, 9 (2016), 33–46.

[6] Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

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

[8] Dahlqvist, F., and Kozen, D. Semantics of higher-order probabilistic programs with conditioning. *Proceedings of the ACM on Programming Languages 4*, POPL (2019), 1–29.

[9] Dahlqvist, F., Kozen, D., and Silva, A. *Semantics of Probabilistic Programming: A Gentle Introduction*. Cambridge University Press, 2020, pp. 1–42.

[10] Dahlqvist, F., Silva, A., and Smith, W. Deterministic stream-sampling for probabilistic programming: semantics and verification. *arXiv preprint* (2023).

[11] Ehrhard, T., Pagani, M., and Tasson, C. The computational meaning of probabilistic coherence spaces. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science* (2011), IEEE, pp. 87–96.

[12] Ehrhard, T., Pagani, M., and Tasson, C. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *Proceedings of the ACM on Programming Languages 2*, POPL (2017), 1–28.

[13] Ehrhard, T., Pagani, M., and Tasson, C. Full abstraction for probabilistic pcf. *Journal of the ACM (JACM) 65*, 4 (2018), 1–44.

[14] Faggian, C., and Della Rocca, S. R. Lambda calculus and probabilistic computation. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (2019), IEEE, pp. 1–13.

[15] Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. Church: a language for generative models. *arXiv preprint arXiv:1206.3255* (2012).

[16] Heunen, C., Kammar, O., Staton, S., and Yang, H. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (2017), IEEE, pp. 1–12.

[17] Hoyrup, M., and Rojas, C. Computability of probability measures and martin-löf randomness over metric spaces. *Information and Computation 207*, 7 (2009), 830–847.

[18] Huang, D., Morrisett, G., and Spitters, B. *Application of Computable Distributions to the Semantics of Probabilistic Programs*. Cambridge University Press, 2020, p. 75–120.

[19] Kallenberg, O. *Foundations of modern probability*. Springer, 1997.

[20] Keremedis, K., Özel, C., Piękosz, A., Shumrani, M. A., and Wajch, E. Compact complement topologies and k-spaces. *arXiv preprint arXiv:1806.10177* (2018).

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

[22] Lago, U. D., and Zorzi, M. Probabilistic operational semantics for the lambda calculus. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications 46*, 3 (2012), 413–450.

[23] L'Ecuyer, P. History of uniform random number generation. In *2017 Winter Simulation Conference (WSC)* (2017), pp. 202–230.

[24] Leobacher, G., and Pillichshammer, F. *Introduction to quasi-Monte Carlo integration and applications*. Springer, 2014.

[25] Lewis, L. G. *The stable category and generalized Thom spectra. Appendix A*. PhD thesis, University of Chicago, Department of Mathematics, 1978.

[26] Lunn, D., Thomas, A., Best, N., and Spiegelhalter, D. Winbugs - a bayesian modeling framework: Concepts, structure and extensibility. *Statistics and Computing 10* (10 2000), 325–337.

[27] Mansinghka, V., Selsam, D., and Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv e-prints* (Mar. 2014), arXiv:1404.0099.

[28] Marsaglia, G., and Tsang, W. W. A simple method for generating gamma variables. *ACM Trans. Math. Softw. 26*, 3 (sep 2000), 363–372.

[29] Martin-Löf, P. The definition of random sequences. *Information and control 9*, 6 (1966), 602–619.

[30] Matsumoto, M., and Nishimura, T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul. 8*, 1 (1998), 3–30.

[31] McCord, M. C. Classifying spaces and infinite symmetric products. *Transactions of the American Mathematical Society 146* (1969), 273–298.

[32] Park, S., Pfenning, F., and Thrun, S. A probabilistic language based upon sampling functions. *ACM SIGPLAN Notices 40*, 1 (2005), 171–182.

[33] Robert, C., and Casella, G. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

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

[35] Steenrod, N. E. A convenient category of topological spaces. *Michigan Mathematical Journal 14*, 2 (1967), 133–152.

[36] Thomas, A. Bugs: a statistical modelling package. *RTA/BCS Modular Languages Newsletter 2* (1994), 36–38.

[37] Trevisan, L., and Vadhan, S. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* (USA, 2000), FOCS '00, IEEE Computer Society, p. 32.

[38] Vákár, M., Kammar, O., and Staton, S. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages 3*, POPL (2019), 1–29.

[39] Vigna, S. Further scramblings of marsaglia's xorshift generators. *Journal of Computational and Applied Mathematics 315* (2017), 175–181.

[40] Wood, F., van de Meent, J. W., and Mansinghka, V. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics* (2014), pp. 1024–1032.