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 have¹; 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], [8], [4]. 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

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

¹See [5] for an overview of probabilistic programming languages, [7] for a historical overview of the Monte Carlo method, and [23] for a history of pseudorandom number generation.

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], [9]. 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], [6], [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], [6], [10], [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

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 $(x_0, x_1, ...)$ 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) - xwill 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 properties². 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 reason-

ing 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 {True, 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 $v: \Sigma B$ – which, given another Boolean-valued sampler flip: ΣB 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 $(b_1,b_2)=$ (False, True) and $(b_1,b_2)=$ (True, 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 .

²This remark is in reference to the related field of quasi-Monte Carlo techniques, outlined in [24].

```
\frac{\vdash \mathtt{flip}^2 : \Sigma(\mathtt{B} \times \mathtt{B}) \leadsto \mathrm{Ber}(p)^2 \quad \vdash \mathtt{choice} : \mathtt{B} \times \mathtt{B} \to \mathtt{R}^+}{\vdash \mathtt{reweight}(\mathtt{choice}, \mathtt{flip}^2) : \Sigma(\mathtt{B} \times \mathtt{B}) \leadsto \llbracket \mathtt{choice} \rrbracket \cdot \mathrm{Ber}(p)^2 \quad \vdash \mathtt{proj} : \mathtt{B} \times \mathtt{B} \to \mathtt{B}} \\ \vdash \mathtt{map}(\mathtt{proj}, \mathtt{reweight}(\mathtt{choice}, \mathtt{flip}^2)) : \Sigma\mathtt{B} \leadsto \llbracket \mathtt{proj} \rrbracket_* (\llbracket \mathtt{choice} \rrbracket \cdot \mathrm{Ber}(p)^2) = \mathrm{Ber}(1/2)
```

Fig. 1: Validity of von Neumann extractor

$$\frac{\vdash \mathtt{rand}^2 : \mathtt{\Sigma} \, (\mathtt{R} \times \mathtt{R}) \leadsto U^2 \quad \vdash \mathtt{plus} : \mathtt{R} \times \mathtt{R} \to \mathtt{R}}{\vdash \mathtt{map}(\mathtt{plus}, \mathtt{rand}^2) : \mathtt{\Sigma} \, \mathtt{R} \leadsto \llbracket \mathtt{plus} \rrbracket_* \, U^2} \quad \vdash \mathtt{phi} : \mathtt{R} \to \mathtt{R}^+} \\ \vdash \mathtt{reweight}(\mathtt{phi}, \mathtt{map}(\mathtt{plus}, \mathtt{rand}^2)) : \mathtt{\Sigma} \, \mathtt{R} \leadsto \llbracket \mathtt{phi} \rrbracket \cdot (\llbracket \mathtt{plus} \rrbracket_* \, U^2) = P$$

Fig. 2: Validity of importance sampling in Listing 2

Note the appearance of \mathtt{flip}^2 in the von Neumann extractor; this is the 'self-product' of the sampler \mathtt{flip} . Where $\mathtt{flip}: \Sigma B$ produces Boolean-valued samples, $\mathtt{flip}^2: \Sigma (B \times B)$ produces samples which are pairs of Booleans. Given a sampler $t: \Sigma T$ of type T, the self-product $t^2: \Sigma (T \times T)$ is a sampler whose elements are adjacent samples from T; the same construction, detailed in § III, is easily extended to arbitrary self-powers $t^K: \Sigma (T^K)$.

The main application of our language is to serve as a setting for the formal verification of its samplers. Let $v : \Sigma B$ be the von Neumann extractor defined in Listing 1; the task of verifying v is the task of showing that v 'targets' the uniform distribution Ber (1/2) – meaning, informally, that in the limit of increasing sample size, v generates unbiased flips. In § V-B, we define a relation \rightsquigarrow between samplers and distributions which reifies this notion: we read $\vdash v : \Sigma B \rightsquigarrow$ Ber (1/2) as 'the sampler v targets the measure 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 v targets Ber (1/2), to assume that flip targets Ber(p)for some $p \in (0,1)$. Instead, we are required to make the stronger assumption that flip² targets $Ber(p)^2$: in essence, that adjacent samples from 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 $\{0, \dots, N-1\}$ is K-equidistributed with respect to the uniform distribution is the assertion that all K-length words $w \in \{0, ..., N-1\}^K$ 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 v is equivalent to the simplified sampler $map(proj, reweight(choice, flip^2))$ in a context in which access to a Boolean-typed sampler flip is assumed, where proj and choice are defined as in Listing 1. We write this equivalence as $flip: \Sigma B \vdash v \approx map(proj, reweight(choice, flip^2)): \Sigma B$; this equivalence relation is discussed in § IV, and is shown in this

particular case using the let-binding rule of Table IV.

Having rewritten v in this way, and using the hypothesis of 2-equidistribution $\vdash \mathtt{flip}^2 : \mathtt{\Sigma} \, \mathtt{Ber}(p)^2$, we derive our conclusion in Fig. 1 by applying the rules from § V-B corresponding to the sampler operations reweight and map. These rules show us that v targets the measure $[\![\mathtt{proj}]\!]_*$ ($[\![\mathtt{choice}]\!]_*$ Ber $(p)^2$). (Here, as we will discuss in § V-B, $[\![f]\!]_*$ μ 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 Ber(1/2), the uniform measure on B; this is straightforward. It is easily seen first that $[\![\mathtt{choice}]\!] \cdot \mathtt{Ber}(p)^2$ assigns probability 1/2 to the samples (True, False) and (False, True) and zero probability to all other samples; the desired result then follows by observing that the function 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], [7] 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 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 rand has the necessary independence property of 2-equidistribution, we

```
\frac{\vdash \mathsf{tri} \otimes \mathsf{rand} : \Sigma \, \mathsf{T} \leadsto \mathsf{Tri} \otimes U \quad \vdash \mathsf{accept} : \mathsf{T} \to \mathsf{R}^+}{\vdash \mathsf{reweight}(\mathsf{accept}, \mathsf{tri} \otimes \mathsf{rand}) : \Sigma \, \mathsf{T} \leadsto [\![\mathsf{accept}]\!] \cdot (\mathsf{Tri} \otimes U) \quad \vdash \mathsf{proj} : \mathsf{T} \to \mathsf{R}}\vdash \mathsf{map}(\mathsf{proj}, \mathsf{reweight}(\mathsf{accept}, \mathsf{tri} \otimes \mathsf{rand})) : \Sigma \, \mathsf{R} \leadsto [\![\mathsf{proj}]\!]_* \left([\![\mathsf{accept}]\!] \cdot (\mathsf{Tri} \otimes U)\right) = P
```

Fig. 3: Validity of rejection sampling in Listing 3

sum two draws from 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.

```
let phi = \lambda x : R . 1/sqrt(2*pi) * exp(-1/2*(3-x)*(3-x))
in let plus = \lambda u : R × R . fst(u) + snd(u)
in reweight(phi, map(plus, rand<sup>2</sup>))
```

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 $[\![phi]\!] \cdot ([\![plus]\!]_* U^2)$. 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 \S 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 = \lambda x : R . 1/sqrt(2*pi) * exp(-1/2*(3-x)*(3-x)) in let accept = \lambda(u,v) : T . if v \le \text{phi}(u) * \text{sqrt}(2*pi) then 1 else 0 in let proj = \lambda z : T. fst(cast\langle R \times R \rangle(z)) in map(proj, reweight(accept, tri \otimes 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 $[\![\operatorname{proj}]\!]_*$ ($[\![\operatorname{accept}]\!]\cdot(\operatorname{Tri}\otimes U)$). 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 $R \times R \to R^+$, instead has type $T \to R^+$. Correspondingly, the product $tri \otimes rand$ must be assumed to produce samples of type T, rather than $R \times R$, and proj must accepts inputs of type T rather than pairs $R \times R$. The nature of this type T, a *subtype* of $R \times R$, 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 the Appendix, Figs. 6 and 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

$$\{N, R, R^+\} \cup \{f^{-1}(i) \mid f \in \{\le, <, \ge, >, =, \ne\}, i = 0, 1\}.$$

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 B $\triangleq 1+1$ will be treated as a ground type.

The only unusual type constructors are the *pullback types* ${}_s\mathrm{T}_t$ which – as the name suggests – will be interpreted as pullbacks (in fact inverse images), and the *sampler types* ${}^{\Sigma}\mathrm{T}$ which will be defined as the coinductive (stream) types defined by the (syntactic) functors $\mathrm{T} \times \mathrm{R}^+ \times -$. 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 $f: T \to G$, 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 $+: R \times R \to R$, but others will be discontinuous, such as the comparison operators $\{\leq, <, \geq, >, =, \neq\}: R \times R \to B$. 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

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

$$\mathbf{S},\mathbf{T}::=\mathbf{G}\in \text{Ground}\mid\mathbf{1}\mid\mathbf{S}\times\mathbf{T}\mid\mathbf{S}+\mathbf{T}\mid_{s}\mathbf{T}_{t}\mid\mathbf{S}\rightarrow\mathbf{T}\mid\mathbf{\Sigma}\mathbf{T}\qquad s,t:\mathbf{T}$$

$$(a) \text{ Type grammar}$$

$$\frac{1}{f^{-1}(0)+f^{-1}(1)\triangleleft\mathbf{R}\times\mathbf{R}}\quad f\in\{<,\leq,>,\geq,=,\neq\} \qquad \frac{1}{s}$$

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

Fig. 4: Grammars and subtyping rules

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

- The operation 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 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 reweight(f,t) applies the reweighting scheme f to the sampler t to form a new sampler.
- Given a sampler t, the operation hd(t) returns the first sample produced by t, wt(t) the weight of the first sample produced by t, and tl(t) returns the sampler t but with its first sample-weight pair dropped.
- The operation 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 T which is continuous w.r.t. a topology τ on the denotation of T, to a computation using a variable of type T but which is continuous w.r.t. to a finer topology $\tau' \supset \tau$ on the denotation of 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

$$t^{-1}(\mathbf{T}_i) \triangleq_{\mathbf{cast}\langle \mathbf{T}\rangle_{\mathbf{in}_i(x)}} \mathbf{T}_t \quad \text{where } x : \mathbf{T}_i$$
 (1)

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

$$S_i \cap S_j' \triangleq \underset{\text{cast}(S) \text{in}_i(x_i)}{\text{S}_{\text{cast}(S) \text{in}_j(x_i')}} \quad \text{where } x_i : S_i, x_j' : S_j'$$

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 $t \in \operatorname{Expr}$ and a number of samples $N \in \mathbb{N}$, and the right side is a *value* $v \in \operatorname{Value}$, i.e. a term generated by the grammar

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

The rules of this big-step operational semantics, shown in full in the Appendix, 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

$$\frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash g : \mathsf{G}} \ g \in \llbracket \mathsf{G} \rrbracket \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma, x : \mathsf{T}, \Delta \vdash x : \mathsf{T}} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash f(t) : \mathsf{G}} \ \operatorname{Func} \ni f : \mathsf{T} \to \mathsf{G} \qquad \frac{\Delta \vdash t : \mathsf{S}}{\Gamma \vdash \operatorname{cast} \langle \mathsf{T} \rangle t : \mathsf{T}} \ \operatorname{S} \lhd \mathsf{T}, \Gamma \lhd \Delta$$

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

$$\frac{\Gamma \vdash s : \mathsf{S}}{\Gamma \vdash (s, t) : \mathsf{S}} \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash (s, t) : \mathsf{S}} \qquad \frac{\Gamma \vdash t : \mathsf{S} \times \mathsf{T}}{\Gamma \vdash \operatorname{sind}(t) : \mathsf{T}} \qquad \frac{\Gamma, \mathsf{x} : \mathsf{S} \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} x = s : n : t : \mathsf{T}}$$

$$\frac{\Gamma, \mathsf{x} : \mathsf{S} \vdash t : \mathsf{T}}{\Gamma \vdash \mathsf{L}(s) : \mathsf{T}} \qquad \frac{\Gamma \vdash t : \mathsf{S} \to \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} \qquad \frac{\Gamma \vdash t : \mathsf{T}_j}{\Gamma \vdash \operatorname{tot} y} : \mathsf{J} = n$$

$$\frac{\Gamma \vdash t : \mathsf{T}_j}{\Gamma \vdash \operatorname{tot} x = s : \mathsf{T}} \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} x : \mathsf{T}}$$

$$\frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T}$$

$$\frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash t : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \operatorname{tot} y} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T}}{\Gamma \vdash \mathsf{T}} : \mathsf{T}}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash \mathsf{T}}$$

TABLE I: Typing rules

rules readable, we also introduce the shorthand $(t, N) \rightarrow ((v_1, w_1), \dots, (v_N, w_N))$ to denote the N reductions

$$(hd(t), wt(t)) \to (v_1, w_1),$$

 $(hd(tl(t)), wt(tl(t))) \to (v_2, w_2), \dots,$
 $(hd(tl^{N-1}(t)), wt(tl^{N-1}(t))) \to (v_N, w_N).$

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. [Appendix A] If $\vdash s : \Sigma S$ 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 S is not a sampler type, then $v_n : S$; 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 $t: \Sigma T$ which evaluates as $(\mathsf{t}, 2N) \to (x_1, \ldots, x_{2N})$, where for notational clarity we have omitted the weights. From the above operational semantics, the lagged sampler $\mathsf{thin}(2, \mathsf{t} \otimes \mathsf{tl}(\mathsf{t})) : \Sigma (\mathsf{T} \times \mathsf{T})$ evaluates to

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

This is the 'self-product' which was denoted 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 $(t \otimes t, N) \to ((x_1, x_1), \dots (x_N, x_N))$. More generally, for any $K \in \mathbb{N}$, we define the K-fold self-product of a sampler as

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

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 K [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 CG of CG-spaces and continuous functions is Cartesian closed.

- 2) The category 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 CG. The product is in general different from the product in **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: $[\![N]\!] = \mathbb{N}$, equipped with the discrete topology, and $[\![R]\!] = \mathbb{R}, [\![R^+]\!] = [0,\infty)$ with the usual

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

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

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

$$\frac{(t, N) \to ((v_1, w_1), \dots, (v_N, w_N)) \quad (t, N) \to ((v_1, w_1), \dots, (v_N, w_N))}{(\mathsf{tl}(t), N \to v_1} \quad (\mathsf{tl}(t), N \to ((v_1, w_1), \dots, (v_N, w_N)))$$

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

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

TABLE II: Big-step operational semantics of sampler operations

topology. The spaces $f^{-1}(i), f \in \{\le, <, \ge, >, =, \ne\}, i \in 2$ are interpreted precisely as the notation suggests, e.g.

$$[\![<^{-1}(0)]\!] = \{(x,y) \mid x,y \in \mathbb{R} \land x \ge y\},\$$
$$[\![=^{-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 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 $1 \stackrel{!}{\longleftarrow} F1 \stackrel{F!}{\longleftarrow} FF1 \stackrel{FF!}{\longleftarrow} \dots$ is the terminal coalgebra of F.

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

$$\llbracket \Sigma \, \mathsf{T} \rrbracket \triangleq \nu(\llbracket \mathsf{T} \rrbracket \times \mathbb{R}^+ \times \mathrm{Id}). \tag{4}$$

Since products are limits, and limits commute with limits, it is clear that the functor $[T] \times \mathbb{R}^+ \times \mathrm{Id}$ 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 terminal object 1 is trivially metrizable, and since \mathbb{R}^+ is metrizable, each object in the terminal sequence will be metrizable provided [T] is, and thus $\prod_n ([T] \times \mathbb{R}^+)^n$ will be metrizable whenever [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 \mathbb{CG} defining $[\![\Sigma\,T]\!]$ is the same as in \mathbb{T} op when $[\![T]\!]$ is

metrizable (for example, if T is a ground type or a product of ground types). However, by defining $[\![\Sigma T]\!]$ coinductively rather than simply as $([\![T]\!] \times \mathbb{R}^+)^\omega$, we obtain a terminal coalgebra structure on $[\![\Sigma T]\!]$, 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 $\llbracket f^{-1}(0) \rrbracket$ and $\llbracket f^{-1}(1) \rrbracket$ 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. $\llbracket <^{-1}(0) \rrbracket$ is closed and $\llbracket <^{-1}(1) \rrbracket$ 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 $f:f^{-1}(0)+f^{-1}(1)\to \mathbf{B}$.

The topology on $\llbracket f^{-1}(0)+f^{-1}(1) \rrbracket$ is finer than the usual topology on $\mathbb{R} \times \mathbb{R}$, which means that the identity map $\mathrm{Id}: \llbracket f^{-1}(0)+f^{-1}(1) \rrbracket \to \mathbb{R} \times \mathbb{R}$ 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 $S \triangleleft T$, then $\llbracket S \rrbracket$ and $\llbracket T \rrbracket$ share the same carrier and the corresponding identity map $\mathrm{Id}: \llbracket S \rrbracket \to \llbracket T \rrbracket$ is continuous.

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

$$\frac{x: \mathtt{R} \vdash x: \mathtt{R} \quad \vdash 0: \mathtt{R}}{x: \mathtt{R} \vdash (x,0): \mathtt{R} \times \mathtt{R}} \quad \mathsf{Neq} + \mathtt{Eq} \lhd \mathtt{R} \times \mathtt{R} \\ \frac{x: (x,0)^{-1} \mathtt{Neq} + (x,0)^{-1} \mathtt{Eq} \vdash (x,0): \mathtt{Neq} + \mathtt{Eq}}{x: (x,0)^{-1} \mathtt{Neq} + (x,0)^{-1} \mathtt{Eq} \vdash x = 0: \mathtt{B} \quad \vdash 1: \mathtt{R}} \quad \vdash -1: \mathtt{R} \\ \frac{x: (x,0)^{-1} \mathtt{Neq} + (x,0)^{-1} \mathtt{Eq} \vdash x = 0: \mathtt{B} \quad \vdash 1: \mathtt{R}}{x: (x,0)^{-1} \mathtt{Neq} + (x,0)^{-1} \mathtt{Eq} \vdash \mathsf{if} \ x = 0 \ \mathsf{then} \ 1 \ \mathsf{else} \ -1: \mathtt{R}}$$

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

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

and thus [p] is the continuous map

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

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 $+: R \times R \to R$ or $\exp: R \to R$, 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' $t^{-1}(\mathsf{T}_i)$ as the pullback (inverse image) of $[\![t]\!]$ along the inclusion $[\![T]\!]_i \hookrightarrow \coprod_{i \in m} [\![T_i]\!]$ which is, as the notation implies, simply given by $[\![t]\!]^{-1}([\![T_i]\!])$. Since $\coprod_{i \in m} [\![T_i]\!]$ and $[\![T]\!]$ share the same carrier, it is clear that this defines a partition of $[\![\Gamma]\!]$, and we can thus retype t as a continuous map $\coprod_{i \in m} [\![t^{-1}(\mathsf{T}_i)]\!] \to \coprod_{i \in m} [\![\mathsf{T}_i]\!]$, 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:

$$\begin{array}{c} x: \mathtt{R}, y: \mathtt{R} \vdash (x,y): \mathtt{R} \times \mathtt{R} \\ \hline (x,y): (x,y)^{-1} (<^{-1}(0)) + (x,y)^{-1} (<^{-1}(1)) \vdash (x,y): <^{-1}(0) + <^{-1}(1) \\ \hline (x,y): (x,y)^{-1} (<^{-1}(0)) + (x,y)^{-1} (<^{-1}(1)) \vdash x < y: \mathtt{B} \end{array}$$

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

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

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 $\mathbb{C}G$.

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 T, $\llbracket \Sigma T \rrbracket \triangleq \nu(\llbracket T \rrbracket \times \mathbb{R}^+ \times \mathrm{Id})$. In particular, $\llbracket \Sigma T \rrbracket$ comes equipped with a coalgebra structure map

$$\mathrm{unfold}_T: \llbracket \Sigma\, T \rrbracket \to \llbracket T \rrbracket \times \mathbb{R}^+ \times \llbracket \Sigma\, T \rrbracket \,.$$

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

$$\operatorname{beh}(\gamma):X\to \llbracket \operatorname{\Sigma} \operatorname{T}
rbracket$$
 .

Since $[\![\Sigma T]\!]$ is interpreted in \mathbb{CG} , it follows automatically that both unfold_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. [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 \mathbf{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 $[\![\Sigma T]\!]$, 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 T and an integer N we inductively define its associated value type $\mathrm{val}^N(\mathtt{T}) \in \mathrm{Value}$ by:

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

TABLE III: Denotational semantics of sampler operations

where $G \in Ground$.³ We now define the *generalized projection* maps $p_T^N : [T] \to [val^N(T)]$ recursively via

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

The reader will have noticed that we have defined $p_{\mathtt{S} \to \mathtt{T}}^N$ trivially. The reason is that, as a quick examination of the rules of Table II will reveal, there is no conclusion and no premise 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. [Appendix A] For any program $\vdash t : T$, we have

$$(t,N) \to v \Leftrightarrow p_{\mathtt{T}}^N(\llbracket t \rrbracket) = \llbracket v \rrbracket.$$

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 $\Gamma \vdash s : T$ and $\Gamma \vdash t : T$ are equivalent, notation $\Gamma \vdash s \approx t : T$, if they are related by the smallest congruence relation on well-typed terms containing the rules of Table IV.⁴

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

Theorem IV.1. [Appendix A] The rules of Table IV are sound: if $\Gamma \vdash s \approx t : T$, then $\llbracket \Gamma \vdash s : T \rrbracket = \llbracket \Gamma \vdash t : T \rrbracket$.

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

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: $\{tl, hd, wt, thin, \otimes\}$ and $\{map, reweight\}$. 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 $tl, hd, wt, thin, \otimes$ are pulled all the way inwards.

Proposition IV.2. Let $\Gamma \vdash t : \Sigma T$ 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

 $^{^3}$ 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.

 $^{^4}$ By congruence relation, we mean that pprox is an equivalence relation preserved by all operations in the language. For example, if $\Gamma \vdash s \approx t : \Sigma T$ holds, then $\Gamma \vdash \mathtt{tl}(s) \approx \mathtt{tl}(t) : \Sigma T$ must hold as well, and the same for all operations in the language.

```
\Gamma, s : S \vdash (\lambda x : S.t)(s) \approx t[x \leftarrow s] : T
                                                                                                                                                                            \Gamma \vdash \lambda x : \mathtt{S}.t(x) \approx t : \mathtt{S} \to \mathtt{T}
                                                                                                        \Gamma, s : S \vdash let x = s in t \approx (\lambda x : S. t)(s) : T
                                                                    \Gamma \vdash if True then s else t \approx s : T \qquad \Gamma \vdash if False then s else t \approx t : T
                                                                                   \Gamma \vdash \mathtt{fst}((s,t)) \approx s : \mathtt{S}
                                                                                                                                                                                    \Gamma \vdash \operatorname{snd}((s,t)) \approx t : T
       \Gamma \vdash \mathtt{hd}(\mathtt{map}(s,t)) \approx s(\mathtt{hd}(t)) : \mathtt{T}
                                                                                                             \Gamma \vdash \mathsf{wt}(\mathsf{map}(s,t)) \approx \mathsf{wt}(t) : \mathsf{R}^+
                                                                                                                                                                                                                            \Gamma \vdash \mathtt{tl}(\mathtt{map}(s,t)) \approx \mathtt{map}(s,\mathtt{tl}(t)) : \Sigma \mathsf{T}
\Gamma \vdash (\mathtt{hd}(s),\mathtt{hd}(t)) \approx \mathtt{hd}(s \otimes t) : \mathtt{S} \times \mathtt{T}
                                                                                                         \Gamma \vdash \mathsf{wt}(s) * \mathsf{wt}(t) \approx \mathsf{wt}(s \otimes t) : \mathsf{R}^+
                                                                                                                                                                                                                        \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
                                                                                                            \Gamma \vdash \mathtt{wt}(\mathtt{thin}(n,t)) \approx \mathtt{wt}(t) : \mathtt{R}^+
                                                                                                                                                                                                           \{\Gamma \vdash \mathsf{tl}(\mathsf{thin}(n,t)) \approx \mathsf{thin}(n,\mathsf{tl}^n(t)) : \Sigma \mathsf{T} \mid n \in \mathbb{N}\}\
                                                                                                                      \Gamma \vdash \mathtt{thin}(\mathtt{1},t) \approx t : \Sigma \mathtt{T}
             \Gamma \vdash \mathtt{hd}(\mathtt{prng}(s,t)) \approx t : \mathtt{T}
                                                                                                                \Gamma \vdash \mathsf{wt}(\mathsf{prng}(s,t)) \approx 1 : \mathsf{R}^+
                                                                                                                                                                                                                          \Gamma \vdash \mathsf{tl}(\mathsf{prng}(s,t)) \approx \mathsf{prng}(s,s(t)) : \Sigma \mathsf{T}
   \Gamma \vdash \mathtt{hd}(\mathtt{reweight}(s,t)) \approx \mathtt{hd}(t) : \mathtt{T}
                                                                                           \Gamma \vdash \mathsf{wt}(\mathsf{reweight}(s,t)) \approx s(\mathsf{hd}(t)) * \mathsf{wt}(t) : \mathsf{R}^+
                                                                                                                                                                                                              \Gamma \vdash \mathsf{tl}(\mathsf{reweight}(s,t)) \approx \mathsf{reweight}(s,\mathsf{tl}(t)) : \Sigma \mathsf{T}
                      \Gamma \vdash \mathtt{thin}(n,\mathtt{thin}(m,t)) \approx \mathtt{thin}(n*m,t) \approx \Sigma \mathtt{T}
                                                                                                                                                                                         \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 \mathsf{T} \mid n \in \mathbb{N}\}
                                                                                                                                                                                \Gamma \vdash \text{thin}(n, \text{map}(s, t)) \approx \text{map}(s, \text{thin}(n, t)) : \Sigma T
                   \Gamma \vdash s \otimes \text{map}(f, t) \approx \text{map}(\text{id}_{S} \times f, s \otimes t) : \Sigma (S \times T)
                                                                                                                                                                              \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})
                                                                                                                                                                 \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}(q, b) \approx \mathtt{prng}(f \times q, (a, b)) : \Sigma(S \times T)
                                                                                                                                                                      \Gamma \vdash \text{thin}(n, s) \otimes \text{thin}(n, t) \approx \text{thin}(n, s \otimes t) : \Sigma(S \times T)
```

TABLE IV: Rules for sampler equivalence

the form $\Gamma \vdash \operatorname{map}(f, \operatorname{reweight}(g, \operatorname{map}(f', \ldots, s))) : \Sigma T$ or $\Gamma \vdash \operatorname{reweight}(g, \operatorname{map}(f, \operatorname{reweight}(g', \ldots, s))) : \Sigma T$, 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 simpler form whose correctness can then be verified.

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

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 $s: \Sigma T$ targets a probability distribution on [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 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 $1 = \{\bot\}$ is the terminal object and + the coproduct in Top.

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

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

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. $\varepsilon_X(\sigma) = \bot$, and consider the map to the terminal object $!: X \to 1$. Then $\varepsilon_1(!\circ\sigma) = \varepsilon_1(\bot,\bot,\ldots) = \delta_\bot$, but $\mathcal{P}_\bot(!)(\varepsilon_X(\sigma)) = \mathcal{P}_\bot(!)(\bot) = \bot$.

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

Proposition V.1. [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 $s riangle pring(\lambda x : \mathbb{R} \cdot x/2, 1)$ and the term p riangle if x = 0 then 1 riangle less - 1 of Ex. III.1. Assume first that \mathbb{R} is equipped with its standard topology, i.e. that $[\![p]\!]$ 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 $\varepsilon([\![s]\!]) = \delta_0$: letting $f: \mathbb{R} \to \mathbb{R}$ be bounded Lipschitz, we have

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

$$\leq \lim_{n \to \infty} \left| \frac{1}{n} \sum_{i=1}^n \frac{1}{2^i} \right| \leq \lim_{n \to \infty} \frac{2}{n} = 0$$

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

Let us now equip \mathbb{R} with the topology given by type-checking p as described in Ex. III.1. This makes [p] bounded and continuous, and we therefore no longer have $\varepsilon([s]) = \delta_0$; indeed $\lim_n \int [p] d\widehat{[s]}_n = -1 \neq [p](0) = 1$. In fact we now have $\varepsilon([s]) = \bot$, i.e. s is no longer a sampler targeting anything for this topology, which prevents the failure of Prop. V.1 on [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 $p \triangleq \text{if } x = 2 \text{ then } 1 \text{ else } -1 \text{ instead.}$ 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 $\varepsilon(\llbracket \mathbf{s} \rrbracket) = \delta_0$, since we can write $\int f d\llbracket \hat{\mathbf{s}} \rrbracket_n = \int_{\{2\}^c} f d\llbracket \hat{\mathbf{s}} \rrbracket_n + \int_{\{2\}} f d\llbracket \hat{\mathbf{s}} \rrbracket_n = \int_{\{2\}^c} f d\llbracket \hat{\mathbf{s}} \rrbracket_n$, and similarly for δ_0 . Because the discontinuity of $\llbracket \mathbf{p} \rrbracket$ is not assigned any mass by δ_0 , the topology on \mathbb{R} making $\llbracket \mathbf{p} \rrbracket$ continuous no longer prevents $\varepsilon(\llbracket \mathbf{s} \rrbracket)$ from converging, and we can therefore safely push \mathbf{s} forward through \mathbf{p} using map.

B. Calculus for asymptotic targeting

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

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

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

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

We introduce in Table V a relation \leadsto which is sound with respect to asymptotic targeting; this is the relation which was used in the proofs of § II. That is, if $\Gamma \vdash s : \Sigma S \leadsto \mu$, then s is a parametrised sampler on S which asymptotically targets a parametrised distribution μ on [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 $t: T \to T$ and an initial value x: T via $\mathtt{prng}(t,x): \Sigma T.^5$ However, Table V also incorporates a set of 'axioms' for built-in samplers \mathtt{rand}_i , each targeting distributions $\mu_i \in \mathcal{P}[\![T_i]\!]$; 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 $\operatorname{prng}(\lambda x: R: 1-x,0)$. This sampler, which generates the unweighted samples $(0,1,0,1,\ldots)$, targets the uniform Bernoulli distribution; however, applying $\operatorname{thin}(2,-)$ 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.

⁵Applying this rule requires showing that the initial point of the sampler is *typical*; a point $x \in [\![T]\!]$ is called *typical* if it belongs to the μ -mass 1 subset $X \subseteq [\![T]\!]$ in which the ergodic theorem holds [19, Theorem 9.6].

Fig. 5: Validity of Marsaglia sampler in Listing 4

Theorem V.1. [Appendix A] Targeting \leadsto is sound: if $\Gamma \vdash s$: $\Sigma S \leadsto \mu$, then $\varepsilon_{\llbracket S \rrbracket} \circ \llbracket s \rrbracket = \mu$.

It is easily seen that the natural corresponding notion of completeness, $\varepsilon_{\llbracket \mathtt{S} \rrbracket} \circ \llbracket \mathtt{s} \rrbracket = \mu \to \Gamma \vdash s : \Sigma \mathtt{S} \leadsto \mu$, does not hold; see the following example.

Example V.6. Let $s riangleq prng(\lambda n : R.n + 1, 0)$; clearly, s does not target any (probability) measure. Apply to s the transformation $map(\lambda n : R . 2^{(-1*n)}, s)$. While s does not target anything, and so our targeting calculus cannot prove that the transformed sampler targets anything, it is immediate that this sampler does target the Dirac measure δ_0 .

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.

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

Example V.7. Suppose we want $s: \Sigma \mathbb{R} \leadsto \operatorname{Bern}(1/2)$. A sampler alternating between the sampler $z \triangleq \operatorname{prng}(\lambda x: \mathbb{R} \cdot x/2,1)$ of Ex. V.3 and its shifted version $\operatorname{map}(\lambda x: \mathbb{R} \cdot x/2,1)$ of Ex. V.3 and its shifted version $\operatorname{map}(\lambda x: \mathbb{R} \cdot x/2,1)$ will satisfy the condition, but will never visit 0 or 1! We can use the previous result to enforce that a sampler s targeting $\operatorname{Bern}(1/2)$ should visit 0 i.o. by constructing s in such a way that it has type $\Sigma((x,0)^{-1}\operatorname{Neq}+(x,0)^{-1}\operatorname{Eq})$ (see Ex. III.1). We can, in the same manner, enforce that a sampler s' targeting $\operatorname{Bern}(1/2)$ 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 $\operatorname{Bern}(1/2)$ and guaranteed to visit 0,1 i.o.

Example V.8. We conclude with an example highlighting sampler compositionality by chaining two well-known sampling algorithms. Consider the following program:

```
let box = \lambda u : R^+ \times R^+ . sqrt(-2*log(fst(u))) * cos(2*pi*snd(u)) in
let joint = map(id_R \times box, rand^3) in
let d = \alpha - 1/3 in
let c = 1/(3*sqrt(d)) in
let c = 1/(3*sqrt
```

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 gammadistributed 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], [9], 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] BILLINGSLEY, P. Convergence of probability measures. John Wiley & Sons, 2013.
- [4] BINGHAM, E., CHEN, J. P., JANKOWIAK, M., OBERMEYER, F., PRAD-HAN, 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.
- [5] 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., ROBIN-SON, 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)
- [6] 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.
- [7] BROOKS, S., GELMAN, A., JONES, G., AND MENG, X.-L. Handbook of Markov Chain Monte Carlo. CRC press, 2011.
- [8] 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).
- [9] 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.
- [10] DAHLQVIST, F., KOZEN, D., AND SILVA, A. Semantics of Probabilistic Programming: A Gentle Introduction. Cambridge University Press, 2020, pp. 1–42.
- [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 Com*putation 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 higherorder 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 623dimensionally 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.

$$\frac{(t,N) \rightarrow v}{(v,N) \rightarrow v} \text{ v a value} \qquad \qquad \frac{(t,N) \rightarrow v}{(f(t),N) \rightarrow f(v)} \text{ Func } \ni f: \mathbf{T} \rightarrow \mathbf{G}$$

$$\frac{((\lambda x:T.t)(s),N) \rightarrow v}{(\mathbf{let} \ x=s \ \text{in} \ t,N) \rightarrow v} \qquad \frac{(t,N) \rightarrow v}{((\lambda x:T.t)(s),N) \rightarrow v}$$

$$\frac{(t,N) \rightarrow v}{(\mathbf{cast}(\top)t,N) \rightarrow v} \qquad \frac{(c,N) \rightarrow j \in I \quad (s_j[x_j \leftarrow t],N) \rightarrow v}{(\mathbf{case} \ (c,t) \ \text{of} \ \{(i,x_i) \Rightarrow s_i\}_{i \in I}),N) \rightarrow v} \qquad \frac{(t,N) \rightarrow v}{(\mathbf{in}_j \ (t),N) \rightarrow v}$$

$$\frac{(s,N) \rightarrow v_1 \quad (t,N) \rightarrow v_2}{((s,t),N) \rightarrow (v_1,v_2)} \qquad \frac{(t,N) \rightarrow (v_1,v_2)}{(\mathbf{fst}(t),N) \rightarrow v_1} \qquad \frac{(t,N) \rightarrow (v_1,v_2)}{(\mathbf{snd}(t),N) \rightarrow v_2}$$

$$\frac{((s(\mathbf{hd}(t)),\mathbf{wt}(t)),N) \rightarrow (v_1,w_1) \quad \dots \quad ((s(\mathbf{hd}(\mathbf{t1}^{N-1}(t)),\mathbf{wt}(\mathbf{t1}^{N-1}(t))),N) \rightarrow (v_N,w_N)}{(\mathbf{map}(s,t),N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N))}$$

$$\frac{((\mathbf{hd}(t),s(\mathbf{hd}(t)) \cdot \mathbf{wt}(t)),N) \rightarrow (v_1,w_1) \quad \dots \quad ((\mathbf{hd}(\mathbf{t1}^{N-1}(t)),s(\mathbf{hd}(\mathbf{t1}^{N-1}(t))) \cdot \mathbf{wt}(\mathbf{t1}^{N-1}(t))),N) \rightarrow (v_N,w_N)}{(\mathbf{reweight}(s,t),N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N))}$$

$$\frac{(s,N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N)) \quad (t,N) \rightarrow ((v_1',w_1'),\dots,(v_N',w_N'))}{(s \otimes t,N) \rightarrow (((v_1,v_1'),w_1 \cdot w_1'),\dots,((v_N,v_N'),w_N \cdot w_N'))}$$

$$\frac{(t,N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N)) \quad (t,N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N))}{(t,N) \rightarrow (v_1,w_1),\dots,(v_N,w_N))} \quad (\text{wt}(t),N) \rightarrow w_1}{(\text{thin}(s,t),N) \rightarrow ((v_1,w_1),(v_1,w_1,w_1,v_1,\dots,(v_N,w_N)))}$$

$$\frac{(s,N) \rightarrow i \quad (t,N) \rightarrow ((v_1,w_1),\dots,(v_N,w_N))}{(t,N) \rightarrow ((v_1,w_1),(v_1,w_1,w_1,w_1,\dots,(v_N,w_N))}} \quad (\text{wt}(t),N) \rightarrow w_1}{(\text{thin}(s,t),N) \rightarrow ((v_1,w_1),(v_1,w_1,w_1,w_1,v_1,\dots,(v_N,w_N))})}$$

TABLE VI: Full big-step operational semantics

APPENDIX

Type-checking of Rejection-sampling (§ II-C)

We first type-check accept from § II-C. This determines the type T which was left unspecified. To keep the derivation readable we define $t \triangleq (y, \text{phi}(x) * \text{sqrt}(2 * \text{pi}))$.

$$\frac{x: \mathtt{R}, y: \mathtt{R} \vdash x: \mathtt{R} \quad \vdash \mathtt{phi} : \mathtt{R} \to \mathtt{R} \quad \vdash 2 : \mathtt{R} \quad \vdash \mathtt{pi} : \mathtt{R}}{x: \mathtt{R}, y: \mathtt{R} \vdash y: \mathtt{R}} \\ \frac{x: \mathtt{R}, y: \mathtt{R} \vdash y: \mathtt{R} \quad x: \mathtt{R}, y: \mathtt{R} \vdash \mathtt{phi} (x) * 2 * \mathtt{pi} : \mathtt{R}}{x: \mathtt{R}, y: \mathtt{R} \vdash (y, \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi})) : \mathtt{R} \times \mathtt{R}} \\ \frac{x: \mathtt{R}, y: \mathtt{R} \vdash (y, \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi})) : \mathtt{R} \times \mathtt{R}}{(x, y): t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \vdash (y, \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi})) : <^{-1} (0) + <^{-1} (1)} \\ \frac{(x, y): t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \vdash y < \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi}) : \mathtt{B} \quad \vdash 0 : \mathtt{R}^+ \quad \vdash 1 : \mathtt{R}^+}{(x, y): t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \vdash \mathtt{if} \ y < \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi}) \ \mathtt{then} \ 1 \ \mathtt{else} \ 0 : t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \to \mathtt{R}^+} \\ \vdash \lambda(x, y): t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \ . \ \mathtt{if} \ y < \mathtt{phi} (x) * \mathtt{sqrt} (2 * \mathtt{pi}) \ \mathtt{then} \ 1 \ \mathtt{else} \ 0 : t^{-1} (<^{-1} (0)) + t^{-1} (<^{-1} (1)) \to \mathtt{R}^+}$$

Fig. 6: Derivation of accept from § II-C

We now define $T = t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1))$ and $proj \triangleq \lambda u : T$. $fst(cast(R \times R)u)$.

$$\frac{\frac{u: \texttt{T} \vdash u: \texttt{T}}{u: \texttt{T} \vdash \mathsf{cast} \langle \texttt{R} \times \texttt{R} \rangle u: \texttt{R} \times \texttt{R}}{\frac{u: \texttt{T} \vdash \mathsf{fst} (\mathsf{cast} \langle \texttt{R} \times \texttt{R} \rangle u: \texttt{R} \times \texttt{R}}{u: \texttt{T} \vdash \mathsf{fst} (\mathsf{cast} \langle \texttt{R} \times \texttt{R} \rangle u): \texttt{R} \times \texttt{R}}} \xrightarrow{\frac{\vdash \mathsf{tri} : \Sigma \, \texttt{R} \quad \vdash \mathsf{rand} : \Sigma \, \texttt{R}}{\vdash \mathsf{tri} \otimes \mathsf{rand} : \Sigma \, (\texttt{R} \times \texttt{R})}}{\vdash \mathsf{tri} \otimes \mathsf{rand} : \Sigma \, \texttt{T}}} \Sigma \, \texttt{T} \triangleleft \Sigma \, (\texttt{R} \times \texttt{R}), \Gamma = \emptyset$$

$$\frac{\vdash \mathsf{proj} : \texttt{T} \rightarrow \texttt{R}}{\vdash \mathsf{proj} : \texttt{T} \rightarrow \texttt{R}} \xrightarrow{\vdash \mathsf{rand} : \Sigma \, \texttt{R}} \quad \vdash \mathsf{tri} \otimes \mathsf{rand} : \Sigma \, \texttt{T}}{\vdash \mathsf{rand} : \Sigma \, \texttt{T}} \times \mathsf{T} \triangleleft \Sigma \, (\texttt{R} \times \texttt{R}), \Gamma = \emptyset$$

$$\frac{\vdash \mathsf{proj} : \texttt{T} \rightarrow \texttt{R}}{\vdash \mathsf{map}(\mathsf{reweight}(\mathsf{accept}, \mathsf{tri} \otimes \mathsf{rand})) : \Sigma \, \texttt{T}}$$

Fig. 7: Derivation of the rejection sampling algorithm § II-C

Proof of Prop. III.1.

In order to prove this result by induction on the derivation tree of $(t, N) \to v$, we must first generalise it to include higher samplers.

Proposition A.1. If $\vdash s : \Sigma^k S$ is a closed k-order sampler where S is not a sampler type and $k \in \{0, 1, 2, \ldots\}$, then for any $N \in \mathbb{N}$, if $(s, N) \to v$, then v has the form of a k-nested weighted list of values of type S. For example, for k = 0, v : S is simply a value of type S; for k = 1, $v = ((v_1, w_1), \ldots, (v_N, w_N))$ is a weighted list of values $v_n : S$ and $w_n \ge 0$; for k = 2, $v = (((v_1^1, w_1^1), \ldots, (v_N^1, w_N^1)), w_1), \ldots, (((v_1^N, w_1^N), \ldots, (v_N^N, w_N^N)), w_N))$ is a weighted list of weighted lists of values of type $v_n^n : S$, and so on.

Base case. As values v cannot have sampler type, the only possibility for a derivation $(v, N) \to v$ where $v : \Sigma^k T$ for some type T is k = 0, which makes our result immediate.

Inductive case. We illustrate the inductive argument for each case, depending on the last rule of the derivation of $(t, N) \to v$, where $\vdash t : \Sigma^k T$ is a k-order sampler for some $k \in \{0, 1, 2, \ldots\}$, and T is not a sampling type (i.e. contains no occurrences of Σ).

- 1) **Built-in functions.** There are no built-in functions which either input or output sampler types, so k=0. Taking the inductive hypothesis that each input s_i : G_i reduces to a value v_i : G_i , where G_n are ground types, we immediately obtain that $(f(s_1,\ldots,s_n),N) \to v$ evaluates to a value v of ground type G, giving our result.
- 2) Case. Assuming that $t = \mathsf{case}\ (c,t')$ of $\{(i,x_i) \Rightarrow s_i\}_{i \in n}$, we must have $\vdash s_i : \Sigma^k \mathsf{T}$. Taking the inductive hypothesis that $(s_i[x_i \leftarrow t'], N) \to v$ evaluates to a k-nested weighted list, if $(c,N) \to i \in n$, it immediately follows that $(t,N) \to v$ does as well.
- 3) Function application. Take $t = (\lambda x : \Sigma^{k'} S.t')(s)$ to be an instance of function application, where the function in question inputs a k-sampler and outputs a k-sampler, where S does not contain any sampler types itself; we must have $\vdash s : \Sigma^{k'} S$ in order for the expression to be well-typed. In order to have $(t, N) \to v$ evaluate to a value, our operational semantics requires $(t'[x \leftarrow s], N) \to v$; taking the inductive hypothesis that $t'[x \leftarrow s]$ evaluates to a k-nested list of values of type S, our desired result follows.
- 4) let-binding. Trivially follows from function application, as (let x = s in $t', N) \to v$ iff $((\lambda x : S.t')(s), N) \to v$.
- 5) **Product.** If t = (s, s'), the result is trivial as $\vdash t : \Sigma^k T$ implies k = 0 and so $T = S \times S'$ where $\vdash s : S, \vdash s' : S'$ are each not sampler types; therefore, (s, s') is clearly a value of type T (i.e., a 0-nested weighted list).
- 6) **Projections.** If t = fst((s, s')), then $\vdash s : \Sigma^k T$, and so the inductive hypothesis $(s, N) \to v$ immediately implies our result; the same argument applies to snd and t.
- 7) **Head.** If t = hd(s), then $\vdash s : \Sigma^{k+1}T$. Taking the inductive hypothesis that $(s, N) \to v$ implies that v is a (k+1)-nested weighted list of values of type T, we need only note that the first element of this list is itself a k-nested weighted list of values of type T.
- 8) Weight. If t = wt(s), our result is trivially true, as the output of wt(s) can only be a nonnegative real number $\vdash t : R^+$.
- 9) **Tail.** If $t = \mathtt{tl}(s)$, then by our inductive hypothesis, $(s, N+1) \to ((v_1, w_1), \dots, (v_{N+1}, w_{N+1}))$ where each v_n is a (k-1)-nested weighted list of elements of type T. It immediately follows that $(\mathtt{tl}(s), N)$ evaluates to a weighted list of N elements whose elements are each (k-1)-nested lists of type T.
- 10) **Thin.** If t = thin(i, s), then $\vdash n : N$ and $\vdash s : \Sigma^k T$, and by our inductive hypothesis, $(s, Ni) \to ((v_1, w_1), \dots, (v_{Ni+1}, w_{Ni+1}))$ where each v_n is a (k-1)-nested weighted list of elements of type T. It immediately follows that (thin(i, s), N) evaluates to a weighted list of N elements whose elements are each (k-1)-nested lists of type T.
- 11) Map. If t = map(s, t') and $(t, N) \to v$, the operational semantics of map requires that

$$\left((s(\mathtt{hd}(\mathtt{tl}^{n-1}(t'))), \mathtt{wt}(\mathtt{tl}^{n-1}(t'))), N\right) \to (v_n, w_n)$$

for each $n \in \{1, \ldots, N\}$. As t' is a subterm of t, our inductive hypothesis implies that if $\vdash t' : \Sigma^{k'}S$ for some $k' \in \{1, 2, \ldots\}$, then for any $N \in \mathbb{N}$, t' evaluates to a k'-nested weighted list of values of type S. Note that in order for t to be well-typed, we must have $\vdash s : \Sigma^{k'-1}S \to \Sigma^{k-1}T$. We have already shown that if this is the case, then $\mathtt{tl}^{n-1}(t')$ evaluates to a k'-nested weighted list of values of type S, and then that $\mathtt{hd}(\mathtt{tl}^{n-1}(t'))$ evaluates to a (k'-1)-nested weighted list of values of type S of length N, and then that $s(\mathtt{hd}(\mathtt{tl}^{n-1}(t')))$ evaluates to a (k-1)-nested weighted list of values of type T of length N. Our result follows by observing that if $((s(\mathtt{hd}(\mathtt{tl}^{n-1}(t'))), \mathtt{wt}(\mathtt{tl}^{n-1}(t'))), N) \to (v_n, w_n)$ for each $n \in \{1, \ldots, N\}$ where each v_n is a k-1-nested weighted list of values of type T, then the expression $((v_1, w_1), \ldots, (v_N, w_N))$ is a k-nested weighted list of values of type T, completing the proof.

- 12) Reweight. This proof works in exactly the same way as that of map.
- 13) **Product of samplers.** If $t = s \otimes s'$ and $\vdash t : \Sigma^k T$ where T contains no instances of Σ , then it must be that $k \geq 1$, that $\vdash s : \Sigma^k S$, and that $\vdash s' : \Sigma^k S'$. Our inductive hypothesis states that $(s, N) \to ((v_1, w_1), \dots, (v_N, w_N))$ where

each v_1 is a (k-1)-nested weighted list of values of type S, and $(s',N) \to ((v'_1,w'_1),\ldots,(v'_N,w'_N))$ where each v'_1 is a (k-1)-nested weighted list of values of type S'. Our result then follows by noting that the product $(((v_1,v'_1),w_1\cdot w'_1),\ldots,((v_N,v'_N),w'_N))$ is a k-nested weighted list of values of type S.

14) **Pseudorandom number generators.** Finally, assume t = prng(s,t'); in order for this expression to be well-typed, we must have $k \geq 1$, $\vdash t' : \Sigma^{k-1}T$, and $\vdash s : \Sigma^{k-1}T \to \Sigma^{k-1}T$. In order for (t,N) to evaluate to anything, we must have $(s^{n-1}(t),N) \to v_n$ for each $n \in \{2,\ldots,N\}$; as we have already proven the case for function abstraction, we know that each v_n is a (k-1)-nested weighted list of values of type T. We need only note then that $((v_1,1),\ldots,(v_N,1))$ is clearly a k-nested weighted list of values of type T.

Proof of Prop. III.3.

Let $f_n \to f$ be a convergent sequence a coalgebra maps in [X, FX]; we need to show that $beh_X(f_n) \to beh_X(f)$ in $[X, \nu F]$. The topology on $[X, \nu F]$ is the compact-open topology, which means that it is generated by the subbase of open sets of the shape

$$(K, V) \triangleq \{h : X \to \nu F \mid h[K] \subset U\}$$

for some fixed compact set $K\subseteq X$ and open set $U\subseteq \nu F$. Moreover, by construction of νF (see Thm. III.1), we know that the topology is induced by the product topology on $\prod_i F^i 1$. A base for this topology is given by intersections of cylinder sets with νF . Because we are also assuming that $\operatorname{int}(\nu F)\neq\emptyset$ in $\prod_i F^i 1$, it contains such an open set, and we can thus simply start with an open neighbourhood of $\operatorname{beh}_X f$ of the shape $(K,\prod_i V_i)$ where for all but *finitely* many indices $V_i=F^i 1$, and for the other indices V_i is an open subset of $F^i 1$ (and we don't have to worry about intersecting with νF). Given such an open set, we need to find $N\in\mathbb{N}$ such that for all n>N beh $_X(f_n)\in(K,\prod_i V_i)$.

By the construction of Thm. III.1 we have that

$$beh_X(f)(x) = (!_X(x), F!_X(f(x)), F^2!_X(Ff(f(x))), \dots)$$

where $!_X: X \to 1$ is the unique morphism to the terminal object. For each of the finitely many non-trivial open subsets $V_{i_k} \subset F^{i_k} 1, 1 \le k \le M$, because $f_n \to f$ and composition with continuous functions is a continuous operation on internal hom sets in \mathbf{CG} ([35, 5.9]), it follows that there exists N_k such that for every $n > N_k$

$$F^{i_k}!_X \circ F^{i_k-1}f_n \circ \ldots \circ f_n \in (K, V_{i_k})$$

By taking $N = \max_{1 \le k \le M} N_k$, we get that for for all $i \in \mathbb{N}$ and all n > N

$$F^i|_X \circ F^{i-1}f_n \circ \ldots \circ f_n \in (K, V_i)$$

In other words, for any n > N, beh_X $(f_n) \in (K, \prod_i V_i)$, which concludes the proof.

Proof of Thm. III.2.

 \Leftarrow) By induction on the derivation tree of $(t, N) \to v$.

Base case. The base case is trivial: the only derivation of length 0 allowed by Table II assumes that t=v is a value. It is easy to check that if t: T is a value, then $p_{\mathtt{T}}^N = \mathrm{id}_{\llbracket \mathtt{T} \rrbracket}$ and thus $p_{\mathtt{T}}^N(\llbracket t \rrbracket) = \llbracket t \rrbracket = \llbracket v \rrbracket$ tautologically.

Inductive case. Assume that the last rule of the derivation of $(t, N) \rightarrow v$ is

(i) **Built-in functions.** $t = f(s_1, \ldots, s_n)$ for some $s_i : G_i, 1 \le i \le n$. Since for a ground type G we have $p_G^N = \mathrm{id}_{\llbracket G \rrbracket}$, we immediately get

$$p_{\mathsf{G}}^{N}(\llbracket f(s_{1},\ldots,s_{n}) \rrbracket) \triangleq \llbracket f \rrbracket (\llbracket s_{1} \rrbracket,\ldots \llbracket s_{n} \rrbracket)$$

$$= \llbracket f \rrbracket (\llbracket v_{1} \rrbracket,\ldots,\llbracket v_{n} \rrbracket) \qquad \text{induction hypothesis}$$

(ii) Case. $t = \operatorname{case}\ (c,t')$ of $\{(i,x_i) \Rightarrow s_i\}_{i \in I}$. If c chooses the branch $j \in n$, then

$$\begin{split} p_{\mathtt{T}}^{N}\left(\left[\!\left[\mathsf{case}\;(c,t')\;\mathsf{of}\;\left\{(i,x_i)\Rightarrow s_i\right\}_{i\in I}\right]\!\right]\right) &\triangleq \left[\!\left[s_j\right]\!\right]\left(\left[\!\left[t'\right]\!\right]\right) \\ &= \left[\!\left[v\right]\!\right] & \text{induction hypothesis} \end{split}$$

(iii) λ -abstraction. $t = (\lambda x : S. t')(s) : T$ for some s : S and some t' : T.

(iv) let-binding. t = let x = s in t' : T for some s : S and t' : T.

$$\begin{split} p_{\mathtt{T}}^{N}\left(\llbracket \mathtt{let}\ x = s\ \mathtt{in}\ t \rrbracket\right) &\triangleq \llbracket t' \rrbracket \left(\llbracket s \rrbracket\right) \\ &= p_{\mathtt{T}}^{N}\left(\llbracket \lambda x.\ t' \rrbracket \left(\llbracket s \rrbracket\right)\right) \\ &= \llbracket v \rrbracket \end{split}$$

induction hypothesis

(v) **Product.** t = (s, s') for some s : S, s' : S'

$$p_{\mathtt{S} \times \mathtt{S}'}^{N}(\llbracket(s, s')\rrbracket) \triangleq p_{\mathtt{S} \times \mathtt{S}'}^{N}(\langle \llbracket s \rrbracket, \llbracket s' \rrbracket \rangle)$$

$$= \langle p_{\mathtt{S}}^{N}(\llbracket s \rrbracket, p_{\mathtt{S}'}^{N}(\llbracket s' \rrbracket) \rangle$$

$$= \langle \llbracket v_{1} \rrbracket, \llbracket v_{2} \rrbracket \rangle$$

$$= \llbracket (v_{1}, v_{2}) \rrbracket$$

inductive definition of p_{T}^{N} induction hypothesis

(vi) **Projections** t = fst(s, s') for some s : S, s' : S'

$$\begin{split} p_{\mathtt{S}}^{N}\left(\llbracket\mathtt{fst}(s,s')\rrbracket\right) &\triangleq p_{\mathtt{S}}^{N}\left(\pi_{1}\langle\llbracket s\rrbracket,\llbracket s'\rrbracket\rangle\right) \\ &= p_{\mathtt{S}}^{N}(\llbracket s\rrbracket) \\ &= \llbracket v_{1}\rrbracket \end{split}$$

induction hypothesis

and similarly for snd.

(vii) **Pushforward.** t = map(s,t) for some $s: S \to T$ and $t: \Sigma S$. To keep the derivation readable we will write s instead of $[\![s]\!]$, t instead of $[\![t]\!]$ and we introduce the following notation. Let F denote the functor $[\![T]\!] \times \mathbb{R}^+ \times \text{Id}$, let $\gamma: \nu F \to F \nu F$ denote the terminal coalgebra structure map unfold_T, let $\delta \triangleq [\![s]\!] \times \text{id}_{\mathbb{R}^+} \times \text{id}_{\Sigma S} \circ \text{unfold}_S$, the coalgebra structure map defining the map operation, let $b = \text{beh}(\delta)$, and let

$$h \triangleq \pi_1 \circ \mathrm{unfold_S}$$
 i.e. $h(t)$ is the first sample of t $w \triangleq \pi_2 \circ \mathrm{unfold_S}$ i.e. $w(t)$ is the weight of the first sample of t $f \triangleq \pi_3 \circ \mathrm{unfold_S}$ i.e. $f(t)$ is the tail of t

With this we can now derive

$$\begin{split} & p_{\text{TT}}^{N}([\![\mathsf{map}(s,t)]\!]) \\ & \triangleq \pi_{1:N} \circ \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{\omega} \left([\![\mathsf{map}(s,t)]\!]\right) \\ & \triangleq \pi_{1:N} \circ \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{\omega} \left(b(t)\right) \\ & \stackrel{(1)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ \pi_{1:N}(b(t)) \\ & \stackrel{(2)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ F \pi_{1:N-1} \circ \gamma \circ \left(b(t)\right) \\ & \stackrel{(3)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ F^{N-1} \pi_{1} \circ F^{N-2} \gamma \circ \ldots \circ F^{0} \gamma(b(t)) \\ & \stackrel{(4)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ F^{N-1} \pi_{1} \circ F^{N-1} b \circ F^{N-2} \delta \circ \ldots \circ F^{0} \delta(t) \\ & \stackrel{(5)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ F^{N-1} \pi_{1} \circ F^{N-1} b \left(\left(s(h(t)), w(t)\right), \ldots, \left(s(h(f^{N-1}(t))), w(f^{N-1}(t))\right), f^{N-1}(t)\right) \\ & \stackrel{(6)}{=} \left(p_{\text{T}\times \text{R}^{+}}^{N}\right)^{N} \circ \left(\left(s(h(t))\right), w(t)\right), \ldots, \left(s(h(f^{N-1}(t))), w(f^{N-1}(t))\right) \\ & = \left(p_{\text{T}\times \text{R}^{+}}^{N}(s(h(t))), w(t)\right), \ldots, p_{\text{T}\times \text{R}^{+}}^{N}(s(h(f^{N-1}(t))), w(f^{N-1}(t)))\right) \\ & = \left(\left(p_{\text{T}}^{N}(s(h(t))), w(t)\right), \ldots, \left(p_{\text{T}}^{N}(s(h(f^{N-1}(t)))), w(f^{N-1}(t))\right)\right) \\ & \stackrel{(7)}{=} \left[\left((v_{1}, w_{1}), \ldots, (v_{N}, w_{N})\right)\right] \end{split}$$

where (1) is the simple observation that $\pi_{1:N} \circ (p_{\mathsf{T}}^N)^\omega = (p_{\mathsf{T}}^N)^N \circ \pi_{1:N}$, (2) is by definition of γ , (3) is by iteration of (2), (4) follows from the fact that b is a coalgebra morphism, (5) is by definition of δ , (6) is by definition of F, b and p_{T}^1 , and (7) is by the induction hypothesis on the N premises of the rule.

(viii) Reweight. The proof is very similar to the case of map. Again, writing

$$\delta \triangleq \mathrm{id}_{\llbracket \mathtt{T} \rrbracket} \times (-\cdot -) \times \mathrm{id}_{\llbracket \mathtt{\Sigma} \mathtt{T} \rrbracket} \circ \langle \mathrm{id}_{\llbracket \mathtt{T} \rrbracket}, \llbracket s \rrbracket \rangle \times \mathrm{id}_{\mathbb{R}^+} \times \mathrm{id}_{\llbracket \mathtt{\Sigma} \mathtt{T} \rrbracket} \circ \gamma$$

for the coalgebra structure defining reweight and $b = beh(\delta)$, we get

$$\begin{split} & p_{\mathtt{TT}}^{N}(\mathtt{reweight}(s,t)) \\ & \triangleq \pi_{1:N} \circ \left(p_{\mathtt{T} \times \mathtt{R}^{+}}^{N} \right)^{\omega} (\mathtt{reweight}(s,t)) \\ & \triangleq \left(p_{\mathtt{T} \times \mathtt{R}^{+}}^{N} \right)^{N} \circ \pi_{1:N} \circ b(t) \\ & \stackrel{(1)}{=} \left(p_{\mathtt{T} \times \mathtt{R}^{+}}^{N} \right)^{N} \circ F^{N-1} \pi_{1} \circ F^{N-1} b \left((h(t), s(h(t)) w(t)), \dots, (h(f^{N-1}(t)), s(f^{N-1}(t)) w(f^{N-1}(t))), f^{N-1}(t) \right) \\ & \stackrel{(2)}{=} \left((p_{\mathtt{T}}^{N}(h(t)), s(h(t)) w(t)), \dots, (p_{\mathtt{T}}^{N}(h(f^{N-1}(t))), s(f^{N-1}(t)) w(f^{N-1}(t))) \right) \\ & \stackrel{(3)}{=} \left(([v_{1}], [w_{1}]), \dots, ([v_{N}], [w_{N}]) \right) \end{split}$$

where (1) follows the same derivation as in the case of map but with the definition of δ as above, (2) is by definition of F and $p_{\mathsf{T}\times\mathsf{R}^+}^N$, and (3) is by the the induction hypothesis applied to the N premises of the reweight rule.

- (ix) Product of samplers. The proof works in exactly the same way as for map and reweight.
- (x) Thin. The proof works in exactly the same way as for map and reweight.
- (xi) **Pseudorandom number generators.** Consider the term $prng(s, t) : \Sigma T$. Using

$$\delta \triangleq \langle \mathrm{id}_{\llbracket \mathtt{T} \rrbracket}, 1, \llbracket s \rrbracket \rangle$$

and $b = beh(\delta)$, the same steps as in the case of map and reweight yield

$$\begin{split} p_{\mathtt{TT}}^{N}([\![\mathtt{prng}(s,t)]\!]) &\triangleq \pi_{1:N} \circ (p_{\mathtt{T}\times\mathtt{R}^{+}}^{N})^{\omega}([\![\mathtt{prng}(s,t]\!]) \\ &= \left(p_{\mathtt{T}\times\mathtt{R}^{+}}^{N}\right)^{N} \circ F^{N-1}\pi_{1} \circ F^{N-1}b((t,1),(s(t),1),\ldots,s^{N-1}(t),1),s^{N}(t)) \\ &= \left((p_{\mathtt{T}}^{N}(t),1),(p_{\mathtt{T}}^{N}(s(t)),1),\ldots,(p_{\mathtt{T}}^{N}(s^{N-1}(t)),1)\right) \\ &= [\![(v_{1},1),(v_{2},1),\ldots,(v_{N},1))]\!] \end{split}$$

(xii) **Head.** Consider the term hd(t) for some $t : \Sigma T$. Using the same notation as above

$$\begin{split} p_{\mathtt{T}}^N \circ \llbracket \mathtt{hd}(t) \rrbracket &\triangleq p_{\mathtt{T}}^N \circ \pi_1 \circ \gamma(\llbracket t \rrbracket) \\ &= \pi_1 \circ \pi_1 \circ \left(p_{\mathtt{T} \times \mathtt{R}^+}^N \right)^N \circ \pi_{1:N}(\llbracket t \rrbracket) \\ &= \pi_1 \circ \pi_1 \, \llbracket (v_1, w_1), \dots, (v_N, w_N) \rrbracket & \text{induction hypothesis} \\ &= \llbracket v_1 \rrbracket \end{split}$$

(xiii) **Weight.** Consider the term wt(t) for some $t : \Sigma T$. The proof is the same as the above:

$$\begin{split} p_{\mathtt{T}}^N \circ \llbracket \mathtt{wt}(t) \rrbracket &\triangleq p_{\mathtt{T}}^N \circ \pi_2 \circ \gamma(\llbracket t \rrbracket) \\ &= \pi_2 \circ \pi_1 \circ \left(p_{\mathtt{T} \times \mathtt{R}^+}^N \right)^N \circ \pi_{1:N}(\llbracket t \rrbracket) \\ &= \pi_2 \circ \pi_1 \, \llbracket (v_1, w_1), \dots, (v_N, w_N) \rrbracket & \text{induction hypothesis} \\ &= \llbracket w_1 \rrbracket \end{split}$$

(xiv) **Tail.** Consider the term tl(t) f for some $t : \Sigma T$. It is immediate that

$$\begin{split} p_{\mathtt{ET}}^N \circ & \texttt{[tl}(t) \texttt{]} \triangleq \left(p_{\mathtt{T} \times \mathtt{R}^+}^N \right)^N \circ \pi_{1:N}(\pi_3 \circ \gamma(\llbracket t \rrbracket)) \\ &= \left(p_{\mathtt{T} \times \mathtt{R}^+}^N \right)^N \circ \pi_{2:N+1}(\llbracket t \rrbracket) \\ &= \pi_{2:N+1} \circ \left(p_{\mathtt{T} \times \mathtt{R}^+}^N \right)^{N+1} \circ \pi_{1:N+1}(\llbracket t \rrbracket) \\ &= \pi_{2:N+1} \left[(v_1, w_1), \dots, (v_{N+1}, w_{N+1}) \right] & \text{inductive hypothesis} \\ &= \left[(v_2, w_2), \dots, (v_{N+1}, w_{N+1}) \right] \end{split}$$

 \Rightarrow) By induction on the typing-proof of t. Note that for any term t: T, $p_{\mathtt{T}}^{N}$ [t] is necessarily a value, by definition of $p_{\mathtt{T}}^{N}$.

Base case. The only programs which are type-checkable in 0 steps are the constants. Since all constants are values and values operationally evaluate to themselves, the base case holds trivially.

Inductive case. The proof is routine and we only show a few cases. Suppose that the last step of the rule applied in the type-checking of t was

(i) **Product.** Suppose $\vdash (s,t) : S \times T$ and that $p_{S \times T}^N(\llbracket (s,t) \rrbracket) = \llbracket v \rrbracket$ for some value v. Since the last applied rule had premises $\vdash s : S$ and $\vdash t : T$ we have

$$\begin{split} \llbracket v \rrbracket &= p_{\mathsf{S} \times \mathsf{T}}^N(\llbracket s \otimes t \rrbracket) \\ &= p_{\mathsf{S}}^N \times p_{\mathsf{T}}^N \langle \llbracket s \rrbracket, \llbracket t \rrbracket \rangle & \text{inductive definition of } p_{\mathsf{S} \times \mathsf{T}}^N \\ &= (p_{\mathsf{S}}^N \ \llbracket s \rrbracket, p_{\mathsf{T}}^N \ \llbracket t \rrbracket) \\ &= (\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) \end{split}$$

for some values v_1, v_2 . By the induction hypothesis it is therefore the case that $(s, N) \to v_1$ and $(t, N) \to v_2$ for any $N \in \mathbb{N}$ and it follows that $(s \otimes t, N) \to (v_1, v_2)$ by definition of the reduction relation \to .

- (ii) λ -abstraction. If $\vdash \lambda x : S \to T$, then the term $\lambda x : S$ is a value, and thus $(\lambda x : S : t, N) \to \lambda x : S$ it trivially.
- (iii) **Head.** Suppose that $\vdash \operatorname{hd}(t)$: T, and that $p_{\operatorname{T}}^N(\llbracket\operatorname{hd}(t)\rrbracket) = \llbracket v_1 \rrbracket$ for some value v_1 . Since the last applied rule has the premise $\vdash t$: $\Sigma\operatorname{T}$, and given the semantics of hd, it must be the case that for any $N \geq 1$, $\left(p_{\operatorname{T}\times\operatorname{R}}^N\right)^N \circ \pi_{1:N}(t) = \llbracket ((v_1,w_1),\ldots,(v_N,w_N)) \rrbracket$ for some values v_i,w_i . By the induction hypothesis it must be the case that $(t,N) \to ((v_1,w_1),\ldots,(v_N,w_N))$, and thus that $(\operatorname{hd}(t),N) \to v_1$.
- (iv) Weight. The proof is the same as that for hd. Suppose that $\vdash \mathsf{wt}(t) : \mathsf{T}$, and that $p_{\mathsf{T}}^N(\llbracket \mathsf{wt}(t) \rrbracket) = \llbracket w_1 \rrbracket$ for some weight $w_1 \geq 0$. Since the last applied rule has the premise $\vdash t : \mathsf{\Sigma} \mathsf{T}$, and given the semantics of wt, it must be the case that for any $N \geq 1$, $\left(p_{\mathsf{T} \times \mathsf{R}^+}^N\right)^N \circ \pi_{1:N}(t) = \llbracket ((v_1, w_1), \ldots, (v_N, w_N)) \rrbracket$ for some values v_i, w_i . By the induction hypothesis it must be the case that $(t, N) \to ((v_1, w_1), \ldots, (v_N, w_N))$, and thus that $(\mathsf{wt}(t), N) \to w_1$.
- (v) **Pushforward.** Suppose that $\vdash \max(t,s) : \Sigma T$ and that $p_{\Sigma T}^N([\max(t,s)]) = [((v_1,w_1),\ldots,(v_n,w_n)]]$. The premises of the last applied rule must have been $\vdash s : \Sigma S$ and $\vdash t : S \to T$, and it follows from the semantics of map that $[v_i] = p_T^N[t(\operatorname{hd}(\operatorname{tl}^{i-1}(s)))]$ and $[w_i] = [\operatorname{wt}(\operatorname{tl}^{i-1}(s))]$. It follows from the induction hypothesis that $(t(\operatorname{hd}(\operatorname{tl}^{i-1}(s)),N) \to v_i$ and $(wt(\operatorname{tl}^{i-1}(s)),N) \to w_i$, and thus by the definition of \to we have that $(\operatorname{map}(t,s),N) \to ((v_1,w_1),\ldots,(v_n,w_n))$.

Proof of Thm. IV.1.

Standard rules:

1) β - and η -equivalence.

$$\begin{split} & \llbracket \Gamma \vdash (\lambda x : \mathtt{S}.t)(s) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash t[x \leftarrow s] : T \rrbracket \,, \\ & \llbracket \Gamma \vdash \lambda x : \mathtt{S}.t(x) : \mathtt{S} \to \mathtt{T} \rrbracket = \llbracket \Gamma \vdash t : \mathtt{S} \to \mathtt{T} \rrbracket \end{split}$$

The soundness of β - and η -equivalence is well-known and immediate from the properties of exponential objects.

2) let-reduction.

$$\llbracket \Gamma, s : S \vdash \mathtt{let} \ x = s \ \mathtt{in} \ t : \mathtt{T} \rrbracket = \llbracket \Gamma, s : S \vdash (\lambda x : S.t)(s) : \mathtt{T} \rrbracket$$

True by definition of the denotational semantics of let.

3) Projections.

$$\begin{split} & \llbracket \Gamma \vdash \mathtt{fst}((s,t)) : \mathtt{S} \rrbracket = \llbracket \Gamma \vdash s : \mathtt{S} \rrbracket, \\ & \llbracket \Gamma \vdash \mathtt{snd}((s,t)) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash t : \mathtt{T} \rrbracket \end{split}$$

Immediate from the properties of Cartesian products.

Congruence rules: Trivial in the denotational setting: if $\llbracket \Gamma \vdash s : S \rrbracket = \llbracket \Gamma \vdash s' : S \rrbracket$ have identical semantics, then clearly, for any built-in operation op : $S \to T$, $\llbracket \Gamma \vdash op(s) : T \rrbracket = \llbracket \Gamma \vdash op(s') : T \rrbracket$; the same extends to n-ary operations. **Coinductive definitions**:

1) **Map.**

$$\begin{split} & \llbracket \Gamma \vdash \mathtt{hd}(\mathtt{map}(s,t)) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash s(\mathtt{hd}(t)) : \mathtt{T} \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathtt{wt}(\mathtt{map}(s,t)) : \mathtt{R}^+ \rrbracket = \llbracket \Gamma \vdash s(\mathtt{wt}(t)) : \mathtt{R}^+ \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathtt{tl}(\mathtt{map}(s,t)) : \Sigma \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(\mathtt{tl}(t)) : \Sigma \, \mathtt{T} \rrbracket \end{split}$$

Immediate from the coinductive definition of map.

2) Product.

$$\begin{split} & \llbracket \Gamma \vdash (\mathsf{hd}(s), \mathsf{hd}(t)) : \mathsf{S} \times \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathsf{hd}(s \otimes t) : \mathsf{S} \times \mathsf{T} \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathsf{wt}(s) * \mathsf{wt}(t) : \mathsf{R}^+ \rrbracket = \llbracket \Gamma \vdash \mathsf{wt}(s \otimes t) : \mathsf{R}^+ \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathsf{tl}(s) \otimes \mathsf{tl}(t) : \Sigma \left(\mathsf{S} \times \mathsf{T} \right) \rrbracket = \llbracket \Gamma \vdash \mathsf{tl}(s \otimes t) : \Sigma \left(\mathsf{S} \times \mathsf{T} \right) \rrbracket \end{split}$$

Immediate from the coinductive definition of \otimes .

3) Thinning.

$$\begin{split} \llbracket \Gamma \vdash \mathsf{hd}(\mathsf{thin}(n,t)) : \mathsf{T} \rrbracket &= \llbracket \Gamma \vdash \mathsf{hd}(t) : \mathsf{T} \rrbracket \,, \\ \llbracket \Gamma \vdash \mathsf{wt}(\mathsf{thin}(n,t)) : \mathsf{R}^+ \rrbracket &= \llbracket \Gamma \vdash \mathsf{wt}(t) : \mathsf{R}^+ \rrbracket \,, \\ \forall n \in \mathbb{N}, \llbracket \Gamma \vdash \mathsf{tl}(\mathsf{thin}(n,t)) : \Sigma \, \mathsf{T} \rrbracket &= \llbracket \Gamma \vdash \mathsf{thin}(n,\mathsf{tl}^n(t)) : \Sigma \, \mathsf{T} \rrbracket \,, \end{split}$$

Immediate from the coinductive definition of thin.

4) Pseudorandom number generators.

$$\begin{split} & \llbracket \Gamma \vdash \mathtt{hd}(\mathtt{prng}(s,t)) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash t : \mathtt{T} \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathtt{wt}(\mathtt{prng}(s,t)) : \mathtt{R}^+ \rrbracket = \llbracket \Gamma \vdash 1 : \mathtt{R}^+ \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathtt{tl}(\mathtt{prng}(s,t)) : \mathtt{\Sigma}\,\mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{prng}(s,s(t)) : \mathtt{\Sigma}\,\mathtt{T} \rrbracket \end{split}$$

Immediate from the coinductive definition of prng.

5) Reweighting.

$$\begin{split} & \llbracket \Gamma \vdash \mathsf{hd}(\mathsf{reweight}(s,t)) : \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathsf{hd}(t) : \mathsf{T} \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathsf{wt}(\mathsf{reweight}(s,t)) : \mathsf{R}^+ \rrbracket = \llbracket \Gamma \vdash s(\mathsf{hd}(t)) * \mathsf{wt}(t) : \mathsf{R}^+ \rrbracket \,, \\ & \llbracket \Gamma \vdash \mathsf{tl}(\mathsf{reweight}(s,t)) : \Sigma \, \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathsf{reweight}(s,\mathsf{tl}(t)) : \Sigma \, \mathsf{T} \rrbracket \end{split}$$

Immediate from the coinductive definition of reweight.

Composition rules:

1) Thinning over thinning.

$$\llbracket \Gamma \vdash \mathsf{thin}(n, \mathsf{thin}(m, t)) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathsf{thin}(n * m, t) : \Sigma \mathsf{T} \rrbracket$$

For any possible value of the context $\gamma \in \llbracket \Gamma \rrbracket$, we show equality between the elements $\llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{thin}(m,t)) : \Sigma \, \mathtt{T} \rrbracket \, (\gamma)$ and $\llbracket \Gamma \vdash \mathtt{thin}(n*m,t) : \Sigma \, \mathtt{T} \rrbracket \, (\gamma)$ of $\llbracket \Sigma \, T \rrbracket$ coinductively. As all of the arguments we will make have precisely the same structure, we will only give that structure in full detail for this proof; for the rest, we will only present the bisimulation which gives our result.

We show this result by constructing, for each $\gamma \in \llbracket \Gamma \rrbracket$, a bisimulation $R(\gamma) \subseteq \llbracket \Sigma T \rrbracket \times \llbracket \Sigma T \rrbracket$. This is a set of samplers satisfying three properties:

- a) $\forall (s,t) \in R(\gamma), \pi_1(\mathrm{unfold}_{\mathtt{T}}(s)) = \pi_1(\mathrm{unfold}_{\mathtt{T}}(t));$ that is, the head of s and the head of t are the same
- b) $\forall (s,t) \in R(\gamma), \pi_2(\text{unfold}_T(s)) = \pi_2(\text{unfold}_T(t));$ that is, the first weight of s and the first weight of t are the same
- c) $\forall (s,t) \in R(\gamma), (\pi_3(\mathrm{unfold}_T(s)), \pi_3(\mathrm{unfold}_T(t))) \in R(\gamma);$ that is, applying t1 to two samplers in the bisimulation yields two more samplers in the bisimulation.

The structure of this bisimulation $R(\gamma)$ is typically found by applying t1 to both sides of the equivalence we wish to show, and then applying the rules we have previously shown (typically, the coinductive definitions of each operation, in this case thin) to simplify what results. For example, in this case, we can simplify

$$[\![\Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,\mathtt{thin}(m,t))) : \Sigma \, \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^{m*n}(t))) : \Sigma \, \mathtt{T}]\!]$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n*m,t)) : \Sigma \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n*m,\mathtt{tl}^{n*m}(t)) : \Sigma \, \mathtt{T} \rrbracket \, .$$

This suggests as a bisimulation the following set:

$$R(\gamma) = \left\{ \left[\!\!\left[(\Gamma \vdash \mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^k(t))) : \mathtt{\Sigma}\,\mathtt{T} \right]\!\!\right](\gamma), \left[\!\!\left[\Gamma \vdash \mathtt{thin}(n*m,\mathtt{tl}^k(t)) : \mathtt{\Sigma}\,\mathtt{T} \right]\!\!\right](\gamma) \mid k \in \mathbb{N} \right\}.$$

In future, as this expression is quite crowded, we will drop the dependence on γ .

We must now show that this is a valid bisimulation. First, we must show that applying hd and wt to each of these programs yields the same result, which is always immediate. In this case, applying the rule we had previously referred to as the coinductive definition of thin gives

$$\llbracket \Gamma \vdash \mathtt{hd}(\mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^k(t)))) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{hd}(\mathtt{thin}(m,\mathtt{tl}^k(t))) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{hd}(\mathtt{tl}^k(t)) : \mathtt{T} \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{hd}(\mathtt{thin}(n*m,\mathtt{tl}^k(t))) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{hd}(\mathtt{tl}^k(t)) : \mathtt{T} \rrbracket \; ;$$

The same argument exactly applies for wt:

$$\llbracket \Gamma \vdash \mathtt{wt}(\mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^k(t)))) : \mathtt{R}^+ \rrbracket = \llbracket \Gamma \vdash \mathtt{wt}(\mathtt{thin}(m,\mathtt{tl}^k(t))) : \mathtt{R}^+ \rrbracket = \llbracket \Gamma \vdash \mathtt{wt}(\mathtt{tl}^k(t)) : \mathtt{R}^+ \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{wt}(\mathtt{thin}(n*m,\mathtt{tl}^k(t))) : \mathtt{R}^+ \rrbracket = \llbracket \Gamma \vdash \mathtt{wt}(\mathtt{tl}^k(t)) : \mathtt{R}^+ \rrbracket \; ;$$

Finally, we must show that applying t1 to each of these expressions yields another element of the bisimulation. This is essentially the same argument as the one which led us to the bisimulation R:

$$\begin{split} & \left[\!\!\left[\Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^k(t)))) : \Sigma \, \mathtt{T}\right]\!\!\right] \\ &= \left[\!\!\left[\Gamma \vdash \mathtt{thin}(n,\mathtt{tl}^n(\mathtt{thin}(m,\mathtt{tl}^k(t)))) : \Sigma \, \mathtt{T}\right]\!\!\right] \\ &= \left[\!\!\left[\Gamma \vdash \mathtt{thin}(n,\mathtt{thin}(m,\mathtt{tl}^{n*m+k}(t))) : \Sigma \, \mathtt{T}\right]\!\!\right] \end{split}$$

and

$$\begin{split} & \big[\!\!\big[\Gamma \vdash \mathtt{tl}(\mathtt{thin}(n*m,\mathtt{tl}^k(t))) : \Sigma \, \mathtt{T}\big]\!\!\big] = \\ & \big[\!\!\big[\Gamma \vdash \mathtt{thin}(n*m,\mathtt{tl}^{n*m+k}(t)) : \Sigma \, \mathtt{T}\big]\!\!\big] \end{split}$$

Therefore, for any γ , applying t1 will yield another element of our bisimulation, and so our proof is complete. Using this proof as a reference, we will abbreviate the remainder of the bisimulation proofs in the Appendix, as the structure of each argument is identical.

2) Map over map.

$$\llbracket \Gamma \vdash \mathtt{map}(g,\mathtt{map}(f,t)) : \Sigma \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(g \circ f,t) : \Sigma \, \mathtt{T} \rrbracket$$

Applying the coinductive definition of map, we can easily see

$$\llbracket\Gamma \vdash \mathtt{tl}(\mathtt{map}(g,\mathtt{map}(f,t))) : \mathtt{\Sigma}\,\mathtt{T}\rrbracket = \llbracket\Gamma \vdash \mathtt{map}(g,\mathtt{map}(f,\mathtt{tl}(t))) : \mathtt{\Sigma}\,\mathtt{T}\rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{map}(g \circ f, t)) : \mathtt{\Sigma} \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(g \circ f, \mathtt{tl}(t)) : \mathtt{\Sigma} \, \mathtt{T} \rrbracket \, ;$$

which suggests the bisimulation

$$R = \{(\llbracket \Gamma \vdash \mathtt{map}(g,\mathtt{map}(f,\mathtt{tl}^n(t))) : \Sigma \, \mathtt{T} \rrbracket, \llbracket \Gamma \vdash \mathtt{map}(g \circ f,\mathtt{tl}^n(t)) : \Sigma \, \mathtt{T}) \rrbracket \mid n \in \mathbb{N} \},$$

This bisimulation is easily verified: simply apply hd to both sides and reduce by applying the coinductive definition of map and we will see that we obtain two equal expressions; apply wt to both sides and reduce by applying the coinductive definition of map, and two equal expressions will result; and finally apply tl to both sides and reduce by applying the coinductive definition of map, and we will see that the resulting pair is also included within this bisimulation.

3) Reweighting over reweighting.

$$\llbracket \Gamma \vdash \mathtt{reweight}(g,\mathtt{reweight}(f,t)) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(f \cdot g,t) : \Sigma \mathsf{T} \rrbracket$$

Applying t1 to both sides and using the previous rule relating t1 and reweight, we easily obtain

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{reweight}(g,\mathtt{reweight}(f,t))) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(g,\mathtt{reweight}(f,\mathtt{tl}(t))) : \Sigma \mathsf{T} \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathsf{tl}(\mathsf{reweight}(q \cdot f, t)) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathsf{reweight}(q \circ f, \mathsf{tl}(t)) : \Sigma \mathsf{T} \rrbracket$$
.

Equivalence then follows from the bisimulation

$$R = \{(\llbracket\Gamma \vdash \mathtt{reweight}(g,\mathtt{reweight}(f,\mathtt{tl}^m(t))) : \mathtt{\Sigma}\,\mathtt{T}\rrbracket, \llbracket\Gamma \vdash \mathtt{reweight}(g \circ f,\mathtt{tl}^m(t)) : \mathtt{\Sigma}\,\mathtt{T}]\rrbracket \mid m \in \mathbb{N}\}$$

which is easily verified, giving our desired equality.

4) Thinning over pseudorandom number generators.

$$\forall n \in \mathbb{N}, \llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{prng}(s,t)) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{prng}(s^n,t) : \Sigma \mathsf{T} \rrbracket$$

Applying t1 to both sides of each expression and simplifying using the coinductive definitions of thin and prng, we obtain

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,\mathtt{map}(s,t))) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{map}(s,\mathtt{tl}^n(t))) : \Sigma \mathsf{T} \rrbracket$$

and

$$[\![\Gamma \vdash \mathtt{tl}(\mathtt{map}(s,\mathtt{thin}(n,t))) : \Sigma \, \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{map}(s,\mathtt{thin}(n,\mathtt{tl}^n(t))) : \Sigma \, \mathtt{T}]\!] \, .$$

This suggests the choice of bisimulation

$$R = \{(\llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{map}(s,\mathtt{tl}^m(t))) : \Sigma \, \mathtt{T} \rrbracket, \llbracket \Gamma \vdash \mathtt{map}(s,\mathtt{thin}(n,\mathtt{tl}^m(t))) : \Sigma \, \mathtt{T} \rrbracket \} \mid m \in \mathbb{N} \}$$

which is easily verified and gives our result.

5) Thinning over map.

$$\llbracket \Gamma \vdash \mathtt{thin}(\mathtt{n},\mathtt{map}(\mathtt{s},\mathtt{t})) : \Sigma \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(\mathtt{s},\mathtt{thin}(\mathtt{n},\mathtt{t})) : \Sigma \, \mathtt{T} \rrbracket$$

Using the coinductive definitions of map and thin, we obtain

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,\mathtt{map}(s,t)) : \Sigma \, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{map}(s,\mathtt{tl}^n(t))) : \Sigma \, \mathtt{T} \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{map}(s,\mathtt{thin}(n,t))) : \Sigma \mathsf{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(s,\mathtt{thin}(n,\mathtt{tl}^n(t))) : \Sigma \mathsf{T} \rrbracket.$$

The desired result follows from

$$R = \{(\llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{map}(s,\mathtt{tl}^m(t))) : \Sigma \, \mathtt{T} \rrbracket, \llbracket \Gamma \vdash \mathtt{map}(s,\mathtt{thin}(n,\mathtt{tl}^m(t))) : \Sigma \, \mathtt{T} \rrbracket) \mid m \in \mathbb{N} \}$$

which is easily seen to be a valid bisimulation.

Product rules:

1) Thinning.

$$\llbracket \Gamma \vdash \mathtt{thin}(n,s) \otimes \mathtt{thin}(n,t) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n,s \otimes t) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket$$

Use the coinductive definition of thin to show

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,s) \otimes \mathtt{thin}(n,t)) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{tl}^n(s)) \otimes \mathtt{thin}(n,\mathtt{tl}^n(t)) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n, s \otimes t)) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n, \mathtt{tl}^n(s) \otimes \mathtt{tl}^n(t)) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket,$$

which suggests

$$R = \{ (\llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{tl}^m(s)) \otimes \mathtt{thin}(n,\mathtt{tl}^m(t)) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket, \\ \llbracket \Gamma \vdash \mathtt{thin}(n,\mathtt{tl}^m(s \otimes t)) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket) : m \in \mathbb{N} \}$$

as a bisimulation.

2) **Map.**

$$\llbracket \Gamma \vdash s \otimes \mathtt{map}(g,t') : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(id_{\mathtt{S}} \times g, s \otimes t') : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket$$

Applying t1 to each expression yields

$$\llbracket\Gamma\vdash \mathtt{tl}(s\otimes \mathtt{map}(g,t')): \Sigma(\mathtt{S}\times \mathtt{T})\rrbracket = \llbracket\Gamma\vdash \mathtt{tl}(s)\otimes \mathtt{map}(g,\mathtt{tl}(t')): \Sigma(\mathtt{S}\times \mathtt{T})\rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{map}(id_{\mathtt{S}} \times g, s \otimes t')) : \Sigma(\mathtt{S} \times \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(id_{\mathtt{S}} \times g, \mathtt{tl}(s) \otimes \mathtt{tl}(t')) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket,$$

suggesting

$$R = \left\{ (\llbracket \Gamma \vdash \mathtt{tl}^m(s) \otimes \mathtt{map}(g,\mathtt{tl}^m(s')) : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket, \\ \llbracket \Gamma \vdash \mathtt{map}(id_{S'} \times g,\mathtt{tl}^m(s),\mathtt{tl}^m(s))) : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket \right) \mid m \in \mathbb{N} \right\}$$

as a bisimulation.

$$\llbracket \Gamma \vdash \mathtt{map}(f,t) \otimes s' : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{map}(f \times id_T, t \otimes s') : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket$$

Same proof as previous.

3) Reweighting.

$$\llbracket \Gamma \vdash s \otimes \mathtt{reweight}(g,t') : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(1_S \cdot g, s \otimes t') : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket$$

Applying tl to both sides and simplifying using the coinductive definition of reweight gives

$$\llbracket \Gamma \vdash \mathsf{tl}(s \otimes \mathsf{reweight}(q, t')) : \Sigma(\mathsf{S} \times \mathsf{T}) \rrbracket = \llbracket \Gamma \vdash \mathsf{tl}(s) \otimes \mathsf{reweight}(q, \mathsf{tl}(t')) : \Sigma(\mathsf{S} \times \mathsf{T}) \rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{reweight}(1_\mathtt{S} \cdot g, s \otimes t')) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(1_\mathtt{S} \cdot g, \mathtt{tl}(s) \otimes \mathtt{tl}(t')) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket.$$

Choosing the bisimulation

$$R = \{(\llbracket \Gamma \vdash \mathtt{tl}^m(s) \otimes \mathtt{reweight}(g,\mathtt{tl}^m(t')) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket, \\ \llbracket \Gamma \vdash \mathtt{reweight}(1_{\mathtt{S}} \cdot g,\mathtt{tl}^m(s) \otimes \mathtt{tl}^m(t')) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket) \mid m \in \mathbb{N} \},$$

our result follows.

$$\llbracket \Gamma \vdash \mathtt{reweight}(f,t) \otimes s' : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(f \cdot 1_{\mathtt{T}}, s' \otimes t) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket$$

Same proof as previous.

4) Pseudorandom number generators.

$$\llbracket \Gamma \vdash \mathtt{prng}(f, a) \otimes \mathtt{prng}(g, b) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{prng}(f \times g, (a, b)) : \Sigma (\mathtt{S} \times \mathtt{T}) \rrbracket$$

Using the coinductive definition of prng, we quickly obtain

$$\llbracket\Gamma\vdash \mathtt{tl}(\mathtt{prng}(f,a)\otimes\mathtt{prng}(g,b)):\Sigma\left(\mathtt{S}\times\mathtt{T}\right)\rrbracket=\llbracket\Gamma\vdash\mathtt{prng}(f,f(a))\otimes\mathtt{prng}(g,g(b)):\Sigma\left(\mathtt{S}\times\mathtt{T}\right)\rrbracket$$

and

$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{prng}(f \times g, (a, b))) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket = \llbracket \Gamma \vdash \mathtt{prng}(f \times g, (f \times g)(a, b)) : \Sigma(\mathtt{S} \times \mathtt{T}) \rrbracket,$$

suggesting the bisimulation

$$\begin{split} R = \left\{ (\llbracket \Gamma \vdash \mathtt{prng}(f, f^m(a)) \otimes \mathtt{prng}(g, g^m(b)) : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket, \\ \llbracket \Gamma \vdash \mathtt{prng}(f \times g, (f \times g)^m(a, b)) : \Sigma \left(\mathtt{S} \times \mathtt{T} \right) \rrbracket) \mid m \in \mathbb{N} \right\} \end{split}$$

which gives our desired result.

Proof of Prop. IV.1.

Expanding Eq. (3), for any well-typed sampler $\Gamma \vdash s : \Sigma S$, the nested self-product $(s^m)^n$ is defined as

$$thin(n, thin(m, s \otimes tl(s) \otimes \cdots \otimes tl^{m-1}(s)) \otimes \cdots \otimes tl^{n-1}(thin(m, s \otimes tl(s) \otimes \cdots \otimes tl^{m-1}(s)))).$$

Applying the rule $\Gamma \vdash \mathtt{tl}(\mathtt{thin}(m,t)) \approx \mathtt{thin}(m,\mathtt{tl}^m(t))$: $\Sigma \mathtt{T}$ from Table IV on the innermost expressions, it follows that this program is equivalent in the context Γ to

$$thin(n, thin(m, tl^0(s) \otimes \cdots \otimes tl^{m-1}(s)) \otimes \cdots \otimes thin(m, tl^{mn-m}(s) \otimes \cdots \otimes tl^{mn-1}(s))).$$

Next, applying the rule $\Gamma \vdash \text{thin}(m, s \otimes t) \approx \text{thin}(m, s) \otimes \text{thin}(m, t) : \Sigma(S \times T)$, we see that the nested self-product is equivalent to

$$thin(n, thin(m, tl^0(s) \otimes tl^1(s) \otimes \cdots \otimes tl^{mn-1}(s))).$$

Applying the rule $\Gamma \vdash \text{thin}(n, \text{thin}(m, t)) \approx \text{thin}(m, t) : \Sigma T$ for composition of thin yields

$$thin(mn, tl^0(s) \otimes tl^1(s) \otimes \cdots \otimes tl^{mn-1}(s)),$$

and the above is precisely the definition of the self-product s^{mn} .

Proof of Prop. IV.3.

• Map: Applying the definition of the self-product Eq. (3), the syntax $map(f,s)^n$ is shorthand for the sampler

$$thin(n, map(f, s) \otimes tl(map(f, s)) \otimes \cdots \otimes tl^{n-1}(map(f, s)))$$

In a context Γ in which this sampler is well-typed, applying the rule $\Gamma \vdash \mathtt{tl}(\mathtt{map}(f,s)) \approx \mathtt{map}(f,\mathtt{tl}(s)) : \Sigma \mathsf{T}$ shows that the above sampler is equivalent to

$$thin(n, map(f, tl^0(s)) \otimes \cdots \otimes map(f, tl^{n-1}(s))).$$

For the purposes of this proof, abbreviate the n-fold Cartesian product of a program $f: S \to T$ as $f^{\times n}: S^n \to T^n$. Applying the rule $\Gamma \vdash \text{map}(f,s) \otimes \text{map}(g,t) \approx \text{map}(f \times g,(s,t)): \Sigma(S \times T)$, this sampler can also be written in the equivalent form

$$thin(n, map(f^{\times n}, tl^0(s) \otimes \cdots \otimes tl^{n-1}(s))).$$

Finally, applying the rule $\Gamma \vdash \text{thin}(n, \text{map}(f, s)) \approx \text{map}(f, \text{thin}(n, s)) : \Sigma S$ yields

$$\operatorname{map}(f^{\times n},\operatorname{thin}(n,\operatorname{tl}^0(s)\otimes\cdots\otimes\operatorname{tl}^{n-1}(s))))$$

which is, by the definition of the self-product, our desired result $map(f^{\times n}, s^n)$.

• **Reweight**: This proof proceeds the same as the above, but with map replaced with reweight and the Cartesian product × replaced with the pointwise product ·; nevertheless, we will go through it. Applying the definition of the self-product Eq. (3), the syntax reweight(f, s)ⁿ is shorthand for

$$thin(n, \mathtt{reweight}(f, s) \otimes \mathtt{tl}(\mathtt{reweight}(f, s)) \otimes \cdots \otimes \mathtt{tl}^{n-1}(\mathtt{reweight}(f, s)))$$

In a context Γ in which this sampler is well-typed, applying the rule $\Gamma \vdash \mathtt{tl}(\mathtt{reweight}(f,s)) \approx \mathtt{reweight}(f,\mathtt{tl}(s)) : \Sigma \mathtt{T}$ shows that the above sampler is equivalent to

$$thin(n, reweight(f, tl^0(s)) \otimes \cdots \otimes reweight(f, tl^{n-1}(s))).$$

For the purposes of this proof, abbreviate the n-fold pointwise product of a program $f: S \to R$ as $f^{\cdot n}: S^n \to R$. Applying the rule $\Gamma \vdash \mathtt{reweight}(f,s) \otimes \mathtt{reweight}(g,t) \approx \mathtt{reweight}(f \cdot g,(s,t)) : \Sigma(S \times T)$, this sampler can also be written in the equivalent form

$$thin(n, reweight(f^{\cdot n}, tl^0(s) \otimes \cdots \otimes tl^{n-1}(s))).$$

Finally, applying the rule $\Gamma \vdash \text{thin}(n, \text{reweight}(f, s)) \approx \text{reweight}(f, \text{thin}(n, s)) : \Sigma S$ yields

$$\mathtt{reweight}(f^{\cdot n},\mathtt{thin}(n,\mathtt{tl}^0(s)\otimes\cdots\otimes\mathtt{tl}^{n-1}(s))))$$

which is, by the definition of the self-product, our desired result reweight($f^{\cdot n}, s^n$).

Proof of Prop. V.1.

Let $g:Y\to\mathbb{R}$ be a bounded continuous function. Then $g\circ f:X\to\mathbb{R}$ is also bounded continuous, and it follows from the definition of weak convergence and of ε that

$$\lim_{n\to\infty}\int_X g\ d\widehat{f\circ\sigma_n} = \lim_{n\to\infty}\int_X g\circ f\ d\widehat{\sigma_n} \qquad \qquad \text{by definition}$$

$$= \int_X g\circ f\ d\mu \qquad \qquad \text{since } \varepsilon_X(\sigma) = \mu$$

$$= \int_X g\ df_*\mu \qquad \qquad \text{change of variable}$$

Thus
$$\widehat{f \circ \sigma}_n \longrightarrow f_* \mu$$
 weakly, i.e. $\varepsilon(f \circ \sigma) = f_*(\mu)$.

Proof of Thm. V.1.

By induction on the derivation.

- (i) **Built-in samplers.** These axioms are true by assumption.
- (ii) Equivalence. The fact that equivalent terms target the same measure is a simple consequence of the definition of targeting and of Thm. IV.1.

- (iii) **Tail.** The fact that the tail of a sampler σ targets the same measure as σ is a simple consequence of the definition of targeting in terms of a limit.
- (iv) **Pushforward.** If $[\![f]\!]$ is continuous for the standard topologies of the type system, then either: (a) a measure $\mu(\gamma)$ assigns some mass to the boundary of an element of the partition making $[\![f]\!]$ piecewise continuous, in which case $\varepsilon[\![s]\!]$ (γ) will not converge to $\mu(\gamma)$ and the premise of the rule does not hold, or (b) no measure $\mu(\gamma)$ assigns any mass to the boundary of an element of the partition making $[\![f]\!]$ continuous, in which case $\varepsilon[\![s]\!]$ (γ) = $\mu(\gamma)$, and the conclusion is again a consequence of Prop. V.1.
- (v) **Reweight.** This rule simply encodes the validity of importance sampling. Dropping the dependency in γ for clarity of notation, and letting $\nu = f \cdot \mu$ be the reweighted measure, the premise and side-condition of the rule together say that there exists $\alpha \in \mathbb{R}^+$ such that $f = \alpha \frac{d\mu}{d\nu}$, and that f is bounded on the support of μ . Since μ is a probability distribution, we have

$$\int f \ d\mu = \alpha \int \frac{d\mu}{d\nu} \ d\nu = \alpha \int \ d\nu = \alpha$$

Moreover, since s targets μ and f is bounded continuous, we get by writing $x_i \triangleq \pi_1(\pi_i(\llbracket s \rrbracket))$ and $w_i \triangleq \pi_2(\pi_i(\llbracket s \rrbracket))$ that

$$\alpha = \int f \ d\mu = \lim_{N \to \infty} \frac{1}{N} \frac{\sum_{i=1}^{N} f(x_i) w_i}{\sum_{k=1}^{N} w_k}$$
 (5)

Letting g be any bounded continuous function $[S] \to \mathbb{R}$ and noting that the pointwise product g.f is bounded continuous on the support of μ and ν , we have

$$\begin{split} \int g \; d\nu &= \frac{1}{\alpha} \int g.f \; d\mu \\ &= \frac{1}{\alpha} \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^N g(x_i) f(x_i) \frac{w_i}{\sum_{k=1}^N w_k} \\ &= \left(\lim_{N \to \infty} \frac{1}{N} \frac{\sum_{i=1}^N f(x_i) w_i}{\sum_{k=1}^N w_k} \right)^{-1} \left(\lim_{N \to \infty} \frac{1}{N} \frac{\sum_{i=1}^N g(x_i) f(x_i) w_i}{\sum_{k=1}^N w_k} \right) \\ &= \lim_{N \to \infty} \frac{1}{N} \frac{\sum_{i=1}^N g(x_i) f(x_i) w_i}{\sum_{i=1}^N f(x_i) w_i} \\ &\triangleq \lim_{N \to \infty} \int g \; d\mathbf{reweight}(f, s)_n \end{split}$$

In other words, reweight(f, s) targets ν .

(vi) **Pseudorandom number generators.** The prng rule is just a restatement of the well-known ergodic theorem [19, Theorem 9.6].

Proof of Prop. V.2.

The first part of the proof follows immediately if we can show that $S \triangleleft T$ implies that [S] and [T] are the same measurable space; this will be shown by induction on the sub-typing derivation. We start by showing that the functor Borel: $Top \rightarrow Meas$ commutes with coproducts. This will prove the base case, the coproduct rule, and the last two rules of Fig. 4b.

Let X,Y be two topological spaces (we will use the same name for topological (resp. measurable) spaces and their topologies (resp. σ -algebras)). We use the π - λ lemma to prove $\mathsf{Borel}(X+Y)=\mathsf{Borel}(X)+\mathsf{Borel}(Y)$. First note that $\mathsf{Borel}(X+Y)=\sigma(X+Y)$ by definition. Since X+Y is a topology, it is trivially also a π -system, and since $\mathsf{Borel}(X)+\mathsf{Borel}(Y)$ is a σ -algebra it is also trivially a λ -system. By definition, every open set U in X+Y has the property that $U=X\cap U$ is open in X, and is thus an element of $\mathsf{Borel}(X)$. Similarly $Y\cap U$ is open in Y and thus belongs to $\mathsf{Borel}(Y)$. It follows that $U=(U\cap X) \uplus (U\cap Y)$ belongs to $\mathsf{Borel}(X)+\mathsf{Borel}(Y)$ by definition of the coproduct in Meas. The inclusion $\mathsf{Borel}(X+Y)\subseteq \mathsf{Borel}(X)+\mathsf{Borel}(Y)$ now follows from the π - λ lemma.

Conversely, every measurable A in $\mathsf{Borel}(X) + \mathsf{Borel}(Y)$ is, by definition, of the shape $(A \cap X) \uplus (A \cap Y)$ with $(A \cap X) \in \mathsf{Borel}(X)$ and $(A \cap Y) \in \mathsf{Borel}(Y)$. Using the π - λ lemma it is easy to show that $\mathsf{Borel}(X) \subseteq \mathsf{Borel}(X+Y)$ and $\mathsf{Borel}(Y) \subseteq \mathsf{Borel}(X+Y)$, and it thus follows, since $\mathsf{Borel}(X+Y)$ is closed under unions, that $A = (A \cap X) \uplus (A \cap Y) \in \mathsf{Borel}(X+Y)$ which proves $\mathsf{Borel}(X+Y) \supseteq \mathsf{Borel}(X) + \mathsf{Borel}(Y)$.

To show that the functor Borel: $\mathbf{Top} \to \mathbf{Meas}$ commutes with products we need the extra assumption that the spaces are second-countable. A proof can then be found in e.g. [3, p244].

For the second part of the proof, let U be in the topology of [S] but not in the topology of [T]. This means that $\partial_T(U) = U \cap \operatorname{int}_T(U) \neq is$ open in [S] (since it's the intersection of two open sets in [S]). In particular it is a continuity set in [S] (since it is open, its interior is the empty set and it can therefore not have any μ -mass). By the Portmanteau lemma (which applies since the spaces are assumed to be metrizable) we must thus have

$$\lim_{n \to \infty} \widehat{\llbracket s \rrbracket}_n \left(\partial_{\mathtt{T}}(U) \right) = \mu(\partial_{\mathtt{T}}(U)) > 0$$

In particular, this is clearly impossible if $\llbracket s \rrbracket$ only visits $\partial_T(U)$ finitely many times.