Sampling languages:

Semantics and verification of sampling-based inference algorithms for probabilistic programming

William Smith

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

I, William Smith, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Practical probabilistic programs, especially those with approximate conditioning, by necessity combine a number of sampling techniques in sequence in order to produce samples - as well as certain complex language features in order to implement those techniques. Sufficiently complex probabilistic programs therefore require a compositional approach to verification, beginning with the fundamentals of language design, syntax, and semantics, and building to the higher-level questions about samplers and their statistical properties. In this thesis, we design a typed probabilistic programming language which is equipped with the features necessary to serve as a setting for the verification of sampling techniques. These features include: continuous random variables; higher-order functions; dependent types; compatibility with deterministic, that is to say pseudorandom, generation of samples; and samplers for (the marginal distributions of) stochastic processes. We give a semantics for the language in terms of concrete concepts, and show equivalence of the denotational and operational semantics. We present a set of rules for the effective verification of sampler equivalence in a suitable subset of the language. Stating, in this setting, the desired targeting relationship between samplers and the probability measures that they are to produce samples from, we then lay out a calculus centred on this targeting relationship, which enables the compositional verification of sampling techniques such as importance sampling and rejection sampling. These verification procedures are then extended so as to enable the verification of samplers which produce samples targeting (the marginal distributions of) stochastic processes. We conclude by arguing that a sampling

language with these features is a natural compilation target for a probabilistic programming language with approximate Bayesian conditioning, flexible enough to frame many Bayesian inference methods in common use without incorporating features that hinder effective verification.

Impact Statement

Outside academia. Programs which use random sampling have become ubiquitous. Any field of research which uses statistics, which is to say any field of research, makes use of them at some level. The more complex these uses are, the more important the task of program verification becomes. In particular, applied machine learning and machine learning research inevitably include large-scale and complex use of random sampling. This thesis deals with the problem of verifying that the outputs of programs which use random sampling have statistical properties which must hold in order for their use to be formally valid. In principle, subtle bugs and mistaken assumptions in the use of random samplers could yield invalid conclusions in research (whether in industry or academia) or improper behaviour of deployed programs – including bugs which cannot reasonably be detected with unit tests. The development of techniques to check statistical correctness, possibly in real time as programs are written, therefore, has the potential to affect research in any field which relies heavily on statistical modelling and machine learning, such as applications within – to name just a few – physics, advertising, climate modelling, and public health.

Inside academia. We have already discussed the applications of the verification of statistical properties for programs which use random sampling to research in any field which uses statistical modelling – whether that research takes place within or outside of academia. We will instead discuss contributions to the field of verification itself. In this thesis, we lay out an approach to verifying statistical properties for programs which use random sampling,

including pseudorandom number generation. We do this by providing the syntax and semantics of a probabilistic programming language designed in order to easily state the sampler operations we will consider; formally defining the properties we are to verify; and then building a system for verification within this language. We then extend this language, including its syntax, semantics, and its verification system, to incorporate the verification of programs which in some sense sample from a stochastic process (i.e. infinite families of samplers with certain consistency properties). Aside from the first-order goal of potentially preventing errors in research, our secondary aim includes popularising and spurring interest in verifying statistical properties of probabilistic programs within the academic community.

Acknowledgements

First and foremost I must thank my supervisors, Alexandra Silva and Fredrik Dahlqvist, for listening to me, giving excellent feedback, and more broadly taking a chance on me.

I'd also like to thank the entire PPLV group at UCL, past and present, for their conversation, ideas, and good cheer. I'm glad to have been able to share a Gower Street basement with you, whether just for an afternoon or for years.

Finally, I am deeply grateful to my parents, Robert and Gaetana Smith, for their support and patience, especially during the height of the pandemic.

◇◇◇

This thesis was partially supported by ERC grant Autoprobe (no. 101002697).

Contents

List of Figures

Chapter 1

Introduction

John von Neumann's famous remark, 'Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin' [1], is endlessly quoted in papers on pseudorandom number generation and probability theory. We won't break from this tradition; it's pithy, entertaining, and quoting von Neumann has the tendency to make one feel intelligent by association. We will differ from this established practice, though, by discussing the context.

Monte Carlo methods were first applied in 1947 by Stanisław Ulam and John von Neumann, as part of their research at Los Alamos, to estimate neutron multiplication rates [2]. That these first simulations were carried out on the ENIAC, the first true digital computer, illustrates that Monte Carlo methods were among the very first successful applications of digital computing; that these first simulations were applied to the task of designing nuclear fission weapons illustrates a much broader truth that is less relevant to computing.

Von Neumann wrote the above-quoted lines several years later, in 1951, in a short article which discusses some of the techniques that he and Ulam had successfully applied in their research. This article discussed the techniques which are today called inverse-transform sampling, rejection sampling (implemented in our language in listing 3.2), and the von Neumann extractor (implemented in listing 3.1), and also included a few remarks on arithmetic methods for generating random numbers. On that last subject, von Neumann

writes,

> Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number – there are only methods to produce random numbers, and a strict arithmetic procedure is not such a method. (It is true that a problem that we suspect of being solvable by random methods may be solvable by some rigorously defined sequence, but this is a deeper mathematical question than we can now go into.) We are here dealing with mere "cooking recipes" for making digits; probably they can not be justified, but should merely be judged by their results. Some statistical study of the digits generated by a given recipe should be made, but exhaustive tests are impractical. If the digits work well on one problem, they seem usually to be successful with others of the same type. [ibid]

Von Neumann's remark that many problems solvable by randomised methods are also often solvable by non-randomised methods is profound, and we will soon discuss many instances in which this is possible. For now it suffices to note that while formal methods for program semantics and verification were still several decades in the future, von Neumann was concerned from the beginning with the question of verifying these arithmetic procedures – though the primary method he had access to was black-box experimentation.

Since von Neumann's time, Monte Carlo methods and probabilistic programming have seen wide adoption in near every quantitative field, and have taken advantage of massive improvements in the available hardware. With the introduction of Markov chain Monte Carlo methods to the field of machine learning in the 1990s, this trend has only accelerated. The size and the complexity of these programs have grown significantly, and their societal importance continues to grow as well. As new techniques are invented and combined with existing techniques, formally articulating the preconditions necessary for

these composite programs to be sound quickly passes beyond the reach of existing methods. With the advent of true probabilistic programming languages in the 1990s and 2000s [3, 4, 5, 6, 7, 8, 9], including such features as conditioning, higher-order functions, continuous variables, and recursion, the task of verifying these programs has begun to necessitate the use of more complex formal methods. This process of verification will be the primary focus of this thesis.

A system for demonstrating that probabilistic programs are statistically sound statistically naturally begins with a semantics – a fully-specified mapping from a program to a mathematical function implemented by this program. These semantics, beginning with [10], necessarily became more complex as the features available to these probabilistic programs grew, reaching into fields such as order theory, topology, and category theory for useful models [11, 12, 13, 14]. We will draw on this literature as necessary for the remainder of this thesis.

Having chosen a semantics, our *verification* task is then to show that this mathematical object representing the program has certain required features. In our case, we aim to verify *samplers*: our desired verification task is to show that these samplers generate samples that, at least in a certain sense, behave as if they were taken from a desired probability distribution. The particular techniques von Neumann discusses in his monograph – in particular, rejection sampling and pseudorandom number generation – will serve as our motivating examples, and are discussed in more detail in the introduction to chapter 3.

Contributions. Our aim with this thesis is to pull together the progress which has been made in these disparate fields since von Neumann's remark, and summarise procedures by which computer programs which rely upon random sampling can be proven to be correct, even if that random sampling is in fact pseudorandom – that is to say, deterministic. This thesis focuses on the verification of *samplers* and *sampling languages*, a useful subset of probabilistic programming languages that encompasses many of their practical applications in the quantitative sciences.

We present a sampling language which is complex enough to include such procedures as rejection sampling, importance sampling, pseudorandom number generation, higher-order samplers, and stochastic processes, and yet simple enough that samplers within it can be effectively verified. This language is specified by an operational semantics (given in section 3.1.2) and denotational semantics (given in section 3.1.3), which are shown to be compatible in theorem 3.1.10. Our operational and denotational semantics are then extended to include dependent types in section 4.1.

Definition 3.3.7 states the natural notion of sampler correctness in this setting, a relationship that we refer to as 'targeting'. Intuitively, a sampler 'targets' a probability distribution if samples produced by this sampler behave, at least asymptotically, as if they were samples from the desired distribution. We introduce in fig. 3.5 a system for verifying samplers by applying equivalence rules to reduce them to simpler forms, and prove soundness in theorem 3.2.2. We then present a set of inference rules which preserve this targeting relationship, enabling the verification of samplers, and show their validity in theorem 3.3.10. This approach to verification is extended to include stochastic processes in section 4.3.

The majority of the material included is based on two papers which were joint work with Frederick Dahlqvist and Alexandra Silva; the first of these papers [15] forms the basis for chapter 3, while the second (unpublished) forms the basis for chapter 4.

Applications. This thesis is concerned with the semantics and verification of programs which incorporate random (or pseudorandom) sampling. That is, our focus is not on developing new techniques for sampling ourselves, but instead on aiding the formal verification of existing sampling schemes.

Problems involving random, pseudorandom, and approximate sampling are ubiquitous in statistics and applied science. To provide context for what is to come, we mention briefly here two broad (admittedly overlapping) categories of applications: Monte Carlo simulation and Bayesian inference.

In Monte Carlo simulation, our aim is to evaluate the expectation $\mathbb{E}[f]$ of a function f with respect to a certain probability distribution P, but we cannot do this closed-form. As a result, we write a program which, we claim, will produce samples x_n distributed according to our target distribution P, perhaps only approximately in some sense, and we then approximate this expectation as $\mathbb{E}[f] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_n)$ (an approximation which, we should be able to prove, converges almost surely to the correct expectation as $n \to \infty$). For example, consider a case in which the function f is a complex but deterministic program, such as a physics simulator, and x includes a set of unknown initial conditions distributed according to P. Then, in order to prove that Monte Carlo simulation does in fact converge, we must show that our chosen sampler, perhaps a pseudorandom number generator, does in fact generate samples from the desired distribution P.

In a typical Bayesian setting, a practitioner aims to generate approximate samples from a *posterior distribution*, a distribution over some set of parameters z which are to be estimated, conditioned on observed data x. Like the previous class of problems, we might aim to compute some expectation $\mathbb{E}[f]$ with respect to this posterior distribution. However, in a general Bayesian inference problem, generating useful approximate samples from the posterior distribution is not immediate. Rather than one single, universal technique for generating representative samples, there are a number of approximate sampling techniques (we will discuss several over the course of this thesis) that one might choose from and, especially, combine with one another, to obtain an approximate sampler. It is the verification of this compositional process of combining sampling techniques that we aim to simplify. Example applications to Bayesian-type problems are given in Example 3.3.17 and Example 4.3.8.

Outline. A short outline of the contents of each chapter follows.

Chapter 2 reviews some necessary concepts from probability theory, including the basics of measure-theoretic probability, laws of large numbers, convergence of empirical measures, and stochastic processes. These prelimi-

naries will be necessary in order to formally state the semantics of and define the task of verification for the programming languages that will be discussed in the following two chapters, and also as the motivation for the probabilistic programs that we will verify. The reader is advised to read these preliminaries alongside the main chapters as necessary, as they are referred back to in the main body of the text.

Chapter 3 will introduce a probabilistic programming language which is particularly well-suited to the verification of samplers. This probabilistic programming language will be deterministic in construction, will feature samplers as first-class objects, and crucially, will be able to cleanly express many of the sampling techniques discussed in chapter 2, making their verification straightforward. We will introduce (denotational and operational) semantics for this language, formally state the problem of sampler verification, and introduce a sound calculus for the verification of samplers. As stated previously, this chapter primarily presents the approach taken in [15], though with significant simplifications in presentation, corrections, and further development.

Chapter 4 extends the language introduced in the previous chapter to a more general setting, one which features dependent types. This extension, based on unpublished work, is necessary in order to incorporate stochastic processes, more specifically samplers that can generate samples from the marginal distributions of stochastic processes, within our framework. The semantics of the language is extended to incorporate this change, and the task of verifying a sampler is suitably generalised – though the results of the previous chapter will still apply in a restricted subset of this language.

Finally, chapter 5 reviews our approach and contrasts it with other approaches to similar problems discussed in the literature. We also discuss possible extensions of this approach to semantics.

Chapter 2

Preliminaries

This chapter reviews the fundamentals of measure-theoretic probability, the axiomatic development of which occurred largely concurrently with the emergence of probabilistic programming. The following outline of the fundamentals of measure-theoretic probability can be found in any textbook on the subject, such as [16].

We recall here definitions and properties that will be required in the technical developments of chapters 3 and 4. None of the results in this section are new, but the notations, definitions, and terminology used will be essential in the following chapters.

We begin with a modern review of measure theory as the axiomatic foundation for probability theory, the first compelling presentation of which was made by Kolmogorov in 1933 [17], building on the Lebesgue integral introduced in [18]. We continue by outlining a few relevant developments of the theory that have been made since, in particular introducing the essentials of ergodic theory and stochastic processes.

2.1 σ-algebras

Definition 2.1.1 (σ-algebra)**.** *A* ***σ-algebra****, or* ***σ-field****, Σ_X on a set X is a collection of subsets of X satisfying the following three properties:*

1. $\emptyset, X \in \Sigma_X$
2. *Closure under complement:* $A \in \Sigma_X \implies X \setminus A \in \Sigma_X$

3. *Closure under countable union:* $\forall n \in \mathbb{N}, A_n \in \Sigma_X \implies \bigcup_{n=1}^{\infty} A_n \in \Sigma_X$

Note that properties (1), (3) together imply closure under finite union (choosing all but finitely many sets A_n to be the empty set); properties (2), (3) together imply closure under countable intersection (noting $\bigcap_{n=1}^{\infty} A_n = X \setminus \bigcup_{n=1}^{\infty}(X \setminus A_n)$); and so properties (1), (2), (3) also imply closure under finite intersection.

In settings in which the container set X and σ-algebra Σ_X on X are understood, sets $A \in \Sigma_X$ are often called **measurable**, and sets $A \notin \Sigma_X$ **non-measurable**; the tuple (X, Σ_X) is often referred to as a **measurable space**.

Definition 2.1.2 (Measurable function)**.** *A function $f : X \to Y$, when X and Y are each equipped with σ-algebras Σ_X, Σ_Y, is called (Σ_X, Σ_Y)-**measurable** if the preimage of each measurable set in Y is measurable in X: that is, if $\forall B \in \Sigma_Y, f^{-1}(B) \in \Sigma_X$. When the choice of σ-algebra on each set is clear from context, we will simply say that the function f is measurable or non-measurable. If there exists an invertible mapping $f : X \to Y$ such that f and f^{-1} are both measurable, we say that the measurable spaces X and Y are isomorphic.*

Each nonempty set X has two trivial σ-algebras: the indiscrete algebra $\Sigma_X = \{\emptyset, X\}$, in which the only measurable sets are $\emptyset$ and X, and the discrete algebra $\Sigma_X = 2^X$, in which all subsets of X are measurable. If X is equipped with a topology, a natural choice of σ-algebra is the **Borel algebra**, defined as the smallest σ-algebra containing all of the open sets of X (which is also the smallest σ-algebra containing all of the closed sets of X). It follows that continuous functions on $\mathbb{R}$ are a subset of measurable functions; that they are a strict subset is easily seen by considering simple examples such as $1_{\mathbb{Q}}$, the indicator function on the rational numbers. Of special interest in the case of Euclidean space $\mathbb{R}^N$ are also the Lebesgue σ-algebras, which are refinements of the Borel algebra generated by the standard Euclidean topology on $\mathbb{R}^N$; these will be discussed shortly, when introducing Lebesgue measure.

A Polish (i.e. separable and completely metrisable) space X equipped with its Borel algebra is often referred to as a **standard Borel space**. This terminology is motivated by the following theorem due to Kuratowski: two standard Borel spaces X and Y are isomorphic if and only if the cardinalities of X and Y are the same [19]. As the largest Polish spaces have the cardinality of the continuum, it follows that every standard Borel space is isomorphic to either a finite set (equipped with its discrete σ-algebra), $\mathbb{Z}$ (likewise), or $\mathbb{R}$. We will not be concerned exclusively with standard Borel spaces here, but many of the spaces we consider will fall under this umbrella, and many of the standard theorems of the field are framed in this setting.

Measurable spaces support many (but not all; see Remark 2.1.7) of the natural constructions one would expect. It will be useful, for the reader less familiar with category theory, to explicitly define the product of measurable spaces, Definition 2.1.3, the coproduct of measurable spaces Definition 2.1.4, pullbacks Definition 2.1.5, and the initial sigma algebra, Definition 2.1.6.

Definition 2.1.3 (Product σ-algebra)**.** *The* ***product*** *of two measurable spaces X, Y is the Cartesian product $X \times Y$ equipped with the smallest σ-algebra such that the sets $A \times Y, X \times B$ are all measurable, where $A \subseteq X, B \subseteq Y$ range over measurable sets. Equivalently, the product can be defined as $X \times Y$ equipped with the smallest σ-algebra such that the canonical projections $\pi_X : X \times Y \to X, \pi_Y : X \times Y \to Y$ are measurable. It is natural to write the product of the σ-algebras Σ_X and Σ_Y as $\Sigma_X \otimes \Sigma_Y$, in order to avoid confusion with the Cartesian product of sets.*

More generally, for an indexed collection $(X_i)_{i \in I}$ of measurable spaces, the product σ-algebra can be defined either as the coarsest σ-algebra such that all of the projections $\pi_i : \prod_{j \in I} X_j \to X_i$ are measurable, or alternatively as the σ-algebra generated by the cylinder sets *$\prod_{i \in I} U_i$, in which only finitely many U_i are different from X.*

Definition 2.1.4 (Coproduct σ-algebra)**.** *Dual to the product, the* ***coproduct*** *of two measurable spaces X, Y is the disjoint union $X + Y$ equipped with the*

largest σ-algebra such that the natural injections $\iota_X : X \to X + Y, \iota_Y : Y \to X + Y$ are measurable maps. The corresponding generalisation to an arbitrarily large indexed collection is obvious.

Definition 2.1.5 (Pullback σ-algebra)**.** *Let $f : X \to Z, g : Y \to Z$ be measurable maps into the measurable space Z. The **pullback**, or **fibred product**, of these maps consists of the set $P = \{(x, y) : f(x) = g(y)\}$ of points at which these functions coincide, often written $P = X \times_Z Y$ in contexts in which the maps f and g are implicit, equipped with the weakest σ-algebra making the natural projections $p_X : P \to X, p_Y : P \to Y$ measurable.*

Definition 2.1.6 (Initial σ-algebra)**.** *Let (X, Σ_X) be a measurable space, and $f : Y \to X$ a function from the set Y to the set X. The induced **initial σ-algebra** Σ_Y on Y consists of the preimages of all the measurable sets in X under f; that this collection of preimages is a σ-algebra is easily seen.*

Remark 2.1.7. *Other familiar limits and colimits of measurable spaces, such as coproducts, can be easily defined, though we will not need them here. It is important to note, though, that exponentials of measurable spaces do not exist for many nontrivial spaces of interest. For example, there is no σ-algebra on the set of measurable functions $\mathbb{R}^{\mathbb{R}}$ such that the evaluation function $\mathrm{ev}_{\mathbb{R}} : \mathbb{R}^{\mathbb{R}} \times \mathbb{R} \to \mathbb{R}$ is measurable, where $\mathbb{R}$ is equipped with its Borel algebra [20, 21]; the same holds for the spaces $[0, 1]$ and 2^ω.*

2.2 Measures

Definition 2.2.1 (Measure)**.** *Given a measurable space (X, Σ), a **measure** is a map $\mu : \Sigma \to \bar{\mathbb{R}}$, taking values in the extended real numbers, satisfying*

1. *$\mu(\emptyset) = 0$*
2. *Nonnegativity: for all measurable A, $\mu(A) \geq 0$*
3. *Countable additivity: for a countably-infinite collection of pairwise-disjoint measurable sets A_n, i.e. such that $\forall m, n \in \mathbb{N}, A_m \cap A_n = \emptyset$ iff $m \neq n$, $\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n)$*

Note that as trivial consequences of (3), measures satisfy finite additivity and monotonicity $A \subseteq B \implies \mu(A) \leq \mu(B)$; for any measurable A, B, we have $\mu(A \cup B) = \mu(A \setminus B) + \mu(B \setminus A) + \mu(A \cap B)$; and for any measurable A, $\mu(X \setminus A) = \mu(X) - \mu(A)$.

A measurable space (X, Σ) equipped with a measure μ is often referred to as a **measure space**. Measurable sets A such that $\mu(A) = 0$ are often said to be μ**-null**, or simply **null** if the measure is clear from context.

A measure is called **finite** when $\mu(X)$ is finite, and σ**-finite** when there exists a countable cover $X = \bigcup_{n=1}^{\infty} A_n$ such that each A_n has finite μ-measure. A **probability measure** is a measure P such that $P(X) = 1$.

Simple examples of measures include the zero measure $\mu(A) = 0$; the infinite measure, for which $\mu(A) = \infty$ for all non-empty A; and the counting measure, which assigns each finite set to its cardinality, and each infinite set to ∞. Sums of measures are easily seen to be measures; scalar multiples of measures (by nonnegative real numbers) are as well.

Definition 2.2.2 (Empirical measure)**.** *Given a point $x \in X$, the **Dirac measure** on any σ-algebra on X assigns unit measure to a set A iff it contains the point x:*

$$\delta_x(A) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

*Given a finite sequence of points $x_1, \ldots, x_n \in X$, the resulting **empirical measure** is defined as the normalised sum of the resulting Dirac measures. Let P_n be the function taking such a sequence of points to the corresponding empirical measure; that is,*

$$P_n(x_1, \ldots, x_n)(A) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}(A).$$

The choice of notation P_n is motivated by the convergence of empirical measures under i.i.d. sampling $x_n \in P$; see theorem 2.6.5. Given a sequence of corresponding nonnegative weights $w_n \geq 0$, the corresponding definition for a

weighted empirical measure is obvious (provided at least one of the weights is nonzero).

Definition 2.2.3 (Product measure)**.** *Given two measure spaces* (X, Σ_X, μ_X) *and* (Y, Σ_Y, μ_Y)*, a* ***product measure*** $\mu_X \otimes \mu_Y$ *is any measure on the product* σ*-algebra such that* $(\mu_X \otimes \mu_Y)(A \times B) = \mu_X(A)\mu_Y(B)$ *for all measurable sets* $A \in \Sigma_X, B \in \Sigma_Y$*.*

In general, there is not always one unique product measure on the product σ-algebra, though this *does* hold when both μ_X and μ_Y are σ-finite [16, Section 35, Theorem B].

Definition 2.2.4 (Subspace)**.** *Given a measure space* (X, Σ_X, μ_X) *and a measurable subset* $A \subseteq X$*, define the corresponding subspace*[1] (A, Σ_A, μ_A) *as the measure space which equips* A *with the* σ*-algebra* $\Sigma_A = \{S \cap A : \Sigma_X\}$ *and the measure* $\mu_A(E) = \mu_X(E \cap A)$*.*

Another very important technique for constructing new measures is the pushforward:

Definition 2.2.5 (Pushforward)**.** *Given a measure space* (X, Σ_X, μ_X) *and a measurable function* $f : X \to Y$*, where* Y *is equipped with* σ*-algebra* Σ_Y*, define the* ***pushforward*** *of* μ_X *through* f*, written* $f_*\mu_X$*, to be the measure on* (Y, Σ_Y) *given by*

$$f_*\mu_X(B) = \mu_X(f^{-1}(B))$$

which assigns to each measurable set in Y *the measure of its preimage in* X*.*

This concept of the pushforward is commonly used in probability theory to specify a particular joint distribution, via the concept of *random variable*.

Definition 2.2.6 (Random variable)**.** *In probability theory, it is typical to begin by fixing a* ***base space*** $(\Omega, \Sigma, \mathbb{P})$*. The particular structure of the base space typically need not be specified; its primary function is to induce probability*

[1] Measure spaces can, with a little more effort, be restricted to non-measurable subsets as well, though we will not need this construction here.

measures on other spaces via pushforwards, implicitly functioning as a 'source of all randomness'.

*Having chosen a base space, measurable functions $x : \Omega \to X$, where X is a measurable space, are commonly called (X-valued) **random variables**, naturally associated with the pushforward measure $P_X = x_*(\mathbb{P})$. Given two random variables $x : \Omega \to X, y : \Omega \to Y$, their joint distribution $P_{X\times Y}$ is naturally the pushforward measure of the product $(x, y) : \Omega \to X \times Y$. In this way, it is common to define a measure on a product space by specifying a collection of functions out of Ω, the 'base space', i.e. a collection of random variables.*

Doubtless the central example of a measure on $\mathbb{R}$ is the Lebesgue measure, which reifies the intuitive concept of the length of a set.

Definition 2.2.7 (Lebesgue measure)**.** *The **Lebesgue measure** is defined as the unique*[2] *measure λ on the Borel sets of $\mathbb{R}$ satisfying*

1. *For all closed intervals $[a, b]$ where $a \leq b$, $\lambda([a, b]) = b - a$*
2. *Translation invariance: for all measurable A and $x \in \mathbb{R}$, $\lambda(A + x) = \lambda(A)$, where $A + x = \{a + x : a \in A\}$*

The Lebesgue measure of a (Lebesgue-measurable) set can be more explicitly obtained as the infimum of the lengths of all possible coverings of that set by a countable collection of open sets:

$$\lambda(A) = \inf \left\{ \sum_{n=1}^{\infty} (b_n - a_n) : a_n, b_n \in \mathbb{R} \ \wedge \ A \subseteq \bigcup_{n=1}^{\infty} (a_n, b_n) \right\}$$

In the same way that the Lebesgue measure on $\mathbb{R}$ reifies the intuitive concept of length, there is a unique Lebesgue measure on $\mathbb{R}^2$ reifying the intuitive concept of area, and so on in arbitrary $\mathbb{R}^N$, which can be defined analogously – though see the following Remark.

[2]Uniqueness is typically obtained by application of extension theorems such as [16, Section 13, Theorem A].

Remark 2.2.8 (Lebesgue algebra)**.** *For technical reasons, the Lebesgue measures on $\mathbb{R}^N$ are often considered as measures on $\mathcal{M}_N$, the **Lebesgue σ-algebra**, which is a refinement of the Borel algebra that also includes all subsets of null sets.*

*A measure space (X, Σ, μ) is called **complete** if, for any null sets $\mu(A) = 0$, all subsets of A are measurable $A \in \Sigma$ (and hence, by monotonicity, null as well). Define the **completion** $(X, \bar{\Sigma}, \bar{\mu})$ of a measure space as consisting of X, equipped with $\bar{\Sigma}$, the smallest σ-field including Σ such that all μ-null sets are measurable, and with $\bar{\mu}$, the unique measure on $\bar{\Sigma}$ that coincides with μ on Σ while assigning zero measure to all subsets of μ-null sets.*

As the completion of a measure space is unique, the distinctions between the Borel and Lebesgue algebras are often not so important. But one crucial difference that motivates the use of the Lebesgue algebras is the fact that the Lebesgue measure on the Borel sets of $\mathbb{R}$, multiplied by the Lebesgue measure on the Borel sets of $\mathbb{R}$, does not in fact yield the Lebesgue measure on the Borel sets of $\mathbb{R}^2$ (due precisely to the absence of these null sets), while if we consider the Lebesgue measure as a measure on the Lebesgue algebras, the corresponding statement does hold.

Definition 2.2.9 (Equality a.e.)**.** *It is natural in a measure-theoretic context to identify functions $f : X \to Y$ which differ only on (subsets of) null sets. Explicitly, if $f, g : X \to Y$ satisfy $f(x) = g(x)\ \forall x \in X \setminus N$ where N is null, we say that f and g are **equal almost everywhere (a.e.)**. Thus in a measure-theoretic context, when we consider spaces of measurable functions, the elements of these spaces will typically not be individual functions, but rather classes of measurable functions partitioned up to a.e.-equivalence.*

2.3 Lebesgue integration

Each measure space (X, Σ, μ) naturally corresponds to a particular integration operator on real-valued measurable functions $f : X \to \mathbb{R}$, where $\mathbb{R}$ is assigned the Borel algebra, known as the Lebesgue integral.

Definition 2.3.1 (Lebesgue integral)**.** *Defining the Lebesgue integral of indicator functions $1_A(x)$ to be the measure of the corresponding set:*

$$\int_X 1_A(x)\,d\mu(x) = \mu_X(A).$$

Then, as this integral is to be a linear operator, define the Lebesgue integral of 'simple functions' , i.e. weighted sums $\varphi(x) = \sum_{n=1}^{N} w_n 1_{A_n}(x)$ of indicator functions, to be

$$\int_X \varphi(x)\,d\mu(x) = \sum_{n=1}^{N} w_n \int_X 1_{A_n}(x)\,d\mu(x) = \sum_{n=1}^{N} w_n \mu(A_n).$$

Based on that, define the Lebesgue integral of any nonnegative measurable function $f : X \to \mathbb{R}_{\geq 0}$ as

$$\int_X f(x)\,d\mu(x) = \sup\left\{\int_X \varphi(x)\,d\mu(x) : \varphi \text{ simple}, 0 \leq \varphi \leq f\right\}$$

where this supremum exists. The Lebesgue integral of a measurable function $f : X \to \mathbb{R}$ which is not necessarily nonnegative is straightforwardly defined by partitioning its domain into negative and nonnegative regions.

It is easily shown that the class of Lebesgue-integrable functions is closed under linear combination, and that it includes all continuous functions. With more effort, it can be shown that this class includes all piecewise-continuous functions as well, and that the Lebesgue integral (with respect to the Lebesgue measure) and (proper) Riemann integral coincide whenever the latter is defined:

Theorem 2.3.2 ([22, Theorem 4.33])**.** *When $f : [a, b] \to \mathbb{R}$ is (proper) Riemann-integrable, we have*

$$\int_{[a,b]} f(x)\,d\lambda(x) = \int_a^b f(x)\,dx.$$

Once partitioned up to a.e.-equivalence, spaces of Lebesgue-integrable

functions are Banach in a natural way.

Definition 2.3.3 (Lebesgue space)**.** *Given a measure space (X, Σ, μ), define the* L^p ***spaces*** $\mathrm{L}^p(X, \mu)$*, for $p \in [1, \infty)$, as the spaces of measurable functions $f : X \to \mathbb{R}$ such that the Lebesgue integral $\int_X |f(x)|^p \, d\mu(x)$ is defined, partitioned up to a.e.-equivalence, and equipped with the p-norm $\|f\|_p = \left(\int_X |f(x)|^p \, d\mu(x)\right)^{1/p}$; for $p = \infty$, take $\|f\|_\infty = \operatorname{ess\,sup}_{x \in X} |f(x)|$, where the essential supremum is defined as the smallest $C > 0$ such that $f(x) < C$ for almost all x (i.e. outside of a μ-null set).*

Theorem 2.3.4 (Riesz-Fischer, [23, Theorem 4.26])**.** *The Lebesgue spaces are Banach; for separable X and $p < \infty$, the spaces $\mathrm{L}^p(X, \mu)$ are also separable.*

In the context of probability theory, Lebesgue integrals with respect to probability measures P, referred to as **expectations**, are often written using the notation

$$\mathbb{E}[f] = \mathbb{E}_P[f] = \int_X f(x) \, dP(x)$$

with the probability measure P often left implicit. For probability measures on $\mathbb{R}$, the expectation of the identity $f(x) = x$ (if defined) is called the **mean** μ of P, the expectation of $f(x) = (x - \mu)^2$ the **variance** σ^2 of P (and its root the **standard deviation**), and the expectations of the centralised and standardised polynomials $f(x) = \left(\frac{x-\mu}{\sigma}\right)^n$, for $n \geq 3$, the **(standardised) moments** of P.

Lebesgue integration has a natural relationship with the pushforward measure construction, referred to as the change-of-variables formula.

Theorem 2.3.5 (Change of variables)**.** *Given a measure space (X, Σ, μ) and measurable functions $f : X \to \mathbb{R}, g : \mathbb{R} \to \mathbb{R}$ such that both f and $g \circ f$ are Lebesgue-integrable, the* ***change-of-variables formula*** *relates the integral of g under the pushforward measure $f_*\mu$ to the integral of the composition $g \circ f$ under the original measure μ:*

$$\int_{\mathbb{R}} g(y) \, d(f_*\mu)(y) = \int_X g(f(x)) \, d\mu(x).$$

Its validity is immediate from the definitions of the pushforward measure and Lebesgue integration.

Fubini's theorem analogously clarifies the relationship between Lebesgue integration and the product measure construction.

Theorem 2.3.6 (Fubini, [16, Section 36, Theorem C])**.** *In the event that $(X, \Sigma_X, \mu_X), (Y, \Sigma_Y, \mu_Y)$ are σ-finite measure spaces, Fubini's theorem states that for Lebesgue-integrable $f : X \times Y \to \mathbb{R}$, integrals over the product space can be evaluated in either order:*

$$\int_{X\times Y} f(x,y)\, d(\mu_X \otimes \mu_Y)(x,y) = \int_X \left(\int_Y f(x,y)\, d\mu_Y(y) \right) d\mu_X(x) = \int_Y \left(\int_X f(x,y)\, d\mu_X(x) \right) d\mu_Y(y).$$

2.4 Densities

Fix a measure space (X, Σ, μ).

Definition 2.4.1 (Density)**.** *Any nonnegative $f : X \to \mathbb{R}_{\geq 0}$ which is Lebesgue-integrable with respect to μ defines, via Lebesgue integration, a new measure $f \cdot \mu$ on X given by*

$$(f \cdot \mu)(A) = \int_A f(x)\, d\mu(x).$$

*We say that f is the **density** of the measure $f \cdot \mu$, relative to the measure μ. When the ambient measure μ is clear from context, in particular when it is Lebesgue measure, it is common to neglect to mention it.*

Definition 2.4.2 (Absolute continuity)**.** *If, for some measure ν, a density f exists such that $\nu = f \cdot \mu$, we say that the measure ν is **absolutely continuous** with respect to μ. It is immediate that any such density f is unique up to a.e.-equivalence. Note also that if $\mu(A) = 0$, then it follows that $(f \cdot \mu)(A) = 0$ for all f.*

The Radon-Nikodym theorem provides a converse to the previous statement.

Theorem 2.4.3 (Radon-Nikodym, [16, Section 31, Theorem B])**.** *For σ-finite measures μ and ν, ν is absolutely continuous with respect to μ if and only if*

$\mu(A) = 0 \to \nu(A) = 0$ for all measurable A. The density f such that $\nu = f \cdot \mu$, unique up to a.e.-equivalence, is called the ***Radon-Nikodym derivative*** *of ν with respect to μ, and is often written using the differential notation $\frac{d\nu}{d\mu}$.*

Definition 2.4.4 (Probability density function)**.** *In probability theory, the density $p : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ of a probability measure P on $\mathbb{R}^n$ with respect to Lebesgue measure λ (if it exists) is referred to as its* ***probability density function (pdf)****; in order for P to be a probability measure, we must have $\int_{\mathbb{R}^n} p(x)\, d\lambda(x) = 1$. By the Radon-Nikodym theorem, probability measures which are absolutely continuous with respect to Lebesgue measure are uniquely specified by such a density.*

Example 2.4.5 (Gaussian density)**.** *The* ***Gaussian density****, with mean μ and standard deviation σ, is defined as the density on $\mathbb{R}$ given by*

$$\mathrm{N}\left(x \mid \mu, \sigma\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

with respect to Lebesgue measure.

The ***multivariate Gaussian density*** *on $\mathbb{R}^n$, with parameters $\mu \in \mathbb{R}^n$ and positive-definite $\Sigma \in \mathbb{R}^{n \times n}$, is defined as*

$$\mathrm{N}\left(x \mid \mu, \Sigma\right) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

with respect to Lebesgue measure on $\mathbb{R}^n$.

Definition 2.4.6 (Cumulative density function)**.** *Probability measures on $\mathbb{R}$ can also be uniquely specified by a* ***cumulative density function (cdf)*** *– whether the measure is absolutely continuous with respect to Lebesgue measure or not, in fact. The cumulative density function of a probability measure P on $\mathbb{R}$ is defined as the function $F : \mathbb{R} \to [0, 1]$ given by*

$$F(x) = P((-\infty, x]);$$

in particular, we have the limiting values $F(-\infty) = 0, F(\infty) = 1$.

In the event that P is absolutely continuous with respect to Lebesgue measure with Riemann-integrable density p, its cdf and pdf are related by

$$F(x) = \int_{(-\infty,x]} p(x)\, d\lambda(x) = \int_{-\infty}^{x} p(x)\, dx,$$

hence the name 'cumulative density function'.

Definition 2.4.7 (Quantile)**.** *As the cdf $F : \mathbb{R} \to [0,1]$ is always nondecreasing, we can define the generalised inverse*

$$Q(u) = \inf \{x \in \mathbb{R} : u \leq F(x)\};$$

this function is called the ***quantile function*** *of P. In the event that the cdf F is monotonically increasing, the quantile function Q is its inverse.*

Let P be a probability measure on $\mathbb{R}$, F its cumulative density function, and Q its quantile function. Now consider the pushforward of the Lebesgue measure λ on $[0,1]$ through the quantile function Q. The cumulative density function of the resulting pushforward measure is

$$(\lambda_* Q)((-\infty, x]) = \lambda(Q^{-1}(-\infty, x]) = \lambda((F(-\infty), F(x))) = \lambda((0, F(x)) = F(x),$$

where the second equality follows from the fact that $F(x)$ is nondecreasing (the others following from the definitions of the pushforward, cdf, and Lebesgue measure). It follows that any probability distribution on $\mathbb{R}$ can be obtained as a pushforward of the uniform distribution on the unit interval through its quantile function Q.

2.5 Spaces of probability measures

Let X be a topological space and let $\mathcal{P}X$ be the set of probability measures on the Borel sets of X. This section is concerned with the structure, topological and measurable, of the space $\mathcal{P}X$, and primarily follows [24, Appendix A].

The space $\mathcal{P}X$ is not closed under linear combination, as neither scalar

multiples nor sums of probability measures are themselves probability measures, but it can be equipped with several natural topologies. For our purposes, the most important of these will be the weak topology, under which $\mathcal{P}X$ is metrisable, separable when X is, completely metrisable when X is, and compact when X is [25]. This weak topology of probability measures, often called the weak-* topology for the sake of consistency with the terminology used in operator theory, can be defined in a number of equivalent ways. To motivate the operator-theoretic perspective, consider that each (probability) measure P defines a natural map $f \mapsto \int_X f\, dP$ on the space of continuous and bounded maps $\mathcal{C}(X, \mathbb{R})$.[3]

Definition 2.5.1 (Weak convergence)**.** *A sequence of Borel probability measures P_n on X is said to* ***converge weakly*** *to a limit P if, for all continuous bounded $f : X \to \mathbb{R}$, $\lim_{n\to\infty} \int_X f\, dP_n = \int_X f\, dP$.*

Definition 2.5.2 (Weak topology)**.** *The* ***weak topology*** *on measures is simply the topology that yields the weak convergence. As weak convergence of measures will be heavily used here, we will adopt the notation $P_n \xrightarrow{*} P$.*

The weak topology on $\mathcal{P}X$ can be also be characterised as the weakest topology under which the maps $P \mapsto \int_X f\, dP$ are measurable for all continuous bounded $f : X \to \mathbb{R}$.

Lemma 2.5.3 (Portmanteau, [24, Theorem A.2], [26, Theorem 11.1.1])**.** *The well-known* ***portmanteau lemma*** *gives a number of equivalent conditions for weak convergence, in the case that X is a metrisable Borel space:*

1. *$\lim_{n\to\infty} \int_X f\, dP_n = \int_X f\, dP$ for all bounded continuous f*
2. *$\lim_{n\to\infty} \int_X f\, dP_n = \int_X f\, dP$ for all bounded Lipschitz f*

[3] Note that in general, this map is neither a surjection nor an injection; it is one-to-one only in certain restricted settings. The most well-known result in this area is the Riesz-Markov-Kakutani representation theorem, which states that Radon measures on locally σ-compact Hausdorff spaces X are isometrically isomorphic to positive bounded linear functionals on the space of continuous real-valued functions on X that vanish at infinity. Regardless of whether the relationship is one-to-one, though, given only that measures have a natural interpretation as linear operators, concepts from operator theory can be applied to study them.

3. $\limsup_{n\to\infty} P_n(C) \leq P(C)$ *for all closed* C

4. $\liminf_{n\to\infty} P_n(U) \leq P(U)$ *for all open* U

5. $\lim_{n\to\infty} P_n(A) = P(A)$ *for all Borel sets* A *with measure-zero boundary* $P(\partial A) = 0$

2.6 Laws of large numbers

This overview will primarily follow [26, Chapters 8, 11], [22, Chapter 8]. We will begin by discussing several notions of convergence of random variables, and then discuss a few important settings in which this convergence can be shown, results that are typically called 'laws of large numbers'.

Fix a Borel space X and equip its countable product X^ω with a probability measure $\mathbb{P}$; we will take X^ω equipped with $\mathbb{P}$ as our base space, in the sense of Definition 2.2.6. Let $X_n : X^\omega \to X$ be each of the natural projections, and let $\mathbb{E}[\cdot] = \int_\Omega \cdot \, d\mathbb{P}$ represent integration under $\mathbb{P}$. Let $L : X^\omega \to X$ be an X-valued random variable. We will need to differentiate between three types of convergence of the random variables X_n to L, in increasing order of strength:

Definition 2.6.1 (Weak convergence)**.** X_n ***converges weakly*** *to* L *if, for all continuous bounded* $f : X \to \mathbb{R}$*,* $\lim_{n\to\infty} \mathbb{E}\left[f \circ X_n\right] = \mathbb{E}\left[f \circ L\right]$.

Note that the above definition is an application of our previous definition of weak convergence for measures to the pushforward distributions of each X_n; accordingly, it is natural to use the notation $X_n \xrightarrow{*} L$.

Definition 2.6.2 (Convergence in probability)**.** *When the Borel space* X *has metric* d*,* X_n ***converges in probability*** *to* L *if, for all* $\varepsilon > 0$*,* $\lim_{n\to\infty} \mathbb{P}\left[d(X_n, L) \geq \varepsilon\right] = 0$.

Definition 2.6.3 (Almost sure convergence)**.** X_n ***converges almost surely*** *to* L *if* X_n *converges to* L *pointwise on a set of measure 1, i.e.* $\mathbb{P}\left[\lim_{n\to\infty} X_n = L\right] = 1$.

In the case that $L(x_1, x_2, \ldots) = x$ is a constant random variable and X_n converges (in any of the above senses) to L, it is natural to say that X_n converges to the constant x. In the case of convergence to a constant, convergence in probability and convergence in distribution are equivalent; almost sure convergence remains stronger.

Consider now the case in which $\mathbb{P} = P^\omega$ is a product measure – in other words, the case in which X_n is a sequence of independent and identically-distributed (i.i.d) random variables, each distributed according to P. Let also $X = \mathbb{R}$, with the standard Borel structure.

Theorem 2.6.4 (Strong law of large numbers, [22, Theorem 8.32])**.** *The* ***strong law of large numbers (SLLN)*** *states that, provided that the mean $\mu = \int_{-\infty}^{\infty} x\, dP$ is defined, the sequence of empirical means $\frac{1}{n}\sum_{i=1}^{n} X_i$ converges to μ almost surely:*

$$\mathbb{P}\left[\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i = \int_{-\infty}^{\infty} x\, dP\right] = 1.$$

Correspondingly, the weak law of large numbers (WLLN) most often[4] *refers to the weaker statement that the empirical means converge weakly to the population mean μ.*

We finish this section with a statement of some basic results from empirical process theory that we will need. Consider again an arbitrary base measure $\mathbb{P}$ on the product space X^ω, with projections $X_n : X^\omega \to X$, and consider the resulting sequence of empirical measures as in Definition 2.2.2, i.e. the random variables $P_n : X^\omega \to \mathcal{P}X$ given by

$$P_n(x_1, x_2, \ldots)(A) = \frac{1}{n}\sum_{k=1}^{n} \delta_{X_n(x_1,x_2,\ldots)}(A).$$

As each of these empirical measures is a $\mathcal{P}X$-valued random variable, it is

[4]There is a constellation of related statements referred to as the laws of large numbers, which are focused on weakening the i.i.d. assumption we've adopted for this exposition. We will not need these alternative versions here.

natural to ask whether and how they they converge to the constant measure P. It is an immediate consequence of the strong law of large numbers that, when X_n are i.i.d. (i.e. $\mathbb{P} = P^\omega$), then for any bounded and continuous $f : X \to \mathbb{R}$, we have almost sure convergence

$$\mathbb{P}\left[\lim_{n\to\infty} \int_X f \, dP_n = \int_X f \, dP\right] = 1$$

of expectations under the empirical measures to the corresponding expectation under P. More interestingly, the following theorem shows that the quantification over the test function f made by the SLLN can be pulled inside $\mathbb{P}$. This theorem, and the perspective it represents, will be crucial in the coming chapters.

Theorem 2.6.5. *[26, Theorem 11.4.1] For standard Borel X, empirical measures almost surely converge weakly; that is,*

$$\mathbb{P}\left[P_n \xrightarrow{*} P\right] = 1.$$

The actual sequences produced by i.i.d. sampling from a standard Borel measure P, that is, have a certain special property: their empirical measures converge to P. We will refer to this property in the coming chapters as P-typicality.

Definition 2.6.6 (Typical sequences)**.** *Given a measure P on X, we will refer to the countably-infinite sequences of samples $(x_1, x_2, \ldots) \in X^\omega$ such that $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^n f(x_i) = \int_X f(x) \, dP(x)$ for all continuous bounded f as P-**typical** sequences. Theorem 2.6.5 is then the statement that, with probability one, i.i.d. sampling from P produces a P-typical sequence.*

As we will discuss in the next section, and in the remainder of this thesis, P-typical sequences can be produced for many P without assuming the ability to generate i.i.d. samples from P directly.

It will be useful in chapter 3 to generalise this notion of typicality to

weighted sequences of samples; this extension is obvious. We will say that the weighted sequence $((x_1, w_1), (x_2, w_2), \ldots)$, where $x_n \in X$ and $w_n \geq 0$, is P-typical if

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} w_i f(x_i)}{\sum_{i=1}^{n} w_i} = \int_X f(x)\, dP(x)$$

for all continuous bounded f; obviously Definition 2.6.6 is a special case when all weights w_i are unity.

2.7 Ergodic theory

Ergodic theory supplies a natural way to construct P-typical sequences using purely deterministic means; we will recap the basic definitions and results of the field, primarily following [25], beginning with the field's historical origins.

Example 2.7.1 (Irrational rotation)**.** *The historical development of modern ergodic theory began with the definition of* ***uniformly-distributed sequences****, also called* ***equidistributed sequences****, and the proof that irrational rotation gives such a sequence. This result was proved independently by Weyl, Sierpinski, and Bohl in 1909-1910 [27, 28, 29]. As defined by Weyl and his contemporaries, a uniformly distributed sequence was a sequence $x_n \in [0, 1]$ of points on the unit interval such that, for all open intervals $(a, b) \subseteq [0, 1]$, we have*

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} 1_{(a,b)}(x_i) = b - a$$

where 1_A is the indicator function on the set A. Noting that open intervals (a, b) are a generating set for the Borel σ-algebra on $\mathbb{R}$, it follows immediately that this definition is a special case of Definition 2.6.6, in the case that $X = [0, 1]$ and P is the uniform distribution.

To Weyl, Sierpinski, and Bohl, the archetypal example of (nontrivial) uniformly-distributed sequences were the ***irrational rotations***

$$x_n = n\alpha \mod 1,$$

where α is irrational; in fact, they showed that these sequences are uniformly distributed if and only if α is irrational.

In its modern form, ergodic theory generalises Weyl's, Sierpinski's, and Bohl's original result, broadening its setting from only the uniform distribution on the unit interval to arbitrary probability measures on Borel spaces, and providing a sufficient condition for constructing such sequences: ergodic transformations.

Definition 2.7.2 (Measure-preserving dynamical system)**.** *A **measure-preserving dynamical system** refers to a tuple (X, P, T) where:*

- *X is a standard Borel space;*
- *P is a probability measure on the Borel sets of X called the **stationary distribution** of the dynamical system;*
- *$T : X \to X$ is a measurable function on X under which P is invariant: that is, $T_*P = P$.*

Definition 2.7.3 (Ergodicity)**.** *A measure-preserving dynamical system (X, P, T) is called **ergodic** if the only T-invariant Borel subsets of X are either P-null or P-full; that is, if for all Borel $A \subseteq X, T^{-1}(A) = A \to P(A) \in \{0, 1\}$.*

Remark 2.7.4. *Ergodic measures are naturally interesting from a geometric perspective, as they are extremal points of the set of invariant measures. That is: let $M(X, T)$ represent the set of probability measures which are invariant under a transformation $T : X \to X$, and note that this set is obviously closed under convex combination. The extremal points of this set, i.e. those which cannot be written as nontrivial convex combinations $P = \alpha P_1 + (1 - \alpha)P_2$ ("trivial" here meaning that either $\alpha = 0, 1$ or $P_1 = P_2$) are then precisely the probability measures with respect to which T is an ergodic transformation [25, Proposition 4.3.2].*

Theorem 2.7.5 (Birkhoff, [30, Theorem 9.6])**.** *The **Birkhoff ergodic theorem** states that, for any ergodic dynamical system (X, P, T), P-almost all*

initial points generate P-typical sequences in the sense of Definition 2.6.6. That is, where P_n refer to the empirical measures of Definition 2.2.2,

$$P\left\{x \in X : P_n(x, T(x), \ldots, T^{n-1}(x)) \xrightarrow{*} P\right\} = 1.$$

As a result of Birkhoff's theorem, if our aim is to produce samples from a given probability distribution P, it suffices to find a transformation T which, with P, forms an ergodic dynamical system, and choose any initial point $x \in X$ which belongs to this privileged measure-1 subset.

This final condition of the Birkhoff ergodic theorem, that of avoiding a measure-zero subset of initial points that are not P-typical, is difficult to relax. However, the following result shows one approach:

Proposition 2.7.6 ([25, Proposition 6.1.1])**.** *For compact X and continuous $T : X \to X$, the following are equivalent:*

1. *All initial points $x \in X$ yield P-typical sequences;*
2. *T is* uniquely *ergodic – that is, P is the only measure with respect to which T is invariant*[5]*.*

Having introduced measure-preserving dynamical systems, it is natural, and will be useful, to define a notion of homomorphism and isomorphism between them. In ergodic theory, the natural notion of isomorphism is referred to as conjugacy.

Definition 2.7.7 (Conjugacy)**.** *Let (X, P_X, T_X) and (Y, P_Y, T_Y) be two (Borel) measure-preserving dynamical systems. A (Borel) homomorphism consists of a continuous map $f : X \to Y$ such that $P_Y = f_* P_X$ and such that the following diagram commutes:*

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \downarrow{\scriptstyle T_X} & & \downarrow{\scriptstyle T_Y} \\ X & \xrightarrow{f} & Y \end{array}$$

[5]Note that if P is the only measure with respect to which T is invariant, then it is trivially an extremal point of the set of invariant measures, and thus the system is ergodic as well.

The systems $(X, P_X, T_X), (Y, P_Y, T_Y)$ *are* ***(topologically) conjugate*** *if there exists a (bicontinuous) isomorphism* $f : X \to Y$ *such that* f *and* f^{-1} *are each homomorphisms of measure-preserving dynamical systems.*

Example 2.7.8 (Logistic map)**.** *Consider the dynamical system on* $[0, 1]$ *given by* $T(x) = rx(1 - x)$*, the* ***logistic map****, in particular in the case* $r = 4$*. This case was analyzed by von Neumann in his earliest overview of sampling techniques [1], though he attributes to Stanisław Ulam the original suggestion. In modern terms, this transformation* T *is well-known to be ergodic with respect to the arcsine distribution; to understand why this is, simply verify that the homeomorphism* $x = f(y) = \sin^2(2\pi y)$ *on the unit interval shows the logistic map (for* $r = 4$*) to be conjugate to the* ***dyadic map*** $S(y) = \begin{cases} 2y & 0 \leq y < \frac{1}{2} \\ 2y - 1 & \frac{1}{2} \leq y < 1 \end{cases} = (2y) \mod 1$*. That the uniform distribution on the unit interval is invariant and ergodic under this transformation is easily verified. As a result, it follows that the logistic map with* $r = 4$ *is invariant and ergodic under the pushforward of the uniform distribution through* $f^{-1}(x) = \frac{1}{2\pi} \sin^{-1} \sqrt{x}$*, which yields a distribution known as the arcsine distribution.*

The dyadic map is also called the ***bit-shift map****, as if we represent points in* $[0, 1]$ *using their binary expansions* $y = \sum_{n=1}^{\infty} b_n 2^{-n}$*, a little thought quickly shows that the dyadic map can be written as a bit-shift operation* $S(y) = \sum_{n=1}^{\infty} b_{n+1} 2^{-n}$*. For rational initial points* y*, the sequence given by iterating the bit-shift map is necessarily periodic with a finite period (this being the definition of a rational number), and so cannot possibly be equidistributed with respect to the uniform distribution on the unit interval. Despite this, theorem 2.7.5 tells us that for almost all points on the unit interval, iterating the bit-shift map yields uniformly-distributed sequences – though as noted by von Neumann in [1], this makes it impractical for use as a pseudorandom number generator.*

As one would expect, isomorphisms of dynamical systems preserve most

properties of dynamical systems studied in the field. We will only need one result in this area.

Proposition 2.7.9 (Homorphisms preserve ergodicity)**.** *Let (X, P_X, T_X) be an ergodic system, let (Y, P_Y, T_Y) be a dynamical system (not necessarily ergodic), and let $f : X \to Y$ be a homomorphism of dynamical systems. It follows that (Y, P_Y, T_Y) is ergodic.*

Proof of Proposition 2.7.9.
In order for T_Y to be ergodic with respect to $P_Y = f_* P_X$, we must know that for all measurable $B \subseteq Y$, $T_Y^{-1}(B) = B$ implies $f_* P_X(B) = 0$ or $f_* P_X(B) = 1$. Consider the inverse image $A = f^{-1}(B)$. Then, as f is a homomorphism, $T_X^{-1}(A) = T_X^{-1}(f^{-1}(B)) = f^{-1}(T_Y^{-1}(B))$. Then, assuming that $T_Y^{-1}(B) = B$, we immediately obtain $T_X^{-1}(A) = A$; as T_X is ergodic, we have $P_X(A) = 0$ or $P_X(A) = 1$. By the definition of the pushforward $f_* P_X(B) = P_X(f^{-1}(B)) = P_X(A)$, it follows that T_Y is ergodic with respect to P_Y. □

As the following chapter will involve some discussion of products and iterates of dynamical systems, it will be necessary to introduce the definition of a weak mixing dynamical system, which is a useful strengthening of ergodicity.

Definition 2.7.10 (Weak mixing)**.** *A measure-preserving dynamical system (X, P, T) is called **weak mixing** if, for all measurable $A, B \subseteq X$, we have*

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \left| P\left(T^{-i}(A) \cap B\right) - P(A)P(B) \right| = 0.$$

*Weak mixing is, naturally, weaker than **strong mixing**, which requires the stronger but simpler condition $\lim_{n\to\infty} P(T^{-n}(A) \cap B) = P(A)P(B)$; we will not need the latter concept here.*

It is immediate from the definition that if (X, P, T) is weak mixing, then all iterates $(X, P, T^k), k \geq 1$, of the system are weak mixing as well [25, Exercise 7.1.3]. It is also easy to show that

Proposition 2.7.11 (Weak mixing $\implies$ ergodicity)**.** *If (X, P, T) is weak mixing, then it is ergodic.*

Proof of Proposition 2.7.11.

Assume that (X, P, T) is weak mixing, let A be a T-invariant set, and choose $B = \bar{A} = X \setminus A$ the complement of A. We will show that the set A is necessarily either of zero or full measure, which is the definition of ergodicity. Applying the definition of weak mixing, as A is invariant under T (and hence T^i for any $i \geq 1$), we quickly obtain

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \left| P(T^{-i}(A) \cap \bar{A}) - P(A)P(\bar{A}) \right| = 0$$

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \left| P(A \cap \bar{A}) - P(A)P(\bar{A}) \right| = 0$$

$$P(A)P(\bar{A}) = 0;$$

therefore, the system (X, P, T) is ergodic. □

Less obvious from the definition of weak mixing is the connection between weak mixing of a system and ergodicity of its product system, which follows.

Proposition 2.7.12 ([25, Proposition 7.1.11])**.** *The following conditions are equivalent:*

1. *(X, P, T) is weak mixing;*
2. *$(X \times X, P \times P, T \times T)$ is weak mixing;*
3. *$(X \times X, P \times P, T \times T)$ is ergodic.*

The extension to all finite products is immediate.

Simply combining the above results regarding weak mixing of products and iterates, as well as Birkhoff's ergodic theorem, we can obtain the following result:

Proposition 2.7.13. *If (X, P, T) is weak mixing, then for any $k \geq 1$, P^k-almost all initial points $(x_1, \ldots, x_k) \in X^k$ of the iterated product system $(X^k, P^k, T^k \times \ldots \times T^k)$ generate P^k-typical sequences.*

2.8 Stochastic processes

An important application of our sampling language will be verifying samplers for stochastic processes. Stochastic processes, essentially structured collections of random variables that can model time-changing random quantities, have been objects of formal study since Kolmogorov's axiomatisation of probability theory in the early 1930s, but the modern presentation of the theory of stochastic processes dates to the early 1950s, drawing on the work of Joseph L. Doob, Paul Lèvy, Eugene Dynkin, and others [31].

In what follows, fix a probability space $(\Omega, \Sigma, \mathbb{P})$, a set T, and a Borel space S.

Definition 2.8.1 (Stochastic process)**.** *A T-indexed, S-valued **stochastic process** is a T-indexed collection of random variables $X : T \to (\Omega \to S)$.*

As the terminology 'process' suggests, the index sets T most often considered, $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$, are typically interpreted as indexing time. However, the concept of a stochastic process is more general than time-varying processes, extending to any indexed collection of random variables.

Definition 2.8.2 (Law)**.** *A stochastic process X naturally induces a measurable map $X : \Omega \to S^T$, where S^T is assigned the product σ-algebra. The pushforward $X_*(\mathbb{P})$, which we will write as P_X, is known as the **law** of the process X.*

Note that as the product σ-algebra is very coarse, the law of a stochastic process contains a limited amount of information about the process itself; stochastic processes that differ in important ways can have the same law.

Definition 2.8.3 (Marginals)**.** *Let X be a T-indexed, S-valued stochastic process. For any $(t_1, \ldots, t_n) \in T^n$, let $\mathrm{ev}_{(t_1,\ldots,t_n)} : S^T \to S^n$ represent the*

functional $\mathrm{ev}_{(t_1,\ldots,t_n)}(f) = (f(t_1),\ldots,f(t_n))$*, which evaluates its input* f *at a specified collection of points in* T*. Pushing the law of* X *through the evaluation functional gives the joint distribution* $P_{(t_1,\ldots,t_n)}$ *of the random variables* $(X(t_1),\ldots,X(t_n)) : \Omega \to S^n$*. These random variables are called the* ***(finite-dimensional) marginals*** *of the stochastic process* X*. If the marginals of two* T*-indexed,* S*-valued stochastic processes* $X(t), Y(t)$ *all have the same distributions, we will say that the two processes are* ***identically distributed****.*

It can be shown that two stochastic processes induce the same law if and only if they have the same marginal distributions [32].

Consider the finite-dimensional marginals $P_{(t_1,\ldots,t_n)}$ of a T-indexed, S-valued process X. For any permutation π on $\{1,\ldots,n\}$ and any topological space X, let $\pi_X : X^n \to X^n$ represent the corresponding automorphism $\pi_X(x_1,\ldots,x_n) = (x_{\pi(1)},\ldots,x_{\pi(n)})$. It is straightforward to show that for any stochastic process, the collection of marginal distributions satisfies:

1. For every permutation π, $P_{(t_{\pi(1)},\ldots,t_{\pi(n)})} = (\pi_S)_* P_{(t_1,\ldots,t_n)}$.

2. For every $m < n$ and Borel A: $P_{(t_1,\ldots,t_m)}(A) = P_{(t_1,\ldots,t_n)}(A \times S^{n-m})$.

The Kolmogorov extension theorem states that, under broad assumptions, this correspondence goes both ways, and the finite-dimensional marginals are sufficient to identify the process (up to equivalence in law).

Theorem 2.8.4 (Kolmogorov, [32])**.** *If* S *is a standard Borel space and* $P_{(t_1,\ldots,t_n)}$*,* $n \in \mathbb{N}$*,* $t_i \in T$ *is a collection of probability measures on* S^n *satisfying the above conditions, then there exists a unique probability measure* P_X *on* S^T *(equipped with the product algebra*[6]*) whose finite-dimensional marginals are* $P_{(t_1,\ldots,t_n)}$*, that probability measure being the law of our stochastic process.*

Bochner's theorems [34, Ch.2] are alternatives to Kolmogorov's theorem in which assumptions are made about the structure of the space S and

[6] As mentioned earlier, the coarseness of the product algebra is often limiting. Many applications, in particularly any requiring continuity, will require extending this law to a finer σ-algebra; see [33, Appendix A.2]).

where the system of products above is replaced by the more general notion of a projective system. One version of the theorem states:

Theorem 2.8.5 (Bochner, [34, Ch.2, Thm 5.5]]). *If $(I, \leq)$ is a directed set, $(X_i)_{i\in I}$ is a system of locally compact spaces, and for each $i \leq j$, $\pi_{ji} : X_j \to X_i$ are surjections such that for $i \leq j \leq k$, $\pi_{ki} = \pi_{ji} \circ \pi_{kj}$ – in other words if $(X_i, \pi_{ji})_{i,j\in I}$ is a projective system – then $\mathcal{P}^0(\varprojlim_i X_i) \cong \varprojlim_i \mathcal{P}^0 X_i$, where $\mathcal{P}^0 X$ is the set of* regular[7] *Borel probability measures on X.*

This theorem strengthens Kolmogorov's for sufficiently nice spaces. Suppose all measures on S are regular (e.g. if S is Polish) and consider the set $I = \bigcup_n T^n$ of all T-tuples. For every $t = (t_1, \ldots, t_m)$ and $t' = (t'_1, \ldots, t'_n)$ in I define the relation $t \leq t'$ whenever $t = t' \circ i_{t,t'}$ for some injection $i_{t,t'} : m \to n$ mapping the position of each t_i in the tuple t to its position in the tuple t'. Then $(I, \leq)$ is directed and we can define a projective system by putting $S_t = S^n$ and $\pi_{t',t} : S^n \to S^m, (s_1, \ldots, s_n) \mapsto (s_{i_{t,t'}(1)}, \ldots, s_{i_{t,t'}(m)})$. Bochner's theorem then gives

$$\varprojlim_t \mathcal{P} S_t = \varprojlim_t \mathcal{P}^0 S_t \cong \mathcal{P}^0(\varprojlim_t S_t) = \mathcal{P}^0(S^T).$$

Note that every $\pi_{t',t}$ in this system is the composition of a permutation and a projection. From this observation, it is not hard to see that an element of the left-hand-side of the equation is precisely a collection of marginals satisfying the conditions 1. and 2. of Kolmogorov's extension theorem. On the right-hand-side we gain regularity[8]. We will return to this system in section 4.3.

Just as a (real-valued) probability distribution is frequently summarised

[7] A measure μ on a Hausdorff space X is called *regular* if, for all measurable A, $\mu(A) = \sup\{\mu(K) : K \subseteq A \wedge K \text{ compact}\} = \inf\{\mu(U) : U \supseteq A \wedge U \text{ open}\}$. Since all Borel measures on a Polish space are regular, $\mathcal{P}^0 X_i = \mathcal{P} X_i$, the space of *all* Borel measures, when X_i is Polish. However, $\varprojlim_i X_i$ will in general not be Polish (though it is when I is countable), and thus $\mathcal{P}^0(\varprojlim_i X_i) \neq \mathcal{P}(\varprojlim_i X_i) in general$.

[8] To see that $\varprojlim_t S_t = S^T$ note that $T = \varinjlim_t l(t)$ where $l(t) = n$, the set with n elements, if t is n-dimensional (every set is the injective limit of its finite subsets), and since the hom functor $\hom(-, S)$ turns colimits into limits, we get $S^T = \hom(\varinjlim_t l(t), S) = \varprojlim_t \hom(l(t), S) = \varprojlim_t S_t$ as sets. It is straightforward to check that the topologies on S^T and $\varprojlim_t S_t$ are the same.

by its mean and (co)variance, so a (real-valued) stochastic process can be summarised by its mean *function* and covariance *function*.

Definition 2.8.6 (Mean and covariance function)**.** *Define the* ***mean function*** $\mu : T \to \mathbb{R}$ *of an* $\mathbb{R}$*-valued,* T*-indexed stochastic process* $X(t)$ *as the map*

$$\mu(t) = \mathbb{E}\left[X(t)\right] = \int_{\Omega} X(t)(\omega)\, d\mathbb{P}(\omega)$$

and the ***covariance function*** *(also often called the* ***covariance kernel****)* $\kappa : T \times T \to \mathbb{R}$ *as the map*

$$\kappa(t_1, t_2) = \mathrm{Cov}\left[X(t_1), X(t_2)\right] = \mathbb{E}\left[(X(t_1) - \mu(t_1))(X(t_2) - \mu(t_2))\right]$$

where each of these integrals exist. We will refer to $\sigma^2(t) = \kappa(t, t)$ *as the* ***variance function****.*

Example 2.8.7. *A* ***Gaussian process*** *is an* $\mathbb{R}$*-valued stochastic process*[9] *such that all finite-dimensional marginals are Gaussian. Because a normal distribution is fully specified by its mean and covariance, it immediately follows that all the marginal definitions of a Gaussian process are fully specified by a choice of mean and covariance* function*.*

For example, the mean and covariance functions $\mu(t) = 0, \kappa(t_1, t_2) = \begin{cases} \sigma^2 & t_1 = t_2 \\ 0 & t_1 \neq t_2 \end{cases}$ *specify an i.i.d. collection of standard Gaussian random variables with variance* σ^2*; the functions* $\mu(t) = 0, \kappa(t_1, t_2) = \min(t_1, t_2)$ *specify the* ***Wiener process****, or* ***Brownian motion****, a central object of study in finance and physics, among other fields; and the square-exponential covariance function* $k(t_1, t_2) = \exp\left(-\frac{1}{2\sigma^2}\left\|t_1 - t_2\right\|^2\right)$ *gives a smooth Gaussian process commonly used in Gaussian process regression.*

[9] The generalisation to $\mathbb{R}^N$-valued processes, and further, is mostly straightforward, but we will not use it here.

Chapter 3

Deterministic stream-semantics for higher-order probabilistic programming languages

In this chapter, we introduce a language, semantics, and calculus for verifying a certain class of probabilistic programs: those intended to solve problems of approximate sampling. We will begin by introducing the problem of approximate sampling and discussing its importance, and then we will motivate some of the sampler operations our language will make use of before we give their formal syntax and semantics. After formalising our language's syntax and semantics, we can then go on to discuss the verification of sampling techniques within this language.

Approximate sampling. Our purpose for developing this language is to be able to state and analyse algorithms for *approximate sampling*. Given a desired target probability distribution P, we want to verify that a proposed sampler, i.e. program, is able to generate samples which, informally speaking, behave as if they are taken from target distribution P, by making use of other samplers and transforming their samples in various ways. This informal idea that our samples must 'behave as if they are taken from P' will be formalised here as P-typicality Definition 2.6.6, i.e. based on weak convergence. This formulation of the problem of approximate sampling would have been well-understood by

von Neumann and his contemporaries; what they may not have understood quite as well, without our hindsight, is its centrality within a wide array of fields – many of which (such as machine learning) did not exist at that time.

A particularly important class of approximate sampling problems are those posed by Bayesian inference. Bayesian inference itself can be understood as a method for transforming the problem of statistical inference – of making probabilistic inferences based on data – into a problem of approximate sampling. Our aim in Bayesian inference is to generate samples from a *posterior distribution*, a distribution over some set of parameters z which are to be estimated, conditioned on observed data x. This posterior distribution $p(z \mid x)$ is specified by choosing a prior distribution $p(z)$ over parameter values, and by a likelihood $p(x \mid z)$, i.e. Markov kernel, which gives the probability of observing a set of data given certain parameter values. Having chosen a prior and a likelihood, and having observed some data, the posterior distribution then follows from Bayes' theorem[1]

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)} = \frac{p(x \mid z)p(z)}{\int_Z p(x \mid z)p(z)\,dz},$$

but cannot in general be obtained in closed form.

Instructive examples of Bayesian models include: filtering problems, in which case $z = (z_1, \ldots, z_T)$ are latent states with a transition model $p(z_t \mid z_{t-1})$ and observation model $p(x_t \mid z_t)$, and our aim is to characterise the distribution of these latent states conditional on the observations $x = (x_1, \ldots, x_T)$; i.i.d. regression problems, in which case we observe covariates $p(z_t \mid \theta)$ and response variables $p(x_t \mid z_t, \theta)$, and we aim to estimate some regression parameters θ, to which we assign a prior $p(\theta)$; and joint filtering and parameter estimation problems, such as parameter estimation for state-space models and hidden Markov models, in which case we observe temporal data $z = (z_1, \ldots, z_T)$ according to a transition model $p(z_t \mid z_{t-1}, \theta)$ and noisy observations $p(x_t \mid z_t, \theta)$

[1] We assume for simplicity here that the necessary conditions hold such that Bayes' theorem can be stated in terms of probability densities dominated by common ambient measures dz and dx.

with parameters θ, over which we take a prior $p(\theta)$, and our aim is to obtain either the posterior distribution $p(\theta \mid x)$ over parameters with respect to our observations, or the joint posterior distribution $p(z, \theta \mid x)$ of latent states and parameters with respect to our observations.

In any case, as the posterior distribution in the above problems cannot in general be calculated analytically, it is natural to approximate properties of the posterior distribution, such as the posterior mean or variance, by Monte Carlo – that is to say, by using approximate samples from the posterior distribution. This is the sense in which Bayesian inference recasts the problem of statistical inference – of characterising the values of some unknown parameters using observed data – as a problem of approximate sampling.

Our language is designed to be a setting for verifying approximate samplers. In particular, it contains built-in operations corresponding to some of the essential operations of approximate sampling. It will therefore be worthwhile to introduce a few approximate sampling techniques in advance, to illustrate the need for certain language constructs; in particular, we will quickly introduce four. These operations will be discussed in much more detail, and verified, in section 3.3.2.

1. Inverse-transform sampling we have already discussed in section 2.4. Consider the problem of generating a random variable distributed according to desired probability measure P on $\mathbb{R}$. In section 2.4, we showed that if F^{-1} is the (generalised) inverse of the cumulative distribution function $F(x) = P((-\infty, x])$, then if a random variable U is distributed uniformly on the unit interval, it follows that $F^{-1}(U)$ is distributed according to P. Therefore, given the ability to invert the desired cumulative function, exact samples can be taken from any desired probability distribution on $\mathbb{R}$. This technique was well-known to von Neumann and his contemporaries, and was discussed in their monograph [1]. Inverse-transform sampling is typically infeasible outside of the univariate case, but fortunately, multivariate sampling problems can sometimes be reduced to a collection of univariate sampling problems;

for example, Cholesky decomposition of the covariance matrix allows one to transform n independent Gaussian samples into one sample from a Gaussian distribution on $\mathbb{R}^n$ with arbitrary mean and covariance matrix.

2. Pseudorandom number generation we have also already touched on in section 2.7. Approximate samples from a probability distribution P on a Borel space X can be generated by specifying a measure-preserving dynamical system (X, P, T), which consists of a map $T : X \to X$ with respect to which the desired target distribution P is invariant and ergodic. It follows by the Birkhoff ergodic theorem that P-almost all initial points $x_0 \in X$ yield P-typical sequences by iterating T – see sections 2.6 and 2.7.

Von Neumann, in [1], discussed two such samplers in detail. First, he discussed the logistic map, in which case T is the function $T(x) = 4x(1 - x)$ on $X = [0, 1]$, which was discussed in Example 2.7.8. That the logistic map was ergodic with respect to the arcsine distribution, and that the logistic map was conjugate to the bit-shift distribution, were both well-known to von Neumann. Second, von Neumann mentions the middle-square method, an early pseudorandom number generator which has since fallen out of favour. The samples from this PRNG are formed by squaring a four-digit number, and then dropping all but the middle four digits. For the reasons discussed in section 2.7, it is natural to require that our samplers be formed from *ergodic* systems – which the middle-square method clearly cannot be, as it was well-known even by von Neumann's time that it has multiple short cycles (e.g. $3600 \to 9600 \to 1600 \to 5600 \to 3600 \to ...$).

3. Rejection sampling we have not yet discussed. Let P be our target measure, and assume the ability to sample from a proposal measure Q, both defined on a Borel space X; assume also the ability to generate independent uniform samples U on the unit interval. In order for rejection sampling to be possible, we must assume that the target P is absolutely continuous with respect to the proposal Q, and that we can evaluate (at least up to some constant of proportionality) the resulting density $f(x) \propto \frac{dP}{dQ}(x)$. Let K be

any upper bound $f(x) \leq K$, i.e. any K such that $f(x) \leq K$ for Q-almost all x. To perform rejection sampling, we sample independently $x \sim Q$ from our proposal and u from the uniform distribution on the unit interval. If $u \leq \frac{f(x)}{K}$, then x is distributed according to our target distribution P; if $u > \frac{f(x)}{K}$, then we discard the samples x, u and try again. This technique may in fact have been introduced in [1].

4. Importance sampling is in essence a continuous variant of rejection sampling, in which rather than rejecting or accepting proposed samples, we instead assign each sample a positive-valued *weight*. Again, let P be our target measure and Q our proposal, assume P is absolutely continuous with respect to Q, and let $f(x) \propto \frac{dP}{dQ}(x)$ be proportional to the resulting density. To perform (self-normalised) importance sampling [35], generate samples $x_i \sim Q$ from the proposal, and for each sample, first compute the *unnormalised* weight $\tilde{w}_i = f(x_i)$. Having taken n such samples, we can then compute the *normalised* importance weights $w_i = \tilde{w}_i / \sum_{k=1}^{N} \tilde{w}_k$. These unnormalised samples can be used to approximate integrals with respect to the target distribution $\lim_{n\to\infty} \sum_{i=1}^{n} w_i g(x_i) \overset{a.s.}{\to} \int_X g \, dP$, which is another way of saying that these *weighted* sequences of samples are almost surely P-typical (see Definition 2.6.6).

Sampler types. In this chapter, we will formalise each of the above constructions as procedures that output *samplers*. To that end, our language introduces a sampler type, in which samplers are understood as weighted sequences. Samplers are defined as sequences in order to incorporate pseudorandom generation of samples, and are weighted in order to incorporate importance sampling (and rejection sampling, once one recognises that assigning a sample zero weight is essentially the same as rejecting it). Traditional pseudorandom number generators are samplers, as are more complex samplers produced by applying any of the sampling techniques or operations previously discussed.

Sampler operations. Section 3.1 introduces a set of sampler operations. These sampling techniques are chosen primarily to implement sampling tech-

niques like the ones discussed above, and are defined coinductively. For example, the sampler operation `prng`, 'pseudorandom number generator', constructs a sampler using a measure-preserving dynamical system. The sampler operation `map` applies the same operation to each sample, making it analogous to the *pushforward* operation in measure theory (necessary to implement, for instance, inverse-transform sampling). The operation `reweight` continuously reweights our samples; this is used to implement both importance sampling and rejection sampling.

We include also a product of samplers, as well as certain stream operations such as 'thinning' samplers (i.e. dropping certain samples). Particular attention will be paid to the operation we call the 'sampler self-product', which implements the operation of taking multiple adjacent samples from the same sampler. We provide a denotational and operational semantics for our sampling language, and prove an adequacy relationship between the two.

Verification. Rather than existing as objects within our language, probability measures are relegated to a meta-theoretic status used in program verification; the natural relationship between samplers and probability measures, *targeting*, will be defined in section 3.3. We introduce a calculus governing this targeting relation, the application of which can verify the above-mentioned sampling techniques, among others. We also introduce an effective equivalence relation between samplers, in order to aid sampler verification. With this said, we are ready to introduce our formal syntax and semantics.

3.1 Language

3.1.1 Syntax

Formally, our language is implemented as a λ-calculus with a notion of subtype $\lhd$, a call-by-name evaluation strategy, and a type constructor Σ for samplers.

The types of our language are generated by the mostly standard grammar

in fig. 3.1a, choosing as the set of ground types

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

Our ground types include the natural, real and nonnegative real numbers, Boolean values, and, more interestingly, certain important sets of pairs of reals, such as $<^{-1}(1), \neq^{-1}(0)$, and the rest. To understand the meaning of these sets, consider for a moment the operator $<$ as a function $< \ : \mathbb{R} \times \mathbb{R} \to \{0, 1\}$; the inverse image $<^{-1}(1)$ then refers to set of pairs of reals whose first component is strictly stronger than the second, and $<^{-1}(0)$ to its complement. These special sets, we will see, are included as ground types in order to give sensible semantics to the binary operations $\leq, <, \geq, >, =$, and $\neq$.

The only unusual type constructors that appear in fig. 3.1a are the *pullback types* ${}_s\mathtt{T}_t$ and the *sampler types* $\Sigma\,\mathtt{T}$. Pullback types ${}_s\mathtt{T}_t$, as will soon be explained, will be interpreted as (categorical) pullbacks of the term $s : \mathtt{A} \to \mathtt{T}$ along the term $t : \mathtt{B} \to \mathtt{T}$, though in chapter 4 we will better illustrate this construction as a dependent type. Sampling types $\Sigma\,\mathtt{T}$ will be defined as the coinductive (stream) types defined by the (syntactic) functors $\mathtt{T} \times \mathtt{R}^+ \times -$, allowing samplers to 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 will need, and to highlight the central role played by samplers, we choose not to add generic coinductive types to the language.

Terms

Figure 3.1b presents the grammar generating the set Expr of terms in our language. We will refer to terms produced by the small sub-grammar fig. 3.1c as **values**; this sub-grammar will be used in the construction of our operational semantics in section 3.1.2.

We include a set Func of built-in functions which come equipped with typing information $f : \mathtt{T} \to \mathtt{G}$, where $\mathtt{G}$ is a ground type. Some built-in functions will be continuous w.r.t. to the standard topologies, such as the

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

(a) Type grammar

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

(b) Term grammar

$$v ::= x \in \text{Var} \mid b \in \{\mathtt{True}, \mathtt{False}\} \mid n \in \mathbb{N} \mid r \in \mathbb{R} \mid (v, v) \mid \mathtt{in}_i\,(v) \mid \lambda x\colon \mathtt{T}.\ v$$

(c) Value grammar

Figure 3.1: Grammars

addition operation $+ : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$, but others will be discontinuous with respect to the standard topologies, such as the comparison operators $\{\leq, <, \geq, >, =, \neq\} : \mathtt{R} \times \mathtt{R} \to \mathtt{B}$. Dealing with such functions is the main reason for adding coproducts to the grammar, as we will discuss in section 3.1.3.

We define if-statements as simple Boolean cases, employing the syntactic sugar

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

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

- The operation $\mathtt{prng}(f, t)$ is used to construct a sampler as a pseudo-random number generator, using an initial value t and a deterministic

endomap f.

- $s \otimes t$ represents the product of samplers s, t.
- The syntax $\texttt{map}(f, t)$ 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 $\texttt{reweight}(f, t)$ applies the reweighting scheme f to the sampler t to form a new sampler.
- Given a sampler t, the operation $\texttt{hd}(t)$ returns the first sample produced by t, $\texttt{wt}(t)$ the weight of the first sample produced by t, and $\texttt{tl}(t)$ returns the sampler t but with its first sample-weight pair dropped.
- The operation $\texttt{thin}(n, t)$, 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 precise meaning of these language constructs will be made clearer when we introduce their semantics in sections 3.1.2 and 3.1.3. The purpose of introducing them, if not already clear, will be further illustrated in section 3.3, when we use them to represent and then verify certain common sampling techniques.

Note that our language does not have an operation $\texttt{sample}(s)$ for a sampler s, as our samplers do not have internal state. This operation can nevertheless be mimicked by using the 'sample' $\texttt{hd}(s)$, and then `let`-binding all subsequent occurrences of s to $\texttt{tl}(s)$.

Well-formed terms

Our typing system is mostly standard and presented in fig. 3.2a, and the *subtyping relation* $\triangleleft$ on types is the reflexive-transitive closure of the relation generated by the rules of fig. 3.2b. The purpose of the subtyping relation, as we will discuss, is to encode topological information which will allow the interpretation of functions which are discontinuous with respect to the standard topologies.

The only non-standard typing rules are the *context-restriction rule* on the third line of fig. 3.2a, and the typing rules for the sampler operations, which should be straightforward given their descriptions above. The purpose of the context-restriction rule is, in a nutshell, to be able to pass the result of a computation of type $\mathtt{T}$ which is continuous w.r.t. a topology τ on the denotation of $\mathtt{T}$, to a computation using a variable of type $\mathtt{T}$ but which is continuous w.r.t. to a finer topology $\tau' \supset \tau$ on the denotation of $\mathtt{T}$. After application of this rule, it is no longer possible to λ-abstract on the individual variables of the context. There are good semantic reasons for this feature, which we discuss in section 3.1.3. For readability and intuition's sake, the rule is written using the syntactic sugar

$$t^{-1}(\mathtt{T}_i) \triangleq {}_{\mathtt{cast}\langle \mathtt{T} \rangle \mathtt{in}_i(x)}\mathtt{T}_t \qquad \text{where } x : \mathtt{T}_i, t : \mathtt{T}; \tag{3.1}$$

for the subtyping rules fig. 3.2b, we use the syntactic sugar

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

Remark 3.1.1 (Recursion)**.** *Our typed lambda calculus, the reader will notice, does not feature recursion. This is because our aim is to restrict attention to samplers which are produced via techniques which are sensible from a probabilistic perspective, so that verification of these probabilistic properties can be accomplished straightforwardly by following the program structure. Allowing unrestricted recursion undermines that goal: it expands massively the class of programs that can be written, but where the vast majority of these programs do not have an interpretation which is sensible from a probabilistic perspective.*

Moreover, recursion is not strictly necessary for our purposes. While the author is not aware of a general definition of the notion of 'recursive sampler', which would be necessary to formalise this argument, many of the sampling techniques which are typically given recursively can be recast in our language by

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

$$\frac{\Gamma \vdash t : \mathtt{T}_j}{\Gamma \vdash \mathtt{in}_j\,(t) : \sum_{i \in n} \mathtt{T}_i}\ j \in n \qquad \frac{\Delta \vdash t : \mathtt{S}}{\Gamma \vdash \mathtt{cast}\langle \mathtt{T} \rangle t : \mathtt{T}}\ \mathtt{S} \triangleleft \mathtt{T}, \Gamma \triangleleft \Delta$$

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

$$\frac{\Gamma \vdash t : \sum_{i \in n} \mathtt{T}_i \quad \Gamma, x_i : \mathtt{T}_i \vdash s_i : \mathtt{T}}{\Gamma \vdash \mathtt{case}\ t\ \mathtt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i \in n} : \mathtt{T}}$$

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

$$\frac{\Gamma, x : \mathtt{S} \vdash t : \mathtt{T}}{\Gamma \vdash \lambda x : \mathtt{S}\,.\,t : \mathtt{S} \to \mathtt{T}} \qquad \frac{\Gamma \vdash s : \mathtt{S} \quad \Gamma \vdash t : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash t(s) : \mathtt{T}} \qquad \frac{\Gamma, x : \mathtt{S} \vdash t : \mathtt{T} \quad \Gamma \vdash s : \mathtt{S}}{\Gamma \vdash \mathtt{let}\ x = s\ \mathtt{in}\ t : \mathtt{T}}$$

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

$$\frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \quad \Gamma \vdash t : \Sigma\,\mathtt{T}}{\Gamma \vdash s \otimes t : \Sigma\,(\mathtt{S} \times \mathtt{T})} \qquad \frac{\Gamma \vdash s : \mathtt{T} \to \mathtt{T} \quad \Gamma \vdash t : \mathtt{T}}{\Gamma \vdash \mathtt{prng}(s, t) : \Sigma\,\mathtt{T}}$$

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

(a) Typing rules

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

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

(b) Subtyping rules

Figure 3.2: Typing and subtyping rules

using its native operations in a non-recursive manner. For example, rejection sampling, while it is most commonly thought of as a recursive procedure, can alternatively be implemented using the operation `reweight`*, as we will show in section 3.3.2. More complex recursive samplers which feature loops with state can be incorporated as well, but as these samplers are less amenable to verification using our methods, we will not pursue this direction (though see section 5.2 for a sketch of how this can be done).*

Finally, the categorical semantics of a true typed, probabilistic, higher-order lambda calculus with recursion are a more recent area of investigation [13]. As a result, we consider the inclusion of recursively-specified samplers,

along with corresponding verification techniques for recursively-specified samplers, to be further work; see section 5.2.

3.1.2 Operational semantics

In practice, in order to evaluate a program containing a sampler, one must specify a finite number of samples $N \in \mathbb{N}$ which are to be produced. Our (big-step) operational semantics correspondingly takes the form of a reduction relation $(t, N) \to v$, where the left side consists of a well-typed closed term $t \in \text{Expr}$ and a number of samples N, and the right side is a *value* $v \in \text{Value}$, i.e. a term generated by the grammar fig. 3.1c.

For the more common language constructs like `let` and function evaluation, the rules of this big-step operational semantics, given in fig. 3.3a, are mostly standard with the additional input N, the number of samples which will be taken. The big-step operational semantics of sampler operations, which do make use of the inputted number of samples N, are given in fig. 3.3b.

For notational simplicity, these operations make use of lists (a, b, c, d), which are in fact interpreted within our language as nested pairs $(a, (b, (c, d)))$. In order to keep the rules readable, we also introduce the shorthand $(t, N) \to ((v_1, w_1), \ldots, (v_N, w_N))$ to denote the N reductions

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

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 $N \in \mathbb{N}$, programs of sampler type can only reduce to weighted lists of length N.

$$\frac{}{(v,N)\to v}\ v \text{ a value} \qquad \frac{(t,N)\to v}{(\mathtt{cast}\langle \mathtt{T}\rangle t,N)\to v} \qquad \frac{(t,N)\to v}{(f(t),N)\to f(v)}\ \text{Func} \ni f:\mathtt{T}\to\mathtt{G}$$

$$\frac{((\lambda x:T.t)(s),N)\to v}{(\mathtt{let}\ x=s\ \mathtt{in}\ t,N)\to v} \qquad \frac{(t[x:=s],N)\to v}{((\lambda x:\mathtt{T}.t)(s),N)\to v} \qquad \frac{(t,N)\to v}{(\mathtt{in}_i\,(t)\,,N)\to (i,v)}$$

$$\frac{(t,N)\to (j,v_j)\quad (s_j[x_j:=v_j],N)\to v}{(\mathtt{case}\ t\ \mathtt{of}\ \{(i,x_i)\Rightarrow s_i\}_{i\in n}),N)\to v}\ j\in n$$

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

(a) Big-step operational semantics of standard operations

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

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

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

$$\frac{(t,N)\to ((v_1,w_1),\ldots,(v_N,w_N))}{(\mathtt{hd}(t),N)\to v_1}$$

$$\frac{(t,N)\to ((v_1,w_1),\ldots,(v_N,w_N))}{(\mathtt{wt}(t),N)\to w_1}$$

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

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

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

(b) Big-step operational semantics of sampler operations

Figure 3.3: Big-step operational semantics

Proposition 3.1.2. *If* $\vdash s:\Sigma\,\mathtt{S}$ *is a closed sampler, then for any* $N\in\mathbb{N}$*, if* $(s,N)\to v$*, then* v *has the form* $((v_1,w_1),\ldots,(v_N,w_N))$*, where* v_n *are values and* $w_n\in\mathbb{R}_{\geq 0}$ *are weights. If* $\mathtt{S}$ *is not a sampler type, then* $v_n:\mathtt{S}$*; more generally, each* v_n *might be a weighted list itself.*

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

Proposition 3.1.3. *If* $\vdash s:\Sigma^k\mathtt{S}$ *is a closed* k*-order sampler where* $\mathtt{S}$ *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 $\mathtt{S}$*. For example, for $k = 0$, $v : \mathtt{S}$ is simply a value of type* $\mathtt{S}$*; for $k = 1$, $v = ((v_1, w_1), \ldots, (v_N, w_N))$ is a weighted list of values $v_n : \mathtt{S}$ and $w_n \geq 0$; 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 $v_n^m : \mathtt{S}$, and so on.*

Proof of Proposition 3.1.3.

Base case. As values v cannot have sampler type, the only possibility for a derivation $(v, N) \to v$ where $v : \Sigma^k\mathtt{T}$ for some type $\mathtt{T}$ 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 $\vdash t : \Sigma^k\mathtt{T}$ is a k-order sampler for some $k \in \{0, 1, 2, \ldots\}$, and $\mathtt{T}$ is by hypothesis not a sampler 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 induction hypothesis that each input $s_i : \mathtt{G}_i$ reduces to a value $v_i : \mathtt{G}_i$, where $\mathtt{G}_n$ are ground types, we immediately obtain that $(f(s_1, \ldots, s_n), N) \to v$ evaluates to a value v of ground type $\mathtt{G}$, giving our result.

2. **Case.** Assuming that $t = \mathtt{case}\ (c, t')\ \mathtt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i \in n}$, we must have $\vdash s_i : \Sigma^k\mathtt{T}$. Taking the induction hypothesis that $(s_i[x_i := t'], N) \to v$ 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 $t = (\lambda x : \Sigma^{k'}\mathtt{S}.t')(s)$ to be an instance of function application, where the function in question inputs a k'-sampler and outputs a k-sampler, where $\mathtt{S}$ does not contain any sampler types itself; we must have $\vdash s : \Sigma^{k'}\mathtt{S}$ 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 $(t'[x := s], N) \to v$; taking the induction hypothesis

that $t'[x := s]$ evaluates to a k-nested list of values of type $\mathtt{S}$, our desired result follows.

4. **`let`-binding.** Trivially follows from function application, as $(\mathtt{let}\ x = s\ \mathtt{in}\ t', N) \to v$ iff $((\lambda x : S.t')(s), N) \to v$.

5. **Product.** If $t = (s, s')$, the result is trivial as $\vdash t : \Sigma^k\mathtt{T}$ implies $k = 0$ and so $\mathtt{T} = \mathtt{S} \times \mathtt{S}'$ where $\vdash s : \mathtt{S}, \vdash s' : \mathtt{S}'$ are each not sampler types; therefore, (s, s') is clearly a value of type $\mathtt{T}$ (i.e., a 0-nested weighted list).

6. **Projections.** If $t = \mathtt{fst}((s, s'))$, then $\vdash s : \Sigma^k\mathtt{T}$, and so the induction hypothesis $(s, N) \to v$ immediately implies our result; the same argument applies to $t = \mathtt{snd}((s, s'))$.

7. **Head.** If $t = \mathtt{hd}(s)$, then $\vdash s : \Sigma^{k+1}\mathtt{T}$. Taking the induction hypothesis that $(s, N) \to v$ implies that v is a $(k+1)$-nested weighted list of values of type $\mathtt{T}$, we need only note that the first element of this list is itself a k-nested weighted list of values of type $\mathtt{T}$.

8. **Weight.** If $t = \mathtt{wt}(s)$, our result is trivially true, as the output of $\mathtt{wt}(s)$ can only be a nonnegative real number $\vdash t : R^+$.

9. **Tail.** If $t = \mathtt{tl}(s)$, then by our induction hypothesis, $(s, N+1) \to ((v_1, w_1), \ldots, (v_{N+1}, w_{N+1}))$ where each v_n is a $(k-1)$-nested weighted list of elements of type $\mathtt{T}$. 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 $\mathtt{T}$.

10. **Thin.** If $t = \mathtt{thin}(i, s)$, then $\vdash n : N$ and $\vdash s : \Sigma^k\mathtt{T}$, and by our induction hypothesis, $(s, Ni) \to ((v_1, w_1), \ldots, (v_{Ni+1}, w_{Ni+1}))$ where each v_n is a $(k-1)$-nested weighted list of elements of type $\mathtt{T}$. It immediately follows that $(\mathtt{thin}(i, s), N)$ evaluates to a weighted list of N elements whose elements are each $(k-1)$-nested lists of type $\mathtt{T}$.

11. **Map.** If $t = \texttt{map}(s, t')$ and $(t, N) \to v$, the operational semantics of `map` requires that

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

for each $n \in \{1, \ldots, N\}$. As t' is a subterm of t, our induction hypothesis implies that if $\vdash t' : \Sigma^{k'}\texttt{S}$ 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 $\vdash s : \Sigma^{k'-1}\texttt{S} \to \Sigma^{k-1}\texttt{T}$. We have already shown that if this is the case, then $\texttt{tl}^{n-1}(t')$ evaluates to a k'-nested weighted list of values of type `S`, and then that $\texttt{hd}(\texttt{tl}^{n-1}(t'))$ evaluates to a $(k'-1)$-nested weighted list of values of type `S` of length N, and then that $s(\texttt{hd}(\texttt{tl}^{n-1}(t')))$ evaluates to a $(k-1)$-nested weighted list of values of type `T` of length N. Our result follows by observing that if $((s(\texttt{hd}(\texttt{tl}^{n-1}(t'))), \texttt{wt}(\texttt{tl}^{n-1}(t'))), N) \to (v_n, w_n)$ 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 $\vdash t : \Sigma^{k}\texttt{T}$ where `T` contains no instances of Σ, then it must be that $k \geq 1$, that $\vdash s : \Sigma^{k}\texttt{S}$, and that $\vdash s' : \Sigma^{k}\texttt{S}'$. Our induction hypothesis states that $(s, N) \to ((v_1, w_1), \ldots, (v_N, w_N))$ where each v_1 is a $(k-1)$-nested weighted list of values of type `S`, 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 $t = \texttt{prng}(s, t')$; in order for this expression to be well-typed, we must have $k \geq 1$, $\vdash t' : \Sigma^{k-1}\texttt{T}$, and $\vdash s : \Sigma^{k-1}\texttt{T} \to \Sigma^{k-1}\texttt{T}$. 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`.

□

The self-product operation

Having clarified the meaning of the product and of the `thin` operation, we are now in a position to formally define the 'self-product' of a sampler, which enables us to use multiple adjacent samples from the same sampler. To motivate it, consider a sampler $t : \Sigma\,\mathtt{T}$ which evaluates as $(t, 2N) \to (x_1, \ldots, x_{2N})$, where for notational clarity we have omitted the weights. Applying the above operational semantics, the lagged sampler $\mathtt{thin}(2, t \otimes \mathtt{tl}(t)) : \Sigma\,(\mathtt{T} \times \mathtt{T})$ evaluates to

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

We call this sampler the 'self-product' of t, and denote it t^2. Note that, by contrast, the sampler $t \otimes t$ will produce pairs of perfectly correlated samples: the operational semantics gives $(t \otimes t, N) \to ((x_1, x_1), \ldots (x_N, x_N))$.

More generally, for any $k \in \mathbb{N}$, we define the k-fold self-product of a sampler as

$$t^k \triangleq \mathtt{thin}(k, t \otimes \mathtt{tl}(t) \otimes \ldots \otimes \mathtt{tl}^{k-1}(t)). \tag{3.2}$$

Sampling from t^k is intended to allow the sampling of k-tuples of independent deviates generated by the sampler k. Ultimately, it is primarily to define this self-product operation that the sampler operation `thin` is included.

3.1.3 Denotational semantics

Denotational universe. We will see in section 3.3 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 in order to interpret the lambda-abstraction operation of our language. A standard solution is to consider the category of *compactly generated topological spaces* [36, 37, 38] (henceforth *CG-spaces*).

Definition 3.1.4 (CG-space, [36, §1])**.** *A topological space X is compactly generated if it is Hausdorff and has the property that $C \subseteq X$ is closed iff $C \cap K$ is closed in K for every compact K in X.*

We need not worry about the theory of these spaces, but the following facts are essential in what follows.

Proposition 3.1.5 ([36, 38])**.**

1. *The category* **CG** *of CG-spaces and continuous functions is Cartesian closed.*
2. *The category* **CG** *is complete and cocomplete.*
3. *Every metrisable 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 [36, §4]. However, in most practical instances the usual product topology is already compactly generated – for example, any countable product of metrisable spaces is metrisable, and thus compactly generated by Proposition 3.1.5. 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 [36, §5].

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

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

together with the subspace topology inherited from $\mathbb{R} \times \mathbb{R}$. Since all these spaces are metrisable, our ground types are interpreted in **CG** by Proposition 3.1.5.

Products (including the unit type) and function types are interpreted in the obvious way using the Cartesian closed structure of **CG**. Coproduct types are interpreted by coproducts in **CG**, and given two terms $s, t : \mathtt{T}$ interpreted as **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 **CG** by Proposition 3.1.5.

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

Theorem 3.1.6 (Adámek)**.** *Let $\mathscr{C}$ be a category with terminal object 1, and $F : \mathscr{C} \to \mathscr{C}$ be a functor. If $\mathscr{C}$ has and F preserves ω^{op}-indexed limits, then the limit νF of $1 \xleftarrow{!} F1 \xleftarrow{F!} FF1 \xleftarrow{FF!} \ldots$ is the terminal coalgebra of F.*

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

$$[\![\Sigma\,\mathtt{T}]\!] \triangleq \nu([\![\mathtt{T}]\!] \times \mathbb{R}^+ \times \mathrm{Id}). \tag{3.3}$$

Since products are limits, and limits commute with limits, it is clear that the functor $[\![\mathtt{T}]\!] \times \mathbb{R}^+ \times \mathrm{Id}$ preserves limits, and in particular ω^{op}-indexed ones. Adámek's theorem thus guarantees the existence of an object satisfying eq. (3.3). More concretely, since the termimal object 1 is trivially metrisable, and since $\mathbb{R}^+$ is metrisable, each object in the terminal sequence will be metrisable provided $[\![\mathtt{T}]\!]$ is, and thus $\prod_n([\![\mathtt{T}]\!] \times \mathbb{R}^+)^n$ will be metrisable whenever $[\![\mathtt{T}]\!]$ is, and will therefore be equipped with the usual product topology. The limit defining eq. (3.3) is a closed subspace of this product, which means that the limit in **CG** defining $[\![\Sigma\,\mathtt{T}]\!]$ is the same as in **Top** when $[\![\mathtt{T}]\!]$ is metrisable (for example, if $\mathtt{T}$ is a ground type or a product of ground types). However, by defining $[\![\Sigma\,\mathtt{T}]\!]$ coinductively rather than simply as $([\![\mathtt{T}]\!] \times \mathbb{R}^+)^\omega$, we obtain a terminal coalgebra structure on $[\![\Sigma\,\mathtt{T}]\!]$, and therefore the ability to define sampler operations coinductively.

Semantics of the subtyping relation. Our language contains the predicates $f \in \{\leq, <, \geq, >, =, \neq\}$ (essential for, among other applications, rejection sampling theorem 3.3.16) – and yet is meant to be interpreted in a universe of topological spaces and continuous maps. These predicates are of course not continuous maps $\mathbb{R} \times \mathbb{R} \to 2$ for the usual topology on $\mathbb{R} \times \mathbb{R}$. However, for each such predicate f, the sets $[\![f^{-1}(0)]\!]$ and $[\![f^{-1}(1)]\!]$ are *locally closed sets*, that is to say the intersection of an open set and a closed set (for the usual topology on $\mathbb{R} \times \mathbb{R}$), and therefore CG-spaces by Proposition 3.1.5.

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

The topology on $[\![f^{-1}(0) + f^{-1}(1)]\!]$ is finer than the usual topology on $\mathbb{R} \times \mathbb{R}$, which means that the identity map $\mathrm{Id} : [\![f^{-1}(0) + f^{-1}(1)]\!] \to \mathbb{R} \times \mathbb{R}$ is continuous. This is the semantic basis for the axiom in fig. 3.2b. From the other rules it is easy to see by induction that the subtyping relation is only

defined on spaces *sharing the same carrier set* and, semantically, coarsens the space's topology. In other words, if $\mathtt{S} \lhd \mathtt{T}$, then $[\![\mathtt{S}]\!]$ and $[\![\mathtt{T}]\!]$ share the same carrier, and the corresponding identity map $\mathrm{Id} : [\![\mathtt{S}]\!] \to [\![\mathtt{T}]\!]$ is continuous.

Example 3.1.7. *Let* $\mathtt{p} \triangleq \mathtt{if}\ x = 0\ \mathtt{then}\ 1\ \mathtt{else}\ -1$*; we will first show how the context-restriction rule allows us to type-check this program. For readability's sake, let* $\mathtt{Eq} \triangleq =^{-1}(1)$ *and* $\mathtt{Neq} \triangleq =^{-1}(0)$*. We now derive, using* $= : \mathtt{Neq} + \mathtt{Eq} \to \mathtt{R}$*,*

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

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

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

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

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

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

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

We can now describe the semantics of the context-restriction rule. From the premise, our semantics for the subtyping relation, 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. (3.1) we interpret each 'inverse image type' $t^{-1}(\mathtt{T}_i)$ as the pullback (inverse image) of $[\![t]\!]$ along the inclusion $[\![\mathtt{T}]\!]_i \hookrightarrow \coprod_{i\in m} [\![\mathtt{T}_i]\!]$ which is, as the notation implies, simply given by $[\![t]\!]^{-1}([\![\mathtt{T}_i]\!])$. Since $\coprod_{i\in m} [\![\mathtt{T}_i]\!]$ and $[\![\mathtt{T}]\!]$ share the same carrier, it is clear that this defines a partition of $[\![\Gamma]\!]$, and we can thus retype t as a continuous map $\coprod_{i\in m} [\![t^{-1}(\mathtt{T}_i)]\!] \to \coprod_{i\in m} [\![\mathtt{T}_i]\!]$, interpreting the rule.

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

Example 3.1.8. *Consider the simple program $x < y$, with derivation*

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

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

$$[\![<]\!] : [\![<^{-1}(0)]\!] + [\![<^{-1}(1)]\!] \to 2$$

where $[\![<^{-1}(0)]\!] = \{(x,y) \mid x \geq y\}$ and $[\![<^{-1}(1)]\!] = \{(x,y) \mid x < y\}$, each equipped with the subspace topology. While it has the same carrier $\mathbb{R} \times \mathbb{R}$, the domain of $[\![<]\!]$ is no longer a product of topological spaces – it is instead a coproduct of topological spaces. This means that it is no longer possible to λ-abstract over x or y using the Cartesian closed structure of **CG**.

In order to be able to λ-abstract the map $<$ in this way, 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 all of the open sets

$[x_0, \infty)$ *to the topology of* $\mathbb{R}$*, 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 [39].*

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

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

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

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

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

Proposition 3.1.9. *Let* $F : \mathbf{CG} \to \mathbf{CG}$ *satisfy the condition of theorem 3.1.6 as well as the condition that* $\mathrm{int}(\nu F) \neq \emptyset$ *in* $\prod_i F^i 1$*, and let* $\mathrm{beh}_X : [X, FX] \to [X, \nu F]$ *be the (behaviour) map associating to any* F*-coalgebra structure on* X *the unique coalgebra morphism into the terminal coalgebra. The map* beh_X *is continuous, i.e. is a* **CG***-morphism.*

Proof of Proposition 3.1.9.

Let $f_n \to f$ be a convergent sequence a coalgebra maps in $[X, FX]$; we need to show that $\mathrm{beh}_X(f_n) \to \mathrm{beh}_X(f)$ 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

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

for some fixed compact set $K \subseteq X$ and open set $U \subseteq \nu F$. Moreover, by construction of νF (see theorem 3.1.6), 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 $\mathrm{int}(\nu F) \neq \emptyset$ in $\prod_i F^i 1$, it contains such an open set, and we can thus simply start with an open neighbourhood of $\mathrm{beh}_X f$ 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 $n > N$ $\mathrm{beh}_X(f_n) \in (K, \prod_i V_i)$.

By the construction of theorem 3.1.6 we have that

$$\mathrm{beh}_X(f)(x) = (!_X(x), F!_X(f(x)), F^2!_X(Ff(f(x))), \ldots)$$

where $!_X : X \to 1$ is the unique morphism to the terminal object. For each of the finitely many non-trivial open subsets $V_{i_k} \subset F^{i_k} 1, 1 \leq k \leq M$, because $f_n \to f$ and composition with continuous functions is a continuous operation on internal hom sets in **CG** ([36, 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 $N = \max_{1 \leq k \leq M} N_k$, 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 $n > N$, $\mathrm{beh}_X(f_n) \in (K, \prod_i V_i)$, which concludes the proof. □

Using unfold and beh we define the denotational semantics of all the sampler operations in fig. 3.4. These definitions are precisely the infinite (coin-

ductive) versions of the finitary transformations defined in the operational semantics of fig. 3.3. All the maps involved in these definitions are continuous; this follows from Proposition 3.1.9 and the fact that evaluation and function composition are continuous operations on the internal homsets of **CG** ([36, 5.2,5.9]).

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

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

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

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

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

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

Figure 3.4: Denotational semantics of sampler operations

Adequacy

Our language prominently features an interesting asymmetry: its denotational semantics is written in terms of the coinductive sampler type $[\![\Sigma\,\mathtt{T}]\!]$, while its operational semantics is written in terms of finitary operations on finite sequences of samples. More specifically, the operational semantics is given in terms of reductions to the *values* defined by fig. 3.1c, and values with no free variables cannot be of sampler type. To establish a connection between the two, we begin by defining a generic way to convert terms of arbitrary type (including sampler type) into values, which are not samplers, mirroring the rules of the operational semantics. Given a type $\mathtt{T}$ and an integer N we

inductively define its associated value type $\mathrm{val}^N(\mathtt{T}) \in \mathrm{Value}$ by

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

where $\mathtt{G} \in \mathrm{Ground}$. Since we're only interested in closed samplers here, and pullback types can only occur in a context, we need not define val^N on pullback types.

We now define the *generalised projection* maps $p^N_{\mathtt{T}} : [\![\mathtt{T}]\!] \to [\![\mathrm{val}^N(\mathtt{T})]\!]$ recursively via

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

The reader will have noticed that we have defined $p^N_{\mathtt{S}\to\mathtt{T}}$ trivially. The reason is that, as a quick examination of the rules of fig. 3.3 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 a function evaluation, or are *values*, i.e. terms trivially reducing to themselves.

Theorem 3.1.10. *For any program* $\vdash t : \mathtt{T}$*, we have*

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

Proof of Theorem 3.1.10.

$\Leftarrow$) 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 fig. 3.3 assumes that $t = v$ is a value. It is easy to check that if $t : \mathtt{T}$ is a value, then $p^N_{\mathtt{T}} = \mathrm{id}_{[\![\mathtt{T}]\!]}$ and thus $p^N_{\mathtt{T}}([\![t]\!]) = [\![t]\!] = [\![v]\!]$ tautologically.

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

(i) **Built-in function.** $t = f(s_1, \ldots, s_n) : \mathtt{G}$ for some $s_i : \mathtt{G}_i, 1 \leq i \leq n$. This means that the prior rules of the derivation were $(s_i, N) \to v_i$ for each i, and by our inductive hypothesis, $p^N_{\mathtt{G}}(\llbracket s_i \rrbracket) = \llbracket v_i \rrbracket$. Since for a ground type $\mathtt{G}$ we have $p^N_{\mathtt{G}} = \mathrm{id}_{\llbracket \mathtt{G} \rrbracket}$, we immediately get

$$\begin{aligned} p^N_{\mathtt{G}}(\llbracket f(s_1, \ldots, s_n) \rrbracket) &\triangleq \llbracket f \rrbracket \left(p^N_{\mathtt{G}_1}(\llbracket s_1 \rrbracket), \ldots, p^N_{\mathtt{G}_i}(\llbracket s_n \rrbracket)\right) \\ &= \llbracket f \rrbracket \left(\llbracket v_1 \rrbracket, \ldots, \llbracket v_n \rrbracket\right) && \text{induction hypothesis} \end{aligned}$$

(ii) **Case.** $t = \mathtt{case}\ (c, t')\ \mathtt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i \in n}$. If c chooses the branch $j \in n$, then $p^N_{\mathtt{T}}(\llbracket s_j \rrbracket (\llbracket t' \rrbracket)) = \llbracket v \rrbracket$ by our inductive hypothesis, and

$$\begin{aligned} p^N_{\mathtt{T}}\left(\llbracket \mathtt{case}\ (c, t')\ \mathtt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i \in n} \rrbracket\right) &\triangleq p^N_{\mathtt{T}}(\llbracket s_j \rrbracket (\llbracket t' \rrbracket)) \\ &= \llbracket v \rrbracket && \text{induction hypothesis} \end{aligned}$$

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

$$\begin{aligned} p^N_{\mathtt{T}}\left(\llbracket (\lambda x : \mathtt{T}.\ t')(s) \rrbracket\right) &\triangleq p^N_{\mathtt{T}} \circ \mathrm{ev}_{\llbracket \mathtt{S} \rrbracket, \llbracket \mathtt{T} \rrbracket}\left(\llbracket \lambda x : \mathtt{T}.\ t' \rrbracket \times \llbracket s \rrbracket\right) \\ &\triangleq p^N_{\mathtt{T}} \circ \mathrm{ev}_{\llbracket \mathtt{S} \rrbracket, \llbracket \mathtt{T} \rrbracket}\left(\widehat{\llbracket t' \rrbracket} \times \llbracket s \rrbracket\right) && \text{Currying } \llbracket t' \rrbracket \\ &= p^N_{\mathtt{T}} \circ \mathrm{ev}_{\llbracket \mathtt{S} \rrbracket, \llbracket \mathtt{T} \rrbracket}\left(\llbracket t' \rrbracket \times \llbracket s \rrbracket\right) && \llbracket t' \rrbracket \text{ has only one variable} \\ &= p^N_{\mathtt{T}}\left(\llbracket t' \rrbracket (\llbracket s \rrbracket)\right) \\ &= \llbracket v \rrbracket && \text{induction hypothesis} \end{aligned}$$

(iv) **`let`-binding.** $t = \mathtt{let}\ x = s\ \mathtt{in}\ t' : \mathtt{T}$ for some $s : \mathtt{S}$ and $t' : \mathtt{T}$.

$$\begin{aligned} p^N_{\mathtt{T}}\left(\llbracket \mathtt{let}\ x = s\ \mathtt{in}\ t \rrbracket\right) &\triangleq p^N_{\mathtt{T}}(\llbracket t' \rrbracket (\llbracket s \rrbracket)) \\ &= p^N_{\mathtt{T}}\left(\llbracket \lambda x.\ t' \rrbracket (\llbracket s \rrbracket)\right) \\ &= \llbracket v \rrbracket && \text{induction hypothesis} \end{aligned}$$

(v) **Product.** $t = (s, s')$ for some $s : \mathtt{S}, s' : \mathtt{S}'$.

$$\begin{aligned} p^N_{\mathtt{S}\times\mathtt{S}'}([\![(s,s')]\!]) &\triangleq p^N_{\mathtt{S}\times\mathtt{S}'}\left(\langle [\![s]\!], [\![s']\!]\rangle\right) \\ &= \langle p^N_{\mathtt{S}}([\![s]\!]), p^N_{\mathtt{S}'}([\![s']\!])\rangle \qquad \text{inductive definition of } p^N_{\mathtt{T}} \\ &= \langle [\![v_1]\!], [\![v_2]\!]\rangle \qquad\qquad \text{induction hypothesis} \\ &= [\![(v_1, v_2)]\!] \end{aligned}$$

(vi) **Projections.** $t = \mathtt{fst}(s, s')$ for some $s : \mathtt{S}, s' : \mathtt{S}'$.

$$\begin{aligned} p^N_{\mathtt{S}}\left([\![\mathtt{fst}(s,s')]\!]\right) &\triangleq p^N_{\mathtt{S}}\left(\pi_1\langle [\![s]\!], [\![s']\!]\rangle\right) \\ &= p^N_{\mathtt{S}}([\![s]\!]) \\ &= [\![v_1]\!] \qquad\qquad \text{induction hypothesis} \end{aligned}$$

and similarly for `snd`.

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

$$\begin{aligned} h &\triangleq \pi_1 \circ \mathrm{unfold}_{\mathtt{S}}, && \text{i.e. } h(t) \text{ is the first sample of } t \\ w &\triangleq \pi_2 \circ \mathrm{unfold}_{\mathtt{S}}, && \text{i.e. } w(t) \text{ is the weight of the first sample of } t \\ f &\triangleq \pi_3 \circ \mathrm{unfold}_{\mathtt{S}}, && \text{i.e. } f(t) \text{ is the tail of } t. \end{aligned}$$

With this we can now derive

$$
\begin{aligned}
&p^N_{\Sigma\,\mathtt{T}}([\![\mathtt{map}(s,t)]\!]) \\
&\triangleq \pi_{1:N} \circ \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^\omega ([\![\mathtt{map}(s,t)]\!]) \\
&\triangleq \pi_{1:N} \circ \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^\omega (b(t)) \\
&\overset{(1)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ \pi_{1:N}(b(t)) \\
&\overset{(2)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ F\pi_{1:N-1} \circ \gamma \circ (b(t)) \\
&\overset{(3)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ F^{N-1}\pi_1 \circ F^{N-2}\gamma \circ \ldots \circ F^0\gamma(b(t)) \\
&\overset{(4)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ F^{N-1}\pi_1 \circ F^{N-1}b \circ F^{N-2}\delta \circ \ldots \circ F^0\delta(t) \\
&\overset{(5)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ F^{N-1}\pi_1 \circ F^{N-1}b \left((s(h(t)), w(t)), \ldots, (s(h(f^{N-1}(t))), w(f^{N-1}(t))), f^{N-1}(t)\right) \\
&\overset{(6)}{=} \left(p^N_{\mathtt{T}\times\mathbb{R}^+}\right)^N \circ \left((s(h(t))), w(t)), \ldots, (s(h(f^{N-1}(t))), w(f^{N-1}(t)))\right) \\
&= \left(p^N_{\mathtt{T}\times\mathbb{R}^+}(s(h(t))), w(t)), \ldots, p^N_{\mathtt{T}\times\mathbb{R}^+}(s(h(f^{N-1}(t))), w(f^{N-1}(t)))\right) \\
&= \left((p^N_{\mathtt{T}}(s(h(t))), w(t)), \ldots, (p^N_{\mathtt{T}}(s(h(f^{N-1}(t)))), w(f^{N-1}(t)))\right) \\
&\overset{(7)}{=} [\![((v_1, w_1), \ldots, (v_N, w_N))]\!]
\end{aligned}
$$

where (1) is the simple observation that $\pi_{1:N} \circ (p^N_{\mathtt{T}})^\omega = (p^N_{\mathtt{T}})^N \circ \pi_{1:N}$, (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 $p^1_{\mathtt{T}}$, 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

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

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

$$
\begin{aligned}
& p^N_{\Sigma\,\mathtt{T}}(\mathtt{reweight}(s,t)) \\
& \triangleq \pi_{1:N} \circ \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^{\omega}(\mathtt{reweight}(s,t)) \\
& \triangleq \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^{N} \circ \pi_{1:N} \circ b(t) \\
& \overset{(1)}{=} \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^{N} \circ F^{N-1}\pi_1 \circ F^{N-1} b\left((h(t), s(h(t))w(t)), \ldots, (h(f^{N-1}(t)), s(f^{N-1}(t))w(f^{N-1}(t))), f^{N-1}(t)\right) \\
& \overset{(2)}{=} \left((p^N_{\mathtt{T}}(h(t)), s(h(t))w(t)), \ldots, (p^N_{\mathtt{T}}(h(f^{N-1}(t))), s(f^{N-1}(t))w(f^{N-1}(t)))\right) \\
& \overset{(3)}{=} ((\llbracket v_1 \rrbracket, \llbracket w_1 \rrbracket), \ldots, (\llbracket v_N \rrbracket, \llbracket w_N \rrbracket))
\end{aligned}
$$

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 $p^N_{\mathtt{T}\times\mathtt{R}^+}$, 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 $\mathtt{prng}(s,t) : \Sigma\,\mathtt{T}$. Using

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

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

$$
\begin{aligned}
p^N_{\Sigma\,\mathtt{T}}(\llbracket \mathtt{prng}(s,t) \rrbracket) &\triangleq \pi_{1:N} \circ (p^N_{\mathtt{T}\times\mathtt{R}^+})^{\omega}(\llbracket \mathtt{prng}(s,t) \rrbracket) \\
&= \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^{N} \circ F^{N-1}\pi_1 \circ F^{N-1} b((t,1), (s(t),1), \ldots, (s^{N-1}(t),1), s^N(t)) \\
&= ((p^N_{\mathtt{T}}(t),1), (p^N_{\mathtt{T}}(s(t)),1), \ldots, (p^N_{\mathtt{T}}(s^{N-1}(t)),1)) \\
&= \llbracket (v_1,1), (v_2,1), \ldots, (v_N,1) \rrbracket
\end{aligned}
$$

(xii) **Head.** Consider the term $\mathtt{hd}(t)$ for some $t : \Sigma\,\mathtt{T}$. Using the same notation

as above,

$$\begin{aligned} p^N_{\mathtt{T}} \circ [\![\mathtt{hd}(t)]\!] &\triangleq p^N_{\mathtt{T}} \circ \pi_1 \circ \gamma([\![t]\!]) \\ &= \pi_1 \circ \pi_1 \circ \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{1:N}([\![t]\!]) \\ &= \pi_1 \circ \pi_1 \, [\![(v_1, w_1), \ldots, (v_N, w_N)]\!] \quad \text{induction hypothesis} \\ &= [\![v_1]\!] \end{aligned}$$

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

$$\begin{aligned} p^N_{\mathtt{T}} \circ [\![\mathtt{wt}(t)]\!] &\triangleq p^N_{\mathtt{T}} \circ \pi_2 \circ \gamma([\![t]\!]) \\ &= \pi_2 \circ \pi_1 \circ \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{1:N}([\![t]\!]) \\ &= \pi_2 \circ \pi_1 \, [\![(v_1, w_1), \ldots, (v_N, w_N)]\!] \quad \text{induction hypothesis} \\ &= [\![w_1]\!] \end{aligned}$$

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

$$\begin{aligned} p^N_{\Sigma\,\mathtt{T}} \circ [\![\mathtt{tl}(t)]\!] &\triangleq \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{1:N}(\pi_3 \circ \gamma([\![t]\!])) \\ &= \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{2:N+1}([\![t]\!]) \\ &= \pi_{2:N+1} \circ \left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^{N+1} \circ \pi_{1:N+1}([\![t]\!]) \\ &= \pi_{2:N+1} \, [\![(v_1, w_1), \ldots, (v_{N+1}, w_{N+1})]\!] \quad \text{induction hypothesis} \\ &= [\![(v_2, w_2), \ldots, (v_{N+1}, w_{N+1})]\!] \end{aligned}$$

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

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 $\vdash (s,t) : \mathtt{S} \times \mathtt{T}$ and that $p^N_{\mathtt{S}\times\mathtt{T}}(\llbracket(s,t)\rrbracket) = \llbracket v\rrbracket$ for some value v. Since the last applied rule had premises $\vdash s : \mathtt{S}$ and $\vdash t : \mathtt{T}$ we have

$$\begin{aligned}\llbracket v\rrbracket &= p^N_{\mathtt{S}\times\mathtt{T}}(\llbracket s \otimes t\rrbracket)\\ &= p^N_{\mathtt{S}} \times p^N_{\mathtt{T}} \langle \llbracket s\rrbracket, \llbracket t\rrbracket\rangle \qquad \text{inductive definition of } p^N_{\mathtt{S}\times\mathtt{T}}\\ &= (p^N_{\mathtt{S}}\ \llbracket s\rrbracket, p^N_{\mathtt{T}}\ \llbracket t\rrbracket)\\ &= (\llbracket v_1\rrbracket, \llbracket v_2\rrbracket)\end{aligned}$$

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 $\vdash \lambda x : \mathtt{S}.\ t : \mathtt{S} \to \mathtt{T}$, then the term $\lambda x : \mathtt{S}.\ t$ is a value, and thus $(\lambda x : \mathtt{S}.\ t, N) \to \lambda x : \mathtt{S}.\ t$ trivially.

(iii) **Head.** Suppose that $\vdash \mathtt{hd}(t) : \mathtt{T}$, and that $p^N_{\mathtt{T}}(\llbracket\mathtt{hd}(t)\rrbracket) = \llbracket v_1\rrbracket$ for some value v_1. Since the last applied rule has the premise $\vdash t : \Sigma\,\mathtt{T}$, and given the semantics of $\mathtt{hd}$, it must be the case that for any $N \geq 1$, $\left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{1:N}(t) = \llbracket((v_1,w_1),\ldots,(v_N,w_N))\rrbracket$ 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 $(\mathtt{hd}(t),N) \to v_1$.

(iv) **Weight.** The proof is the same as that for $\mathtt{hd}$. Suppose that $\vdash \mathtt{wt}(t) : \mathtt{T}$, and that $p^N_{\mathtt{T}}(\llbracket\mathtt{wt}(t)\rrbracket) = \llbracket w_1\rrbracket$ for some weight $w_1 \geq 0$. Since the last applied rule has the premise $\vdash t : \Sigma\,\mathtt{T}$, and given the semantics of $\mathtt{wt}$, it must be the case that for any $N \geq 1$, $\left(p^N_{\mathtt{T}\times\mathtt{R}^+}\right)^N \circ \pi_{1:N}(t) = \llbracket((v_1,w_1),\ldots,(v_N,w_N))\rrbracket$ 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 $(\mathtt{wt}(t),N) \to w_1$.

(v) **Pushforward.** Suppose that $\vdash \texttt{map}(t, s) : \Sigma\,\texttt{T}$ and that $p^N_{\Sigma\,\texttt{T}}([\![\texttt{map}(t, s)]\!]) = [\![((v_1, w_1), \ldots, (v_n, w_n)]\!]$. The premises of the last applied rule must have been $\vdash s : \Sigma\,\texttt{S}$ and $\vdash t : \texttt{S} \to \texttt{T}$, and it follows from the semantics of $\texttt{map}$ that $[\![v_i]\!] = p^N_{\texttt{T}}\,[\![t(\texttt{hd}(\texttt{tl}^{i-1}(s)))]\!])$ and $[\![w_i]\!] = [\![\texttt{wt}(\texttt{tl}^{i-1}(s))]\!]$. It follows from the induction hypothesis that $(t(\texttt{hd}(\texttt{tl}^{i-1}(s)), N) \to v_i$ and $(wt(\texttt{tl}^{i-1}(s)), N) \to w_i$, and thus by the definition of $\to$ we have that $(\texttt{map}(t, s), N) \to ((v_1, w_1), \ldots, (v_n, w_n)$.

□

3.2 Sampler equivalence

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, and methods for proving that equivalence. 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 prove that this simplified form targets the correct distribution. In this section, we introduce a relation $\approx$ on programs which justifies this type of reasoning.

Definition 3.2.1. *We say that two programs* $\Gamma \vdash s : \texttt{T}$ *and* $\Gamma \vdash t : \texttt{T}$ *are* equivalent*, notation* $\Gamma \vdash s \approx t : \texttt{T}$*, if they are related by the smallest congruence relation on well-typed terms containing the rules of fig. 3.5.*[2]

The rules of fig. 3.5 employ a number of shorthand conventions for a more concise presentation. We introduce identity functions $\texttt{id}_\texttt{S} \triangleq \lambda x : \texttt{S}.\ x : \texttt{S} \to \texttt{S}$, constant functions $\texttt{1}_\texttt{S} \triangleq \lambda x : \texttt{S}.\ \texttt{1} : \texttt{S} \to \texttt{R}^+$, function composition $t \circ$

[2]By *congruence relation*, we mean that $\approx$ is an equivalence relation which is also preserved by all operations in our language. For example, if $\Gamma \vdash s \approx t : \Sigma\,\texttt{T}$ holds, then $\Gamma \vdash \texttt{tl}(s) \approx \texttt{tl}(t) : \Sigma\,\texttt{T}$ must hold as well, and the same for all operations in the language; we omit these rules for brevity. We have also omitted as trivial the evaluation of built-in functions on ground types, e.g. rules such as $\Gamma \vdash 3 + 6 \approx 9 : \texttt{R}$, as well as casts on ground types, e.g. $\texttt{cast}\langle \texttt{R} \times \texttt{R}\rangle(3, 6) \approx (3, 6) : \texttt{R} \times \texttt{R}$, where the first $(3, 6)$ is of a ground-truth subtype of $\texttt{R} \times \texttt{R}$, e.g., $\leq^{-1}(0) + \leq^{-1}(1)$.

$s \triangleq \lambda x : \mathtt{S}.\ t(s(x)) : \mathtt{S} \to \mathtt{U}$ where $s : \mathtt{S} \to \mathtt{T}, t : \mathtt{T} \to \mathtt{U}$, compositions $f^0 \triangleq \mathtt{id}_\mathtt{S} : \mathtt{S} \to \mathtt{S}$, $f^n \triangleq f \circ f^{n-1}$ for any $n \in \mathbb{N}$, pointwise products $s \cdot t \triangleq \lambda x : \mathtt{S}, y : \mathtt{T}.\ s(x) * t(y) : \mathtt{S} \times \mathtt{T} \to \mathtt{R}^+$ of real-valued functions $s : \mathtt{S} \to \mathtt{R}^+, t : \mathtt{T} \to \mathtt{R}^+$, and finally Cartesian products $s \times t \triangleq \lambda x : \mathtt{S}, y : \mathtt{T}.\ (s(x), t(y)) : \mathtt{S} \times \mathtt{T} \to \mathtt{S}' \times \mathtt{T}'$ of functions $s : \mathtt{S} \to \mathtt{S}', t : \mathtt{T} \to \mathtt{T}'$.

Theorem 3.2.2. *The rules of fig. 3.5 are sound: if* $\Gamma \vdash s \approx t : \mathtt{T}$*, then* $[\![\Gamma \vdash s : \mathtt{T}]\!] = [\![\Gamma \vdash t : \mathtt{T}]\!]$.

Proof of Theorem 3.2.2.
Standard rules:

1. **β- and η-equivalence.**

$$[\![\Gamma \vdash (\lambda x : \mathtt{S}.t)(s) : \mathtt{T}]\!] = [\![\Gamma \vdash t[x := s] : T]\!]\,,$$
$$[\![\Gamma \vdash \lambda x : \mathtt{S}.t(x) : \mathtt{S} \to \mathtt{T}]\!] = [\![\Gamma \vdash t : \mathtt{S} \to \mathtt{T}]\!]$$

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

2. **`let`-reduction.**

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

   True by definition of the denotational semantics of `let`.

3. **Projections.**

$$[\![\Gamma \vdash \mathtt{fst}((s,t)) : \mathtt{S}]\!] = [\![\Gamma \vdash s : \mathtt{S}]\!]\,,$$
$$[\![\Gamma \vdash \mathtt{snd}((s,t)) : \mathtt{T}]\!] = [\![\Gamma \vdash t : \mathtt{T}]\!]$$

   Immediate from the properties of Cartesian products.

$$\Gamma \vdash (\lambda x : \mathtt{S}.t)(s) \approx t[x := s] : \mathtt{T}$$
$$\Gamma \vdash \lambda x : \mathtt{S}.t(x) \approx t : \mathtt{S} \to \mathtt{T}$$
$$\Gamma \vdash \mathtt{let}\ x = s\ \mathtt{in}\ t \approx (\lambda x : \mathtt{S}.\ t)(s) : \mathtt{T}$$
$$\Gamma \vdash \mathtt{fst}((s,t)) \approx s : \mathtt{S}$$
$$\Gamma \vdash \mathtt{snd}((s,t)) \approx t : \mathtt{T}$$
$$\Gamma \vdash \mathtt{case}\ \mathtt{in}_j\ (t)\ \ \mathtt{of}\ \ \{(i,x_i) \Rightarrow s_i\}_{i\in n} \approx s_j[x_j := t] : \mathtt{T}$$
$$\{\Gamma \vdash op(\mathtt{case}\ t\ \mathtt{of}\ \{(i,x_i) \Rightarrow s_i\}_{i\in n}) \approx \mathtt{case}\ t\ \mathtt{of}\ \{(i,x_i) \Rightarrow op(s_i)\}_{i\in n} : \mathtt{T} \mid op \in \{\mathtt{fst}, \mathtt{snd}, \mathtt{hd}, \mathtt{wt}, \ldots\}\}$$

(a) Equivalence rules for general programming constructs

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

$$\{\Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,t)) \approx \mathtt{thin}(n, \mathtt{tl}^n(t)) : \Sigma\,\mathtt{T} \mid n \in \mathbb{N}\}$$
$$\Gamma \vdash \mathtt{tl}(s \otimes t) \approx \mathtt{tl}(s) \otimes \mathtt{tl}(t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{tl}(\mathtt{reweight}(f,t)) \approx \mathtt{reweight}(f, \mathtt{tl}(t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{tl}(\mathtt{map}(f,t)) \approx \mathtt{map}(f, \mathtt{tl}(t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{tl}(\mathtt{prng}(f,t)) \approx \mathtt{prng}(f, f(t)) : \Sigma\,\mathtt{T}$$

(b) Equivalence rules for $\mathtt{hd}, \mathtt{wt}, \mathtt{tl}$

$$\Gamma \vdash \mathtt{thin}(n, s \otimes t) \approx \mathtt{thin}(n,s) \otimes \mathtt{thin}(n,t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash s \otimes \mathtt{reweight}(g,t) \approx \mathtt{reweight}(1_{\mathtt{S}} \cdot g, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{reweight}(f,s) \otimes t \approx \mathtt{reweight}(f \cdot 1_{\mathtt{T}}, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash s \otimes \mathtt{map}(f,t) \approx \mathtt{map}(\mathtt{id}_{\mathtt{S}} \times f, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{map}(f,s) \otimes t \approx \mathtt{map}(f \times \mathtt{id}_{\mathtt{T}}, s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$
$$\Gamma \vdash \mathtt{prng}(f,a) \otimes \mathtt{prng}(g,b) \approx \mathtt{prng}(f \times g, (a,b)) : \Sigma\,(\mathtt{S} \times \mathtt{T})$$

(c) Equivalence rules for $\otimes$

$$\Gamma \vdash \mathtt{thin}(n, \mathtt{thin}(m,t)) \approx \mathtt{thin}(n \times m, t) \approx \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{thin}(n, \mathtt{reweight}(f,t)) \approx \mathtt{reweight}(f, \mathtt{thin}(n,t)) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{thin}(n, \mathtt{map}(f,t)) \approx \mathtt{map}(f, \mathtt{thin}(n,t)) : \Sigma\,\mathtt{T}$$
$$\{\Gamma \vdash \mathtt{thin}(n, \mathtt{prng}(f,t)) \approx \mathtt{prng}(f^n, t) : \Sigma\,\mathtt{T} \mid n \in \mathbb{N}\}$$

(d) Equivalence rules for $\mathtt{thin}$

$$\Gamma \vdash \mathtt{reweight}(g, \mathtt{reweight}(f,t)) \approx \mathtt{reweight}(f \cdot g, t) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{map}(g, \mathtt{map}(f,t)) \approx \mathtt{map}(g \circ f, t) : \Sigma\,\mathtt{T}$$
$$\Gamma \vdash \mathtt{reweight}(g, \mathtt{map}(f,t)) \approx \mathtt{map}(f, \mathtt{reweight}(g \circ f, t)) : \Sigma\,\mathtt{T}$$

(e) Equivalence rules for $\mathtt{map}, \mathtt{reweight}$

Figure 3.5: Rules for sampler equivalence

4. **Pattern matching.**

$$[\![\Gamma \vdash \texttt{case in}_j\,(t)\ \texttt{of}\ \{(i, x_i) \Rightarrow s_i\}_{i\in n} : \mathtt{T}]\!] = [\![\Gamma \vdash s_j[x_j := t] : \mathtt{T}]\!]$$

Immediate from the denotational semantics of `case` and injections.

Congruence rules: Trivial in the denotational setting: if $[\![\Gamma \vdash s : \mathtt{S}]\!] = [\![\Gamma \vdash s' : \mathtt{S}]\!]$ have identical semantics, then clearly, for any built-in operation $\mathtt{op} : \mathtt{S} \to \mathtt{T}$, $[\![\Gamma \vdash \mathtt{op}(s) : \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{op}(s') : \mathtt{T}]\!]$; the same extends to n-ary operations.

Coinductive definitions: These rules all follow immediately from the coinductive definitions of our sampler operations, and will be used heavily in the proofs that follow.

1. **Map.**

$$[\![\Gamma \vdash \mathtt{hd}(\mathtt{map}(s, t)) : \mathtt{T}]\!] = [\![\Gamma \vdash s(\mathtt{hd}(t)) : \mathtt{T}]\!],$$
$$[\![\Gamma \vdash \mathtt{wt}(\mathtt{map}(s, t)) : \mathtt{R}^+]\!] = [\![\Gamma \vdash s(\mathtt{wt}(t)) : \mathtt{R}^+]\!],$$
$$[\![\Gamma \vdash \mathtt{tl}(\mathtt{map}(s, t)) : \Sigma\,\mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{map}(\mathtt{tl}(t)) : \Sigma\,\mathtt{T}]\!]$$

Immediate from the coinductive definition of `map`.

2. **Product.**

$$[\![\Gamma \vdash (\mathtt{hd}(s), \mathtt{hd}(t)) : \mathtt{S} \times \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{hd}(s \otimes t) : \mathtt{S} \times \mathtt{T}]\!],$$
$$[\![\Gamma \vdash \mathtt{wt}(s) * \mathtt{wt}(t) : \mathtt{R}^+]\!] = [\![\Gamma \vdash \mathtt{wt}(s \otimes t) : \mathtt{R}^+]\!],$$
$$[\![\Gamma \vdash \mathtt{tl}(s) \otimes \mathtt{tl}(t) : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!] = [\![\Gamma \vdash \mathtt{tl}(s \otimes t) : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!]$$

Immediate from the coinductive definition of $\otimes$.

3. **Thinning.**

$$\llbracket \Gamma \vdash \mathtt{hd}(\mathtt{thin}(n,t)) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{hd}(t) : \mathtt{T} \rrbracket,$$
$$\left\llbracket \Gamma \vdash \mathtt{wt}(\mathtt{thin}(n,t)) : \mathtt{R}^+ \right\rrbracket = \left\llbracket \Gamma \vdash \mathtt{wt}(t) : \mathtt{R}^+ \right\rrbracket,$$
$$\forall n \in \mathbb{N}, \llbracket \Gamma \vdash \mathtt{tl}(\mathtt{thin}(n,t)) : \Sigma\, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{thin}(n, \mathtt{tl}^n(t)) : \Sigma\, \mathtt{T} \rrbracket,$$
$$\llbracket \Gamma \vdash \mathtt{thin}(1,t) : \Sigma\, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash t : \Sigma\, \mathtt{T} \rrbracket$$

Immediate from the coinductive definition of `thin`.

4. **Pseudorandom number generators.**

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

Immediate from the coinductive definition of `prng`.

5. **Reweighting.**

$$\llbracket \Gamma \vdash \mathtt{hd}(\mathtt{reweight}(s,t)) : \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{hd}(t) : \mathtt{T} \rrbracket,$$
$$\left\llbracket \Gamma \vdash \mathtt{wt}(\mathtt{reweight}(s,t)) : \mathtt{R}^+ \right\rrbracket = \left\llbracket \Gamma \vdash s(\mathtt{hd}(t)) * \mathtt{wt}(t) : \mathtt{R}^+ \right\rrbracket,$$
$$\llbracket \Gamma \vdash \mathtt{tl}(\mathtt{reweight}(s,t)) : \Sigma\, \mathtt{T} \rrbracket = \llbracket \Gamma \vdash \mathtt{reweight}(s, \mathtt{tl}(t)) : \Sigma\, \mathtt{T} \rrbracket$$

Immediate from the coinductive definition of `reweight`.

Composition rules:

1. **Thinning over thinning.**

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

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

$[\![\Gamma \vdash \texttt{thin}(n * m, t) : \Sigma\,\texttt{T}]\!]\,(\gamma)$ of $[\![\Sigma\,\texttt{T}]\!]$ 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 context γ, a bisimulation $R(\gamma) \subseteq [\![\Sigma\,\texttt{T}]\!] \times [\![\Sigma\,\texttt{T}]\!]$. This is a set of samplers satisfying three properties:

(a) $\forall (s,t) \in R(\gamma), \pi_1(\text{unfold}_{\texttt{T}}(s)) = \pi_1(\text{unfold}_{\texttt{T}}(t))$; that is, the head of s and the head of t are the same

(b) $\forall (s,t) \in R(\gamma), \pi_2(\text{unfold}_{\texttt{T}}(s)) = \pi_2(\text{unfold}_{\texttt{T}}(t))$; that is, the first weight of s and the first weight of t are the same

(c) $\forall (s,t) \in R(\gamma), (\pi_3(\text{unfold}_{\texttt{T}}(s)), \pi_3(\text{unfold}_{\texttt{T}}(t))) \in R(\gamma)$; that is, applying `tl` 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 `tl` 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

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

and

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

This suggests as a bisimulation the following relation:

$$
\begin{aligned}
R(\gamma) = \{(& \\
& [\![(\Gamma \vdash \texttt{thin}(n, \texttt{thin}(m, \texttt{tl}^k(t))) : \Sigma\, \texttt{T}]\!]\,(\gamma), \\
& [\![\Gamma \vdash \texttt{thin}(n * m, \texttt{tl}^k(t)) : \Sigma\, \texttt{T})]\!]\,(\gamma) \\
&) \mid k \in \mathbb{N}\}
\end{aligned}
$$

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

$$
\begin{aligned}
[\![\Gamma \vdash \texttt{hd}(\texttt{thin}(n, \texttt{thin}(m, \texttt{tl}^k(t)))) : \texttt{T}]\!] &= [\![\Gamma \vdash \texttt{hd}(\texttt{thin}(m, \texttt{tl}^k(t))) : \texttt{T}]\!] \\
&= [\![\Gamma \vdash \texttt{hd}(\texttt{tl}^k(t)) : \texttt{T}]\!]
\end{aligned}
$$

and

$$
[\![\Gamma \vdash \texttt{hd}(\texttt{thin}(n * m, \texttt{tl}^k(t))) : \texttt{T}]\!] = [\![\Gamma \vdash \texttt{hd}(\texttt{tl}^k(t)) : \texttt{T}]\!]\,;
$$

The same argument exactly applies for `wt`:

$$
\begin{aligned}
[\![\Gamma \vdash \texttt{wt}(\texttt{thin}(n, \texttt{thin}(m, \texttt{tl}^k(t)))) : \texttt{R}^+]\!] &= [\![\Gamma \vdash \texttt{wt}(\texttt{thin}(m, \texttt{tl}^k(t))) : \texttt{R}^+]\!] \\
&= [\![\Gamma \vdash \texttt{wt}(\texttt{tl}^k(t)) : \texttt{R}^+]\!]
\end{aligned}
$$

and

$$
[\![\Gamma \vdash \texttt{wt}(\texttt{thin}(n * m, \texttt{tl}^k(t))) : \texttt{R}^+]\!] = [\![\Gamma \vdash \texttt{wt}(\texttt{tl}^k(t)) : \texttt{R}^+]\!]\,.
$$

Finally, we must show that applying `tl` 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, and follows from the coinductive definition of `thin` as above:

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

and

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

Therefore, for any γ, applying `tl` 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.**

$$[\![\Gamma \vdash \mathtt{map}(g, \mathtt{map}(f, t)) : \Sigma\, \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{map}(g \circ f, t) : \Sigma\, \mathtt{T}]\!]$$

Applying the coinductive definition of `map`, we can easily see

$$[\![\Gamma \vdash \mathtt{tl}(\mathtt{map}(g, \mathtt{map}(f, t))) : \Sigma\, \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{map}(g, \mathtt{map}(f, \mathtt{tl}(t))) : \Sigma\, \mathtt{T}]\!]$$

and

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

which suggests the bisimulation

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

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.**

$$[\![\Gamma \vdash \texttt{reweight}(g, \texttt{reweight}(f, t)) : \Sigma\, \texttt{T}]\!] = [\![\Gamma \vdash \texttt{reweight}(f \cdot g, t) : \Sigma\, \texttt{T}]\!]$$

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

$$[\![\Gamma \vdash \texttt{tl}(\texttt{reweight}(g, \texttt{reweight}(f, t))) : \Sigma\, \texttt{T}]\!] = [\![\Gamma \vdash \texttt{reweight}(g, \texttt{reweight}(f, \texttt{tl}(t))) : \Sigma\, \texttt{T}]\!],$$
$$[\![\Gamma \vdash \texttt{tl}(\texttt{reweight}(g \cdot f, t)) : \Sigma\, \texttt{T}]\!] = [\![\Gamma \vdash \texttt{reweight}(g \cdot f, \texttt{tl}(t)) : \Sigma\, \texttt{T}]\!].$$

Equivalence then follows from the bisimulation

$$R = \{([\![\Gamma \vdash \texttt{reweight}(g, \texttt{reweight}(f, \texttt{tl}^m(t))) : \Sigma\, \texttt{T}]\!], [\![\Gamma \vdash \texttt{reweight}(g \cdot f, \texttt{tl}^m(t)) : \Sigma\, \texttt{T}]\!]) \mid m \in \mathbb{N}\}$$

which is easily verified, giving our desired equality.

4. **Reweighting over map.**

$$[\![\Gamma \vdash \texttt{reweight}(g, \texttt{map}(f, t)) : \Sigma\, \texttt{T}]\!] = [\![\Gamma \vdash \texttt{map}(f, \texttt{reweight}(g \circ f, t)) : \Sigma\, \texttt{T}]\!]$$

Identical to the previous proof, replacing the inner `map` with `reweight` and making the analogous changes.

5. **Thinning over pseudorandom number generators.**

$$\forall n \in \mathbb{N}, [\![\Gamma \vdash \texttt{thin}(n, \texttt{prng}(s, t)) : \Sigma\, \texttt{T}]\!] = [\![\Gamma \vdash \texttt{prng}(s^n, t) : \Sigma\, \texttt{T}]\!]$$

Applying $\mathtt{tl}$ to both sides of each expression and simplifying using the coinductive definitions of $\mathtt{thin}$ and $\mathtt{prng}$, we obtain

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

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

$$\{([\![\Gamma \vdash \mathtt{thin}(n, \mathtt{map}(s, \mathtt{tl}^m(t))) : \Sigma\, \mathtt{T}]\!], [\![\Gamma \vdash \mathtt{map}(s, \mathtt{thin}(n, \mathtt{tl}^m(t))) : \Sigma\, \mathtt{T})]\!]) \mid m \in \mathbb{N}\}$$

which is easily verified and gives our result.

6. **Thinning over map.**

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

Using the coinductive definitions of $\mathtt{map}$ and $\mathtt{thin}$, we obtain

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

and

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

The desired result follows from

$$\{([\![\Gamma \vdash \mathtt{thin}(n, \mathtt{map}(s, \mathtt{tl}^m(t))) : \Sigma\, \mathtt{T}]\!], [\![\Gamma \vdash \mathtt{map}(s, \mathtt{thin}(n, \mathtt{tl}^m(t))) : \Sigma\, \mathtt{T}]\!]) \mid m \in \mathbb{N}\}$$

which is easily seen to be a valid bisimulation.

Product rules:

1. **Thinning.**

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

   Use the coinductive definition of `thin` to show

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

   and

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

   which suggests

   $$\begin{aligned} R = \{([\![\Gamma \vdash \mathtt{thin}(n, \mathtt{tl}^m(s)) \otimes \mathtt{thin}(n, \mathtt{tl}^m(t)) : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!]\,, \\ [\![\Gamma \vdash \mathtt{thin}(n, \mathtt{tl}^m(s \otimes t)) : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!]) : m \in \mathbb{N}\} \end{aligned}$$

   as a bisimulation.

2. **Map.**

   $$[\![\Gamma \vdash s \otimes \mathtt{map}(g, t') : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!] = [\![\Gamma \vdash \mathtt{map}(\mathrm{id}_{\mathtt{S}} \times g, s \otimes t') : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!]$$

   Applying `tl` to each expression yields

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

   and

   $$[\![\Gamma \vdash \mathtt{tl}(\mathtt{map}(\mathrm{id}_{\mathtt{S}} \times g, s \otimes t')) : \Sigma\,(\mathtt{S} \times \mathtt{T}]\!] = [\![\Gamma \vdash \mathtt{map}(\mathrm{id}_{\mathtt{S}} \times g, \mathtt{tl}(s) \otimes \mathtt{tl}(t')) : \Sigma\,(\mathtt{S} \times \mathtt{T})]\!]\,,$$

suggesting

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

as a bisimulation.

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

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

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

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.

$$[\![\Gamma \vdash \texttt{reweight}(f,t) \otimes s' : \Sigma\,(\texttt{S} \times \texttt{T})]\!] = [\![\Gamma \vdash \texttt{reweight}(f \cdot 1_{\texttt{T}}, s' \otimes t) : \Sigma\,(\texttt{S} \times \texttt{T})]\!]$$

Same proof as previous.

4. **Pseudorandom number generators.**

$$[\![\Gamma \vdash \texttt{prng}(f,a) \otimes \texttt{prng}(g,b) : \Sigma\,(\texttt{S} \times \texttt{T})]\!] = [\![\Gamma \vdash \texttt{prng}(f \times g, (a,b)) : \Sigma\,(\texttt{S} \times \texttt{T})]\!]$$

Using the coinductive definition of `prng`, we quickly obtain

$$[\![\Gamma \vdash \texttt{tl}(\texttt{prng}(f,a) \otimes \texttt{prng}(g,b)) : \Sigma\,(\texttt{S} \times \texttt{T})]\!] = [\![\Gamma \vdash \texttt{prng}(f, f(a)) \otimes \texttt{prng}(g, g(b)) : \Sigma\,(\texttt{S} \times$$

and

$$[\![\Gamma \vdash \texttt{tl}(\texttt{prng}(f \times g, (a,b))) : \Sigma\,(\texttt{S} \times \texttt{T})]\!] = [\![\Gamma \vdash \texttt{prng}(f \times g, (f \times g)(a,b)) : \Sigma\,(\texttt{S} \times \texttt{T})]\!],$$

suggesting the bisimulation

$$\begin{aligned} R = \{([\![\Gamma \vdash \texttt{prng}(f, f^m(a)) \otimes \texttt{prng}(g, g^m(b)) : \Sigma\,(\texttt{S} \times \texttt{T})]\!], \\ [\![\Gamma \vdash \texttt{prng}(f \times g, (f \times g)^m(a,b)) : \Sigma\,(\texttt{S} \times \texttt{T})]\!]) \mid m \in \mathbb{N}\} \end{aligned}$$

which gives our desired result.

□

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 eq. (3.2). It is crucial that our equivalence rules should show that this self-product is well-

defined.

Proposition 3.2.3. *For any* $\Gamma \vdash s : \Sigma\,\mathtt{S}$*,* $m, n \in \mathbb{N}$*, the self-product satisfies* $\Gamma \vdash (s^m)^n \approx s^{mn} : \Sigma\,(\mathtt{S}^{mn})$.

Proof of Proposition 3.2.3.
Expanding eq. (3.2), for any well-typed sampler $\Gamma \vdash s : \Sigma\,\mathtt{S}$, the nested self-product $(s^m)^n$ is defined as

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

Applying the rule $\Gamma \vdash \mathtt{tl}(\mathtt{thin}(m, t)) \approx \mathtt{thin}(m, \mathtt{tl}^m(t)) : \Sigma\,\mathtt{T}$ from fig. 3.5 on the innermost expressions, it follows that this program is equivalent in the context Γ to

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

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

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

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

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

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

The equivalence rules in fig. 3.5 are designed to yield an effective procedure for simplifying samplers to a certain 'normal form' – for samplers which do not feature the operation `prng`. To see this, consider the remaining sampler operations, listed in order of priority from highest to lowest:

1. `hd`, `wt`, `tl`
2. `thin`
3. $\otimes$
4. `reweight`
5. `map`

Verify that, for each valid combination, fig. 3.5 gives an operation for distributing a higher-priority sampler operation over a lower-priority one; for example, fig. 3.5 shows us that we can distribute the higher-priority operation `hd` over the lower-priority operation $\otimes$ using the equivalence rule $\texttt{hd}(s \otimes t) \approx (\texttt{hd}(s), \texttt{hd}(t))$. We consider `hd`, `wt`, `tl` to be the same priority.

The only operations in our language which can output samplers are: the operations given above, `prng` (which we have excluded from consideration), function application, case statements, and casts (which are not relevant here, as our casts only change the topology of each type, not its points). Note that fig. 3.5 also features a rule for pulling case statements outwards through each sampler operation. It follows, then, that by applying the rules in fig. 3.5, every β-reduced closed sampler not including `prng` can be rewritten such that its sampler operations are ordered according to priority (and non-sampler operations, such as `case` statements, occur last).

Definition 3.2.4 (Sampler normal form)**.** *We will say that a sampler-in-context* $\Gamma \vdash t : \Sigma\,\texttt{T}$ *is in* ***sampler normal form*** *if it is* β*-reduced and its sampler operations occur in priority order.*

The reduction of samplers (that don't include `prng`) to normal form is straightforward: simply apply the rules of fig. 3.5 in any valid order until one cannot be applied. We will see in section 3.3.2 that reduction to normal form often makes the verification of a sampling technique immediate; as our primary purpose is verification, we need make no claims regarding *unique* reduction to normal form (to which trivial counterexamples are easy to formulate).

We can also show that the self-product distributes over the operations `map` 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 3.2.5. *For any mapped sampler* $\Gamma \vdash \texttt{map}(f, s) : \Sigma\,\texttt{T}$ *and any* $n \in \mathbb{N}$*, it follows that* $\Gamma \vdash \texttt{map}(f, s)^n \approx \texttt{map}(f \times \ldots \times f, s^n) : \Sigma\,(\texttt{T}^n)$*; for a reweighted sampler* $\Gamma \vdash \texttt{reweight}(f, s) : \Sigma\,\texttt{S}$*, it follows that* $\Gamma \vdash \texttt{reweight}(f, s)^n \approx \texttt{reweight}(f \cdot \ldots \cdot f, s^n) : \Sigma\,(\texttt{S}^n)$.

Proof of Proposition 3.2.5.

- **Map**: Applying the definition of the self-product eq. (3.2), the syntax $\texttt{map}(f, s)^n$ is shorthand for the sampler

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

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

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

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

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

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

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

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

- **Reweight**: This proof proceeds the same as the above, but with `map` replaced with `reweight` and the Cartesian product $\times$ replaced with the pointwise product $\cdot$; nevertheless, we will go through it. Applying the definition of the self-product eq. (3.2), the syntax $\texttt{reweight}(f, s)^n$ is shorthand for

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

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

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

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

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

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

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

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

□

3.3 Verification

The fundamental correctness criterion for a sampler is that it should produce samples which, speaking informally for the moment, behave as if they are distributed according to the desired target distribution. We refer to this relationship between a sampler and a probability distribution as *targeting*, and will give its formal definition shortly. This section provides and justifies a simple 'targeting calculus' to compositionally verify this property.

3.3.1 The empirical transformation

First, we need to formalise what we mean when we say that a sampler $s : \Sigma\,\mathtt{T}$ *targets* a probability distribution on $[\![\mathtt{T}]\!]$. We define this relationship in terms of *weak convergence* and typicality, as defined in Definition 2.5.1 and Definition 2.6.6.

Weak convergence is the natural choice for our setting, because under stronger notions of convergence, such as strong convergence or convergence in total variation, sequences of discrete measures on continuous spaces will typically fail to converge to a continuous measure. One might ask about intermediate notions of convergence, such as that metrised by the Kolmogorov-Smirnov metric or uniform convergence over certain classes of functions; we view this as an interesting extension to our work, but one perhaps less suited to the compositional techniques of program verification.

As in section 2.5, given a topological space X, let us write $\mathcal{P}X$ for the space of probability measures on the Borel sets of 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 $f : X \to \mathbb{R}$, $\lim_{n\to\infty} \int f \; d\mu_n = \int f \; d\mu$.

Note that $\mathcal{P}$ 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. The continuity of the pushforward follows from the definition of the weak topology: if $\mu_n \to \mu$ in $\mathcal{P}X$, and if g is any bounded continuous function on Y, then $g \circ f$ is bounded

continuous on X and

$$\lim_{n\to\infty} \int g \; d(f_* \mu_n) = \lim_{n\to\infty} \int g \circ f \; d\mu_n = \int g \circ f \; d\mu = \int g \; d(f_* \mu).$$

A note of caution: 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 our semantics – its role is purely in verification.

In this section, let $\Sigma : \mathbf{Top} \to \mathbf{Top}$ be the functor defined as in eq. (3.3), i.e. the denotation of $\mathtt{\Sigma}$. (Technically, for CG spaces X, the topology of ΣX might depend on whether this limit is taken in **Top** or **CG**, but this will not matter for our purposes here; we are only interested in the elements of this set, which are X-valued weighted streams.)

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

Definition 3.3.1. *The* empirical measure transformation *is the* $\mathbf{Top}^{\mathrm{obj}}$*-collection of maps*

$$\varepsilon_X : \Sigma X \to \mathcal{P}_\perp X$$

defined as

$$\varepsilon_X(\sigma) = \begin{cases} \lim_{n\to\infty} \hat{\sigma}_n & \textit{if it exists} \\ \perp \in 1 & \textit{else} \end{cases}$$

The empirical measure transformation is not a natural transformation, as the following counterexample shows.

Example 3.3.2. *Let $\sigma \in \Sigma X$ be a diverging unweighted sampler on X, i.e. $\varepsilon_X(\sigma) = \perp$ with unit weights, and let $! : X \to 1$ represent the map to the*

terminal object. Then $(\varepsilon_1 \circ \Sigma!)(\sigma) = \varepsilon_1((\bot, 1), (\bot, 1), \ldots) = \delta_\bot$, *but* $(\mathcal{P}_\bot! \circ \varepsilon_X)(\sigma) = \mathcal{P}_\bot!(\bot) = \bot$.

While the empirical measure transformation is not natural, it is in a certain sense 'locally natural'. If a sampler does correspond to a probability measure via ε, this property is preserved by *continuous* maps.

Proposition 3.3.3. *Let* $\sigma \in \Sigma X$ *be a weighted sampler and* $f : X \to Y$ *be continuous. If* $\hat{\sigma}_n$ *converges to a probability measure* μ, *then* $\widehat{\Sigma f(\sigma)}_n$ *does as well:* $(\varepsilon_Y \circ \Sigma f)(\sigma) = (\mathcal{P}_\bot f \circ \varepsilon_X)(\sigma) = f_*\mu$.

Proof of Proposition 3.3.3.
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

$$\begin{aligned}
\lim_{n\to\infty} \int_X g \; d\widehat{\Sigma f(\sigma)}_n &= \lim_{n\to\infty} \int_X g \circ f \; d\hat{\sigma}_n && \text{by definition} \\
&= \int_X g \circ f \; d\mu && \text{since } \varepsilon_X(\sigma) = \mu \\
&= \int_X g \; df_*\mu && \text{change of variable}
\end{aligned}$$

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

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 3.3.4. *Let* $X = [0, 1]$ *and* $\sigma = ((x_1, w_1), (x_2, w_2), \ldots) \in \Sigma[0, 1]$ *denote any sampler such that* $\varepsilon_{[0,1]}(\sigma)$ *is the Lebesgue measure on* $[0, 1]$. *Now consider the map* $f : [0, 1] \to \{0, 1\}$ *defined by* $f(x) = 1$ *if* $x = x_i$ *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* $\varepsilon_{\{0,1\}}(\Sigma f(\sigma)) = \delta_1$ *since* $\Sigma f(\sigma)$ *is a constant stream of ones. On the other,*

we have $\mathcal{P}_\perp(f)(\varepsilon_{[0,1]}(\sigma)) = f_*\varepsilon_{[0,1]}(\sigma)$*, the pushforward of the Lebesgue measure through* f*. As* f *only takes a countable set of points to 1, the measure of the set* $\{1\}$ *under this pushforward is zero. Therefore, the property Proposition 3.3.3 does not hold for this sampler* σ *and this function* f*.*

Even for functions with finitely many discontinuities, it is impossible to extend the class of functions for which Proposition 3.3.3 holds. However, the semantic framework we adopt allows us to bypass this problem altogether. We illustrate these two points by revisiting Example 3.1.7.

Example 3.3.5. *Consider the sampler* $\mathtt{s} \triangleq \mathtt{prng}(\lambda x : \mathtt{R}\,.\, x/2, 1)$ *and the term* $\mathtt{p} \triangleq \mathtt{if}\ x = 0\ \mathtt{then}\ 1\ \mathtt{else}\ -1$ *of Example 3.1.7. Assume first that* $\mathbb{R}$ *is equipped with its standard topology, i.e. that* $[\![\mathtt{p}]\!]$ *is not continuous at 0. Since* $\mathbb{R}$ *is a metric space we can use the Portmanteau lemma, Lemma 2.5.3, and rephrase weak convergence by limiting ourselves to bounded Lipschitz functions. It is then easy to show that* $\varepsilon([\![\mathtt{s}]\!]) = \delta_0$*: letting* $f : \mathbb{R} \to \mathbb{R}$ *be bounded Lipschitz, we have*

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

Proposition 3.3.3 now fails on $[\![\mathtt{p}]\!]$*, since* $\varepsilon([\![\mathtt{p}]\!] \circ [\![\mathtt{s}]\!]) = \varepsilon(-1,-1,\ldots) = \delta_{-1} \neq \mathcal{P}([\![\mathtt{p}]\!])(\varepsilon([\![\mathtt{s}]\!])) = [\![\mathtt{p}]\!]_*(\delta_0) = \delta_1$*.*

Let us now equip $\mathbb{R}$ *with the topology given by type-checking* p *as described in Example 3.1.7. This makes* $[\![\mathtt{p}]\!]$ *bounded and continuous, and we therefore no longer have* $\varepsilon([\![\mathtt{s}]\!]) = \delta_0$*; indeed* $\lim_n \int [\![\mathtt{p}]\!]\ d\widehat{[\![\mathtt{s}]\!]}_n = -1 \neq [\![\mathtt{p}]\!](0) = 1$*. In fact we now have* $\varepsilon([\![\mathtt{s}]\!]) = \perp$*, i.e.* s *is no longer a sampler targeting anything for this topology, which prevents the failure of Proposition 3.3.3 on* $[\![\mathtt{p}]\!]$*.*

This example also shows that our semantics has provided us with many more morphisms satisfying Proposition 3.3.3 than would have been the case had we only considered programs which are continuous w.r.t. the usual topol-

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

$$\frac{\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu \quad \Gamma \vdash f : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash \mathtt{map}(f, s) : \Sigma\,\mathtt{T} \rightsquigarrow \gamma \mapsto (\llbracket f \rrbracket(\gamma))_* \mu(\gamma)}$$

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

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

$$\frac{\Gamma \vdash \mathtt{prng}(f, x) \rightsquigarrow \mu \quad \Gamma \vdash h : \mathtt{S} \to \mathtt{T}}{\Gamma \vdash \mathtt{prng}(g, h(x)) \rightsquigarrow (\gamma \mapsto \llbracket h \rrbracket(\gamma)_* \mu(\gamma))}\ \llbracket g \rrbracket(\gamma) \circ \llbracket h \rrbracket(\gamma) = \llbracket h \rrbracket(\gamma) \circ \llbracket f \rrbracket(\gamma)$$

Figure 3.6: Rules for asymptotic targeting

ogy 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 3.3.6. *Consider the sampler of Example 3.3.5, but now let* $\mathtt{p} \triangleq \mathtt{if}\ x = 2\ \mathtt{then}\ 1\ \mathtt{else}\ -1$ *instead. To make this function continuous, our semantics adds the open set* $\{2\}$ *to the usual topology of* $\mathbb{R}$*. This does not interfere with the derivation that* $\varepsilon(\llbracket \mathtt{s} \rrbracket) = \delta_0$*, since we can write* $\int f\ d\widehat{\llbracket \mathtt{s} \rrbracket}_n = \int_{\{2\}^c} f\ d\widehat{\llbracket \mathtt{s} \rrbracket}_n + \int_{\{2\}} f\ d\widehat{\llbracket \mathtt{s} \rrbracket}_n = \int_{\{2\}^c} f\ d\widehat{\llbracket \mathtt{s} \rrbracket}_n$*, and similarly for* δ_0*. Because the discontinuity of* $\llbracket \mathtt{p} \rrbracket$ *is not assigned any mass by* δ_0*, the topology on* $\mathbb{R}$ *making* $\llbracket \mathtt{p} \rrbracket$ *continuous no longer prevents* $\varepsilon(\llbracket \mathtt{s} \rrbracket)$ *from converging, and we can therefore safely push* s *forward through* p *using* map*.*

3.3.2 Calculus for asymptotic targeting

Definition 3.3.7. *We will say that the sampler* $\Gamma \vdash s : \Sigma\,\mathtt{S}$ asymptotically targets*, or simply* targets*, the continuous map* $\mu : \llbracket \Gamma \rrbracket \to \mathcal{P}\,\llbracket \mathtt{S} \rrbracket$ *if,*

$$\varepsilon_{\llbracket \mathtt{S} \rrbracket} \circ \llbracket s \rrbracket = \mu.$$

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

We will say that $\Gamma \vdash s : \Sigma\, \mathtt{S}$ *is* k-equidistributed *with respect to* $\mu : [\![\Gamma]\!] \to \mathcal{P}\,[\![\mathtt{S}]\!]$ *if*

$$\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s^k]\!] = \mu^k$$

where the self-product s^k *is defined in eq.* (3.2) *and* $\mu^k(\gamma) = \mu(\gamma)^k$ *is the* k*-fold product of measures.*

The concept of 'equidistribution' as discussed above is a generalisation of a concept of 'k-equidistribution' common to the literature on pseudorandom number generation [40, 41, 42], which is typically only considered for samplers without context targeting a discrete uniform distribution.

'Equidistribution' as defined above is also highly related to the concept of weak mixing of dynamical systems, defined in Definition 2.7.10. Concretely, if a dynamical system is weak mixing, then its sampled trajectories (starting from typical points, of course) are k-equidistributed for all k; this is simply a rewording of Proposition 2.7.13.

We introduce in fig. 3.6 a relation $\rightsquigarrow$ which is sound with respect to asymptotic targeting: that is, if $\Gamma \vdash s : \Sigma\, \mathtt{S} \rightsquigarrow \mu$, then s is a parametrised sampler on $\mathtt{S}$ which asymptotically targets a parametrised distribution μ on $[\![\mathtt{S}]\!]$. Here, we use Greek lower case letters μ, ν to represent (parametrised) distributions in order to emphasise their role as meta-variables, used only in the context of the targeting calculus, and not within the language itself. In the rule for `reweight`, we write the operation of reweighting a measure μ on X by $f : X \to \mathbb{R}_{\geq 0}$ as the renormalised 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 the denominator is finite and nonzero.

The primary purpose of the rules of fig. 3.6 is to verify the sampling techniques discussed in the introduction to this chapter. For example, our rule for `reweight` is simply a statement that importance sampling is valid. Recall that importance sampling inputs samples from a proposal distribution Q, and produces weighted samples which target a desired probability distribution P by

reweighting these proposal samples proportionally according to the density $\frac{dP}{dQ}$, which we assume exists. Our rule for `reweight` simply reverses this argument, observing that a nonnegative f is a valid density as long as it has positive, finite integral under Q. Similarly, the validity of inverse-transform sampling, also discussed in the introduction, follows trivially from our `map` rule. We will shortly see examples of more complex sampling techniques that will require the application of several of our rules.

Note that while our targeting calculus is *sound* (see theorem 3.3.10) but not *complete*. See Remark 3.3.11 for a discussion of why the natural notion of completeness does not hold in this setting.

Example 3.3.8 (Von Neumann extractor)**.** *Verifying the simple von Neumann extractor, introduced in [1], will serve as a useful illustration of the techniques we advocate. Let* $\mathrm{Ber}(p)$ *represent the Bernoulli distribution with parameter* p*, i.e. the Boolean-valued distribution with probability* p *of a true outcome, and probability* $1-p$ *of a false one. Assume access to a closed sampler* `flip` *of Boolean type such that the self-product* $\texttt{flip}^2$ *targets* $\mathrm{Ber}(p)^2$ *for* $p \in (0,1)$*; that is,* `flip` *produces the sample* `True` *with asymptotic frequency* p *and* `False` *with asymptotic frequency* $1-p$*, and* `flip` *is 2-equidistributed. The von Neumann extractor, which we will denote here* v `: ΣB`*, is then the sampler given in listing 3.1.*

```
let accept = λ(x, y) : Σ(B×B) . 1 if x≠y else 0
  in let proj = λ(x, y) : Σ(B×B) . x in
    map(proj, reweight(accept, flip²))
  : ΣB
```

Listing 3.1: Von Neumann extractor

As this sampler (after standard `let`*-reduction) is written in the sampler normal form of Definition 3.2.4, its verification is a straightforward mathematical exercise. Figure 3.7 shows this procedure, applying first the* `reweight` *and then the* `map` *rules of fig. 3.6. All that remains to show after fig. 3.7 is that the measure* $[\![\texttt{proj}]\!]_* ([\![\texttt{accept}]\!] \cdot \mathrm{Ber}(p)^2)$ *obtained as our conclusion is identical*

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

Figure 3.7: Validity of the von Neumann extractor

to Ber($1/2$)*, i.e. the standard demonstration of the validity of the von Neumann extractor. This follows simply from the fact that* `accept` *assigns positive probability only to adjacent samples* $(x, y) \in \mathbb{B}^2$ *which differ, and the samples* $(\mathtt{False}, \mathtt{True})$ *and* $(\mathtt{True}, \mathtt{False})$ *occur with equal probability of* $p(1-p)$*.*

Many of the rules of fig. 3.6 illustrate methods of transforming samplers targeting one distribution into samplers targeting another distribution using different built-in operations; we include only two rules for constructing *new* samplers from scratch. First, we allow a set of 'axioms' for built-in samplers $\mathtt{rand}_i$, each targeting distributions $\mu_i \in \mathcal{P}\,[\![\mathtt{T}_i]\!]$. In some settings, it may be defensible to assume access to 'truly random' samplers – consider 'true' quantum random numbers, for instance. Second, fig. 3.6 incorporates a rule for building samplers from scratch as pseudo-random number generators defined by a deterministic endomap $f : \mathtt{T} \to \mathtt{T}$ and an initial value $t : \mathtt{T}$ via $\mathtt{prng}(f, t) : \Sigma\,\mathtt{T}$. Applying this rule requires showing that the chosen initial point of the sampler generates a μ-typical sequence; see theorem 2.7.5. Note that the side condition on typicality of x can be demonstrated by appealing to unique ergodicity as in Proposition 2.7.6. Note also that, applying Proposition 2.7.13 and $\mathtt{prng}(f, x)^k \approx \mathtt{prng}(f^k \times \ldots \times f^k, (x, f(x), \ldots, f^{k-1}(x)))$ (immediate from the definition of the sampler self-product and fig. 3.5), we see that in the event that the dynamical system denoted by $\mathtt{prng}(f, t)$ is *weak mixing* as opposed to merely ergodic, we can further conclude that $\Gamma \vdash \mathtt{prng}(f, x)^k : \Sigma\,\mathtt{S}^k \rightsquigarrow \mu^k$ for all $k \geq 1$.

The reader might wonder why fig. 3.6 does not have a rule for transforming samplers using the `thin` operation: after all, if σ is a sampler targeting a distribution μ, then only keeping only every n samples seems like it should

produce a good sampler as well. Whilst this rule does hold for samplers which are produced by weak mixing dynamical systems, it is in not in general sound, as the following simple counterexample shows.

Example 3.3.9. *Consider the sampler on* $\{0,1\}$ *defined by the program* $\mathtt{prng}(\lambda x : \mathtt{R} \,.\, 1 - x, 0)$. *This sampler, which generates the unweighted samples* $(0,1,0,1,\ldots)$, *targets the uniform Bernoulli distribution; however, applying* $\mathtt{thin}(2,-)$ *to it yields a sampler which targets the Dirac measure* δ_0.

This example highlights the fact that samplers can be manifestly non-random, and yet from the perspective of inference – that is to say, from the perspective of the topology of weak convergence – target bona fide probability distributions.[3]

Theorem 3.3.10. *Targeting* $\rightsquigarrow$ *is sound: if* $\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$, *then* $\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s]\!] = \mu$.

Proof of Theorem 3.3.10.
By induction on the derivation.

(i) **Built-in samplers.** These axioms – the base case – 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 theorem 3.2.2.

(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.** A direct consequence of Proposition 3.3.3. Note, again, that $[\![S]\!], [\![T]\!]$ may have finer topologies than the standard choices. As a result, the premise of the pushforward rule, that $\varepsilon[\![s]\!](\gamma) = \mu(\gamma)$, is

[3]This is well-known in the literature on Monte Carlo methods; in fact, certain deterministic sequences can obtain faster, deterministic, convergence bounds than i.i.d. random sampling. Many such sequences are studied under the name *quasi-Monte Carlo methods* [43, 44] (about which we'll say no more here).

stronger than it would otherwise be, as the class of continuous test functions $g : [\![S]\!] \to \mathbb{R}$ has been expanded – also expanding the set of continuous functions $[\![f]\!] : [\![S]\!] \to [\![T]\!]$. For example, if $\{x\}$ is open in our topology on $[\![S]\!] = \mathbb{R}$, then our premise requires that our samplers obtain the correct asymptotic frequencies $\mu(\gamma)(\{x\})$, and $f : [\![S]\!] \to [\![T]\!]$ is allowed to be 'discontinuous' (w.r.t. the standard topology) at this point. Conversely, note that adding additional open sets to $[\![T]\!]$ shrinks the set of continuous functions $[\![f]\!] : [\![S]\!] \to [\![T]\!]$, while strengthening the conclusion of our rule. For example, if $\{y\}$ is open in our topology on $[\![T]\!] = \mathbb{R}$, enabling the conclusion that our samplers asymptotically produce the correct asymptotic frequencies for the set $\{y\}$, then $[\![f]\!]^{-1}(\{y\})$ must be open in $[\![S]\!]$.

(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 $x_i \triangleq \pi_1(\pi_i([\![s]\!])$ and $w_i \triangleq \pi_2(\pi_i([\![s]\!])$ that

$$\alpha = \int f \; d\mu = \lim_{N \to \infty} \frac{1}{N} \frac{\sum_{i=1}^{N} f(x_i) w_i}{\sum_{k=1}^{N} w_k} \tag{3.4}$$

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

ν, we have

$$\begin{aligned}
\int g\ d\nu &= \frac{1}{\alpha}\int g.f\ d\mu \\
&= \frac{1}{\alpha}\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} g(x_i)f(x_i)\frac{w_i}{\sum_{k=1}^{N} w_k} && \text{Since } s \text{ targets } \mu \\
&= \left(\lim_{N\to\infty}\frac{1}{N}\frac{\sum_{i=1}^{N} f(x_i)w_i}{\sum_{k=1}^{N} w_k}\right)^{-1}\left(\lim_{N\to\infty}\frac{1}{N}\frac{\sum_{i=1}^{N} g(x_i)f(x_i)w_i}{\sum_{k=1}^{N} w_k}\right) && \text{By } \textit{eq.}\ (3.4) \\
&= \lim_{N\to\infty}\frac{1}{N}\frac{\sum_{i=1}^{N} g(x_i)f(x_i)w_i}{\sum_{i=1}^{N} f(x_i)w_i} \\
&\triangleq \lim_{N\to\infty}\int g\ d\widehat{\texttt{reweight}(f,s)}_n
\end{aligned}$$

also making use of the fact that the limit of a product equals the product of the limits (where both exist). Therefore, $\texttt{reweight}(f, s)$ targets ν.

(vi) **Pseudorandom number generation.** The unconditional `prng` rule is just a restatement of theorem 2.7.5.

(vii) **Homomorphism of dynamical systems.** The conditional `prng` rule essentially follows from the fact that Borel homomorphisms preserve ergodicity; see Proposition 2.7.9. To see that typical points x yield images which are also typical points, though, we will show this rule directly as well. Our result is immediate if we note that, given a homomorphism $g \circ h = h \circ f$ of the form hypothesised, it follows that $g^i(h(x)) = h(f^i(x))$ for any $i \in \mathbb{N}$. Let $u : [\![\mathtt{T}]\!] \to \mathbb{R}$ be any bounded continuous function; again dropping the dependence on γ for clarity of notation, we have

$$\begin{aligned}
\lim_{N\to\infty}\frac{1}{N}\sum_{i=0}^{N-1} u([\![g]\!]^i([\![h]\!]([\![x]\!]))) &= \lim_{N\to\infty}\frac{1}{N}\sum_{i=0}^{N-1} u([\![h]\!]([\![f]\!]^i([\![x]\!]))) \\
&= \int_{[\![S]\!]} (u \circ [\![h]\!])\, d\mu \\
&= \int_{[\![T]\!]} u\, d([\![h]\!]_* \mu)
\end{aligned}$$

as $\texttt{prng}(f, x)$ targets μ by hypothesis, and $u \circ [\![f]\!]$ is bounded continuous.

□

Remark 3.3.11 (Completeness)**.** *It is easily seen that the most obvious notion of completeness,* $\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![\mathtt{s}]\!] = \mu \to \Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$*, does not hold and should not be expected to; see the following example.*

Example 3.3.12. *Let* $s \triangleq \mathtt{prng}(\lambda \mathtt{n} : \mathtt{R}.\mathtt{n} + \mathtt{1}, \mathtt{0})$*; clearly, s does not target any (probability) measure. Transform s using* `map` *to yield the sampler* $s' \triangleq \mathtt{map}(\lambda \mathtt{n} : \mathtt{R}\ .\ \mathtt{2\hat{}}(-\mathtt{1} * \mathtt{n}), s)$*. While s does not target anything, and so our targeting calculus cannot prove that the transformed sampler* s' *targets anything, it is clear that* s' does *target the Dirac measure* δ_0*.*

Remark 3.3.13. *Note that* joint *targeting* $\Gamma \vdash s \otimes t \rightsquigarrow \mu \otimes \nu$ *is a strictly stronger statement than independent targeting* $\Gamma \vdash s \rightsquigarrow \mu, \Gamma \vdash t \rightsquigarrow \nu$*. This is reminiscent of the fact that the marginals of two probability distributions do not determine their joint distribution. A trivial counterexample is given by choosing* $s = t$*, in which case the product* $s \otimes t$ *cannot possibly target a product measure outside of a trivial setting.*

We saw in section 3.3.1 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 3.3.14. *Assume* $\Gamma \vdash s : \Sigma\,\mathtt{S} \rightsquigarrow \mu$*,* $\mathtt{S} \triangleleft \mathtt{T}$ *and* $[\![\mathtt{T}]\!]$ *second-countable; then* $\Gamma \vdash \mathtt{map}(\lambda x : \mathtt{S.cast}\langle \mathtt{T} \rangle x, s) : \Sigma\,\mathtt{T}$ *targets the same measure* μ *on* T*. Moreover, if* $[\![\mathtt{T}]\!]$ *is metrisable, if* U *is in the topology of* $[\![\mathtt{S}]\!]$ *but not* $[\![\mathtt{T}]\!]$ *and* $\mu(\partial_{\mathtt{T}} U) > 0$ *(where* $\partial_{\mathtt{T}}$ *denotes the boundary in* $[\![\mathtt{T}]\!]$*) then s must visit* $\partial_{\mathtt{T}} U$ *i.o. (infinitely often).*

Proof of Proposition 3.3.14.

The first part of the proof follows immediately if we can show that $\mathtt{S} \triangleleft \mathtt{T}$ implies that $[\![\mathtt{S}]\!]$ and $[\![\mathtt{T}]\!]$ are the same measurable space; this will be shown by

induction on the sub-typing derivation. We start by showing that the functor $\mathsf{Borel} : \mathbf{Top} \to \mathbf{Meas}$ commutes with coproducts. This will prove the base case, the coproduct rule, and the last two rules of fig. 3.2b.

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 $\mathsf{Borel} : \mathbf{Top} \to \mathbf{Meas}$ commutes with products we need the extra assumption that the spaces are second-countable. A proof can then be found in e.g. [45, p244].

For the second part of the proof, let U be in the topology of $[\![\mathtt{S}]\!]$ but not in the topology of $[\![\mathtt{T}]\!]$. This means that $\partial_{\mathtt{T}}(U) = U \cap \mathrm{int}_{\mathtt{T}}(U) \neq$ is open in $[\![\mathtt{S}]\!]$ (since it's the intersection of two open sets in $[\![\mathtt{S}]\!]$). In particular it is a continuity set in $[\![\mathtt{S}]\!]$ (since it is open, its interior is the empty set and it can therefore not have any μ-mass). By the Portmanteau lemma, Lemma 2.5.3, which applies since the spaces are assumed to be metrisable, we must thus

have

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

In particular, this is clearly impossible if $[\![s]\!]$ only visits $\partial_{\mathtt{T}}(U)$ finitely many times. □

Example 3.3.15. *Suppose we want* $s : \Sigma\,\mathtt{R} \rightsquigarrow \mathrm{Bern}(1/2)$. *A sampler alternating between the sampler* $z \triangleq \mathtt{prng}(\lambda x : \mathtt{R}\,.\,x/2, 1)$ *of Example 3.3.5 and its shifted version* $\mathtt{map}(\lambda x : \mathtt{R}\,.\,1 + x, z)$ *will satisfy the condition, but will never visit* 0 *or* 1*! We can use the previous result to enforce that a sampler* s *targeting* $\mathrm{Bern}(1/2)$ *should visit* 0 *i.o. by constructing* s *in such a way that it has type* $\Sigma((x,0)^{-1}\mathtt{Neq} + (x,0)^{-1}\mathtt{Eq})$ *(see Example 3.1.7). We can, in the same manner, enforce that a sampler* s' *targeting* $\mathrm{Bern}(1/2)$ *visits* 1 *i.o. Finally, using the last two rules of fig. 3.2b which build the coarsest common refinement of two topologies, we can combine* s *and* s' *to create a sampler targeting* $\mathrm{Bern}(1/2)$ *and guaranteed to visit* $0, 1$ *i.o.*

A more interesting application of our rules, which will make use of the quirks of our typing system, is the verification of rejection sampling. Rejection sampling makes use of two samplers: a proposal sampler s and a standard uniform sampler `rand`. It also uses a real-valued function $f : \mathtt{S} \to \mathtt{R}$ and an upper bound $K > 0$ on f, and produces samples from its target distribution, which is the proposal distribution reweighted by the function f.[4] Using these ingredients, a general rejection sampling algorithm is implemented in listing 3.2.

We will abbreviate the program given in listing 3.2 as $\mathtt{reject}(f, K, s)$. Note that the type `T` has been left unspecified; we will find this type's form as we type-check below.

[4]Consider for the purposes of illustration a Bayesian setting in which s is a sampler from our prior and f computes the likelihood of a set of data. In this case, rejection sampling will give us samplers from the posterior distribution.

```
let accept = λ(x,u) : T . if u ≤ f(x) / K then 1 else 0
  in let proj = λz : T. fst(cast⟨S×R+⟩(z)) in
    map(proj, reweight(accept, s ⊗ rand))
  : ΣS
```

Listing 3.2: Rejection sampling

Theorem 3.3.16. *Rejection sampling is valid:*

$$\frac{\Gamma \vdash (s \otimes \mathtt{rand}) : \Sigma\,\mathtt{S} \times \mathtt{R}^+ \rightsquigarrow (\mu \otimes U) \quad \Gamma \vdash f : \mathtt{S} \to \mathtt{R}^+ \quad \Gamma \vdash K : \mathtt{R}^+}{\Gamma \vdash \mathtt{reject}(f, K, s) : \Sigma\,\mathtt{S} \rightsquigarrow \gamma \mapsto [\![f]\!](\gamma) \cdot \mu(\gamma)}\ \dagger$$

where the side conditions † *necessary are*

1. $\int_{[\![S]\!]} [\![f]\!](\gamma)\, d\mu(\gamma) \in (0, \infty)$;
2. $\forall x \in [\![\mathtt{S}]\!], [\![K(\gamma)]\!] \geq [\![f]\!](\gamma)(x)$.

Proof of Theorem 3.3.16.

Before we discuss the verification of rejection sampling as an approximate sampling procedure, we must first type-check this program in detail. The type T was not specified in listing 3.2; we determine its form by type-checking accept in fig. 3.8. To keep the derivation readable, we define $t \triangleq (y, f(x) * K)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : \mathtt{S}, y : \mathtt{R}^+ \vdash x : \mathtt{S} \quad \vdash f : \mathtt{S} \to \mathtt{R}^+ \quad \vdash K : \mathtt{R}^+}{x : \mathtt{S}, y : \mathtt{R}^+ \vdash f(x) * K : \mathtt{R}^+} \quad x : \mathtt{S}, y : \mathtt{R}^+ \vdash y : \mathtt{R}^+}{x : \mathtt{S}, y : \mathtt{R}^+ \vdash (y, f(x) * K) : \mathtt{R}^+ \times \mathtt{R}^+}}{(x,y) : t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1)) \vdash (y, f(x) * K) : <^{-1}(0) + <^{-1}(1)}\ <^{-1}(0) + <^{-1}(1) \triangleleft \mathtt{R}^+ \times \mathtt{R}^+}{(x,y) : t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1)) \vdash y < f(x) * K : \mathtt{B} \quad \vdash 0 : \mathtt{R}^+ \quad \vdash 1 : \mathtt{R}^+}}{(x,y) : t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1)) \vdash \mathtt{if}\ y < f(x) * K\ \mathtt{then}\ 1\ \mathtt{else}\ 0 : \mathtt{R}^+}}{\vdash \lambda(x,y) : t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1))\ .\ \mathtt{if}\ y < f(x) * K\ \mathtt{then}\ 1\ \mathtt{else}\ 0 : t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1)) \to \mathtt{R}^+}$$

Figure 3.8: Type-derivation of accept

Next, we show that the entire rejection sampling algorithm type-checks in fig. 3.9, defining $\mathtt{T} = t^{-1}(<^{-1}(0)) + t^{-1}(<^{-1}(1))$ and $\mathtt{proj} \triangleq \lambda z : \mathtt{T}.\mathtt{fst}(\mathtt{cast}\langle \mathtt{S} \times \mathtt{R}^+ \rangle z)$ for brevity.

We can now verify rejection sampling as an approximate sampling method. In fig. 3.10, we begin by assuming that our samplers s and rand each target

$$\dfrac{\dfrac{\dfrac{\dfrac{z : \mathtt{T} \vdash z : \mathtt{T}}{z : \mathtt{T} \vdash \mathtt{cast}\langle \mathtt{S} \times \mathtt{R}^+ \rangle z : \mathtt{S} \times \mathtt{R}^+}\ \mathtt{T} \triangleleft \mathtt{S} \times \mathtt{R}^+}{z : \mathtt{T} \vdash \mathtt{fst}(\mathtt{cast}\langle \mathtt{S} \times \mathtt{R}^+ \rangle z) : \mathtt{S}}}{\vdash \mathtt{proj} : \mathtt{T} \to \mathtt{S}} \qquad \dfrac{\vdash \mathtt{accept} : \mathtt{T} \to \mathtt{R}^+ \qquad \dfrac{\vdash s : \Sigma\, \mathtt{S} \quad \vdash \mathtt{rand} : \Sigma\, \mathtt{R}^+}{\vdash s \otimes \mathtt{rand} : \Sigma\,(\mathtt{S} \times \mathtt{R}^+)}}{\vdash s \otimes \mathtt{rand} : \Sigma\, \mathtt{T}}\ \Sigma\, \mathtt{T} \triangleleft \Sigma\,(\mathtt{S} \times \mathtt{R}^+), \Gamma = \emptyset}{\vdash \mathtt{reweight}(\mathtt{accept}, s \otimes \mathtt{rand})) : \Sigma\, \mathtt{T}}}{\vdash \mathtt{map}(\mathtt{reweight}(\mathtt{accept}, s \otimes \mathtt{rand}), \mathtt{proj}) : \Sigma\, \mathtt{S}}$$

Figure 3.9: Type-derivation of the rejection sampling algorithm

some proposal distribution μ and the uniform distribution U on the unit interval – and that more strongly, these two samplers *jointly* target these two probability distributions (see Remark 3.3.13). As shown in fig. 3.10, application of our targeting calculus quickly shows that the samples produced by rejection sampling target the probability measure $[\![\mathtt{proj}]\!]_* ([\![\mathtt{accept}]\!] \cdot (\mu \otimes U))$.

$$\dfrac{\dfrac{\vdash s \otimes \mathtt{rand} : \Sigma\, \mathtt{T} \rightsquigarrow \mu \otimes U \quad \vdash \mathtt{accept} : \mathtt{T} \to \mathtt{R}^+}{\vdash \mathtt{reweight}(\mathtt{accept}, s \otimes \mathtt{rand}) : \Sigma\, \mathtt{T} \rightsquigarrow [\![\mathtt{accept}]\!] \cdot (\mu \otimes U)} \quad \vdash \mathtt{proj} : \mathtt{T} \to \mathtt{S}}{\vdash \mathtt{map}(\mathtt{proj}, \mathtt{reweight}(\mathtt{accept}, s \otimes \mathtt{rand})) : \Sigma\, \mathtt{S} \rightsquigarrow [\![\mathtt{proj}]\!]_* ([\![\mathtt{accept}]\!] \cdot (\mu \otimes U))}$$

Figure 3.10: Validity of rejection sampling

All that remains is to show that in fact $[\![\mathtt{proj}]\!]_* ([\![\mathtt{accept}]\!] \cdot (\mu \otimes U)) = P$. Given the denotational semantics $[\![\mathtt{proj}]\!](x, u) = x, [\![\mathtt{accept}]\!](x, u) = \begin{cases} 1 & u \leq \frac{[\![f]\!](x)}{K} \\ 0 & else \end{cases}$, and the fact that K is an upper bound on the nonnegative function f, this is immediate. For measurable $A \subseteq S$, ignoring the normalisa-

tion constant for readability,

$$\begin{aligned}
\llbracket \texttt{proj} \rrbracket_* \left(\llbracket \texttt{accept} \rrbracket \cdot (\mu \otimes U) \right)(A) &= \left(\llbracket \texttt{accept} \rrbracket \cdot (\mu \otimes U) \right) \left(\llbracket \texttt{proj} \rrbracket^{-1}(A) \right) \\
&\propto \int_{\llbracket \texttt{proj} \rrbracket^{-1}(A)} \llbracket \texttt{accept} \rrbracket (x,u)\, d(\mu \otimes U)(x,u) \\
&= \int_A \int_0^1 \llbracket \texttt{accept} \rrbracket (x,u)\, dU(u)\, d\mu(x) \\
&= \int_A \int_0^{\llbracket f \rrbracket(x)/K} dU(u)\, d\mu(x) \\
&= \frac{1}{K} \int_A \llbracket f \rrbracket (x)\, d\mu(x).
\end{aligned}$$

Therefore, rejection sampling, like importance sampling, produces a sampler targeting the probability distribution which has density (proportional to) $\llbracket f \rrbracket$ with respect to the proposal distribution Q. □

Example 3.3.17. *For a simple application of listing 3.2, consider a small Bayesian inference problem with prior $q(z)$ and Gaussian likelihood $p(x \mid z) = \mathrm{N}(x \mid z, 1)$. We will apply rejection sampling to generate samples from the posterior distribution $p(z \mid x)$ given by Bayes' theorem, proposing samples from a sampler s which we assume targets the prior q. Bayes' theorem tells us that the density of the posterior distribution with respect to the prior is (proportional to) $f(z) = p(x \mid z)$, neglecting the normalisation constant. To apply rejection sampling, we then need to deduce a bound $K = \sup_{z \in Z} p(x \mid z) = \sup_{z \in Z} \mathrm{N}(x \mid z, 1)$; by symmetry of the Gaussian density this is obviously maximised by choosing $z = x$, yielding $K = \frac{1}{\sqrt{2\pi}}$. Therefore, it follows by theorem 3.3.16 (and a trivial application of the* `let`*-reduction rule in fig. 3.5) that listing 3.3 samples from the intended posterior distribution (given the assumption that s targets our prior, and the requisite independence assumptions).*

```
let K = 1 / (2*π)^2 in
let f = λz : R . K * (z - x)^2 / 2 in
let accept = λ(x,u) : T . if u ≤ f(x) / K then 1 else 0
in let proj = λz : T. fst(cast⟨S×R⁺⟩(z)) in
  map(proj, reweight(accept, s ⊗ rand))
  : ΣS
```

Listing 3.3: Rejection sampling: example

Chapter 4

Construction and verification of stochastic process samplers

This chapter extends the sampling language discussed in the previous chapter to include samplers which meaningfully target stochastic processes, which can be understood as infinite collections of random variables, or as function-valued random variables; see section 2.8.

This extension to stochastic processes is necessary in order to formalise a number of useful probabilistic programs. In the early days of computing, the modern theory of stochastic processes had not yet been formalised, and it would be decades until the theory and applications of continuous-time stochastic processes and Gaussian processes were well-understood and commonplace. These days, though, many of the most interesting and complex applications of probabilistic programming languages feature stochastic processes. For example, higher-order probabilistic programming languages can implement non-parametric Bayesian techniques such as Gaussian process regression [46], infinite mixture models based on Dirichlet processes, continuous-time filtering and smoothing, and diffusion models [47] (which have experienced a surge of interest in recent years). The verification of these complex probabilistic programs requires a language with a sound, succinct treatment of stochastic processes. Fortunately, while these programs are complex, they can typically be understood *compositionally* as being constructed from a small set of simpler

procedures, which is key for scalable verification.

The contributions of this chapter are organised as follows. Section 4.1 first extends the language of the previous chapter, adding the features necessary to construct samplers for stochastic processes – more concretely, families of samplers which target the corresponding marginals of stochastic processes. Framing this concept within the language will, as we'll see, require the addition of dependent types, which will broaden but also significantly complicate the presentation of the denotational semantics. Next, in analogy to how the previous chapter verified samplers by relating the infinite sequences they produce to probability measures, section 4.3.2 shows that we should verify stochastic process samplers by relating the infinite family of finite-dimensional marginal samplers they produce to the stochastic process in question by applying Bochner's theorem (see the preliminaries, section 2.8). Having established this link, section 4.3.3 and section 4.3.4 finish by demonstrate the soundness of a number of techniques commonly used for constructing and transforming stochastic processes, enabling the compositional verification of programs which use stochastic processes. In particular, we identify sets of programs which provably construct stochastic process samplers, and which provably transform one stochastic processes sampler into another.

Stochastic process samplers. Incorporating stochastic processes into a probabilistic programming language, perhaps surprisingly, raises a number of interesting type-theoretic questions. To see why, it is necessary to start by considering the question, *what does "sampling from a stochastic process" concretely mean?* As stochastic processes are probability measures over function spaces, one answer could be that a program sampling from a stochastic process with index set T taking values in a measurable space S is a program which, given a seed, returns an object of function type $T \to S$, i.e. a concrete sample path. However, this understanding of "sampling from a stochastic process" is not computationally achievable. Let $T = \mathbb{R}_{\geq 0}$, $S = \mathbb{R}$, and assume that we wish to sample from a normally distributed white noise process. Computing

a sample path from this process means preparing an uncountable sequence of independent normal samples $(x_t)_{t\in\mathbb{R}_{\geq 0}}$, which is clearly not computationally possible. More realistically, if we assume that reals are concretely represented by the finite set $\mathbb{F}$ of floating-point numbers, and that T is a compact interval, say [0,1], then returning a concrete sample path means returning a finite but enormous (2^{30} elements in single-precision) sequence of normal deviates $(x_t)_{t\in\mathbb{F}\cap[0,1]}$ every time the sampler is invoked. This is clearly not practical from a memory perspective, but it is also doubtful that any practical pseudo-random number generator could produce so many deviates *independently*. The same problem will present itself for any choice of stochastic process indexed by a sufficiently large set T. By virtue of the algorithmic "incompressibility" of randomness, no sample path can have a shorter description than such an enumeration.

Dependent types. Thus, sampling from a stochastic process cannot mean producing sample paths. Instead, one usually understands sampling from a stochastic process as the ability to sample from any finite-dimensional marginal of the process [46, 5]. This perspective has profound type-theoretic implications. Given any tuple $(t_1, \ldots, t_n)$ of elements of T, a stochastic process sampler must be able to return samples in S^n which are distributed according to the joint distribution of the process observed at "times" $(t_1, \ldots, t_n)$. In particular, the output *type* of the sampler depends on the size of the input tuple. This suggests that a stochastic process sampler should be described using the language of *dependent types*. Specifically, if we write $\Sigma\,\mathtt{S}$ for the type of (standard) samplers returning samples of type $\mathtt{S}$, then a sampler for a $\mathtt{T}$-indexed $\mathtt{S}$-valued stochastic process will have the Π-type

$$\Pi\, n : \mathtt{N}\,.\,\mathtt{Vec}_n(\mathtt{T}) \to \Sigma\,(\mathtt{Vec}_n(\mathtt{S})) \tag{4.1}$$

where $\mathtt{Vec}_n(\mathtt{T})$ is the (dependent) type of n-dimensional vectors of type $\mathtt{T}$. We denote the type given in eq. (4.1), which will play a central role in the chapter to come, by $\mathtt{Marginal}(\mathtt{T}, \mathtt{S})$. Most programs of this type do not sample from

a stochastic process—we will identify those that do in section 4.3.3—but in order to sample from a stochastic process in the manner described above, a program must have this type.

Since stochastic processes are implemented in terms of their finite-dimensional marginals, their denotational semantics typically pays little attention to the existence of the *laws* of stochastic processes, i.e. their defining measures on (typically infinite-dimensional) function spaces. Instead, much research on the semantics of these implementations has focused on laziness, recursion, and on side-effects which are often hidden within their invocation [48, 46, 11]. However, in order to verify correctness of a stochastic process implemented as a program of type $\texttt{Marginal}(\texttt{T}, \texttt{S})$, we will need to check that it targets the desired stochastic process, i.e. the correct law. Thus, whilst stochastic processes are, as programs of type $\texttt{Marginal}(\texttt{T}, \texttt{S})$, described operationally in terms of their finite-dimensional marginals, to verify the correctness of these programs is to link this perspective with the probability-measure-on-sample-paths perspective described above. This verification task is this chapter's primary concern.

The sampling language we lay out in this chapter includes and extends the sampling language of the previous chapter. All of the results of the previous chapter will extend to this setting, if only to the subset of our now-extended language which is identical to the language defined in the previous chapter. As a result, we will avoid presenting details of the syntax, semantics, and verification which were discussed in the previous chapter, focusing on the new features necessary to enable constructing and verifying samplers for stochastic processes. We will begin by introducing the syntax and semantics of these new elements, so as to formalise the stochastic process sampler type defined in eq. (4.1).

4.1 Language

In this section, we extend the language of chapter 3 by adding certain dependent types, giving the language the ability to package different finite-dimensional samplers into a single object. The presence of these dependent types complicates the language of chapter 3 somewhat, but as we shall see it is semantically very natural; Π-types provide exactly the right structure in which to construct stochastic processes via Bochner's theorem (section 4.3.3). Just as in chapter 3, the language is interpreted in a category of topological spaces and continuous maps, in order to keep the targeting relation (which is defined in terms of weak convergence) well-defined.

4.1.1 Syntax

The grammars of the various types and terms of our language are given in fig. 4.1, which extends the the corresponding fig. 3.1 in the previous chapter with a number of new language features. The rules for well-formed types are given in fig. 4.2, and the rules for well-formed terms are given in fig. 4.3. Note in particular that, following [49], our contexts $\phi; \Gamma$ now have two pieces: an indexing context ϕ, which will keep track of all the indices on which types depend in a program, and a standard context Γ, which can include variables indexed in ϕ. We will now go through each of these new features in detail.

Indexing types and indexing terms.

We will only need a limited collection of dependent types, and so we follow a restricted syntax in the spirit of dependent ML [49]. We begin by defining the grammar of indexing types and indexing terms in figs. 4.1a and 4.1b, where $\mathtt{G} \in$ Ground ranges over a set of ground types which includes $\mathtt{B}, \mathtt{N}, \mathtt{R}$ and $\mathtt{R}^+$; x ranges over a set Var of variables; g ranges over constants taking values in one of the ground types $\mathtt{G} \in$ Ground; and f ranges over a set of built-in functions taking their values in one of the ground types.

An **indexing context** ϕ will be a context of indexing types $x_1 : \mathtt{I}_1, \ldots, x_n : \mathtt{I}_n$ (we broadly follow the notation of [49]). The rules for well-

$$\mathtt{I} ::= \mathtt{G} \mid \mathtt{I} \times \mathtt{I}$$

(a) Indexing type grammar

$$i ::= x \mid g \mid f(i, \ldots, i) \mid (i, i) \mid \mathtt{fst}(i) \mid \mathtt{snd}(i) \mid \mathtt{cast}\langle \mathtt{I} \rangle i$$

(b) Indexing term grammar

$$\begin{aligned} \mathtt{T} ::= {} & \mathtt{G} \in \text{Ground} \mid 1 \\ & \mid \mathtt{T} \times \mathtt{T} \mid \mathtt{T} + \mathtt{T} \mid \mathtt{T} \to \mathtt{T} \mid \Sigma\, \mathtt{T} \\ & \mid \Pi\, i : \mathtt{I}\,.\,\mathtt{T} \mid {}_i\mathtt{I}_i \mid \mathtt{Vec}_i(\mathtt{T}) \mid \mathtt{Mat}_{i \times i}(\mathtt{T}) \qquad i, i : \mathtt{I} \end{aligned}$$

(c) Type grammar

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

(d) Term grammar

$$\begin{aligned} v ::= {} & x \in \text{Var} \mid b \in \{\mathtt{True}, \mathtt{False}\} \mid n \in \mathbb{N} \mid r \in \mathbb{R} \\ & \mid (v, v) \mid \mathtt{in}_i\,(v) \mid \lambda x\text{:}\ \mathtt{T}.\ v \end{aligned}$$

(e) Value grammar

Figure 4.1: Grammars

formed indexing terms are given in fig. 4.2a.

(Sub-)Types.

The types of our language are given by the grammar fig. 4.1c. The types we consider here are the types of chapter 3 plus Π-, vector and matrix types.

Unlike chapter 3, we have further rules for well-formed types; these are given in fig. 4.2b. Types depending on *open* indexing terms will be called

$$\frac{}{\phi \vdash g : \mathtt{G}}\ g \in [\![\mathtt{G}]\!], \mathtt{G} \in \text{Ground} \qquad \frac{}{\phi \vdash x : \mathtt{I}}\ x \in \text{dom}(\phi)$$

$$\frac{\phi \vdash x_1 : \mathtt{G}_1 \quad \ldots \quad \phi \vdash x_n : \mathtt{G}_n}{\phi \vdash f(x_1, \ldots, x_n) : \mathtt{G}}\ \text{Func} \ni f : \mathtt{G}_1 \times \ldots \times \mathtt{G}_n \to \mathtt{G}$$

$$\frac{\phi \vdash t : \mathtt{I}_1 \times \mathtt{I}_2}{\phi \vdash \mathtt{fst}(t) : \mathtt{I}_1} \quad \frac{\phi \vdash t : \mathtt{I}_1 \times \mathtt{I}_2}{\phi \vdash \mathtt{snd}(t) : \mathtt{I}_2} \qquad \frac{\phi \vdash s : \mathtt{I}_1 \quad \phi \vdash t : \mathtt{I}_2}{\phi \vdash (s,t) : \mathtt{I}_1 \times \mathtt{I}_2}$$

(a) Well-formed indexing terms

$$\frac{}{\vdash \mathtt{G} : \mathtt{Type}}\ \mathtt{G} \in \text{Ground} \qquad \frac{\phi \vdash \mathtt{T} : \mathtt{Type}}{\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}}\ i \notin \text{dom}\,\phi$$

$$\frac{\phi \vdash \mathtt{T} : \mathtt{Type}}{\phi \vdash \Sigma\,\mathtt{T} : \mathtt{Type}} \qquad \frac{\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}}{\phi \vdash \Pi i : \mathtt{I}\,.\,\mathtt{T} : \mathtt{Type}}$$

$$\frac{\phi \vdash \mathtt{S} : \mathtt{Type} \quad \phi \vdash \mathtt{T} : \mathtt{Type}}{\phi \vdash \mathtt{S} \star \mathtt{T} : \mathtt{Type}}\ \star \in \{\times, +, \to\} \qquad \frac{\phi \vdash s : \mathtt{I} \quad \phi \vdash t : \mathtt{I} \quad \phi \vdash \mathtt{I} : \mathtt{Type}}{\phi \vdash {}_s\mathtt{I}_t : \mathtt{Type}}\ s,t \text{ open}$$

$$\frac{\phi \vdash t : \mathtt{N} \quad \phi \vdash \mathtt{T} : \mathtt{Type}}{\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}} \qquad \frac{\phi \vdash s : \mathtt{N} \quad \phi \vdash t : \mathtt{N} \quad \phi \vdash \mathtt{T} : \mathtt{Type}}{\phi \vdash \mathtt{Mat}_{s \times t}(\mathtt{T}) : \mathtt{Type}}$$

(b) Type formation rules

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

$$\frac{\mathtt{S} \lhd \mathtt{T}}{F\mathtt{S} \lhd F\mathtt{T}}\ F \in \{\Sigma\,(-), \mathtt{Vec}_n(-), \mathtt{Mat}_{m \times n}(-), \Pi\, n : \mathtt{N}\,.\,(-)\}$$

$$\frac{\mathtt{S}_1 \lhd \mathtt{S}_2 \qquad \mathtt{T}_1 \lhd \mathtt{T}_2}{\mathtt{S}_1 * \mathtt{T}_1 \lhd \mathtt{S} * \mathtt{T}_2}\ * \in \{+, \times\} \qquad \frac{\mathtt{S}_1 \lhd \mathtt{S}_2 \qquad \mathtt{T}_1 \lhd \mathtt{T}_2}{\mathtt{S}_2 \to \mathtt{T}_1 \lhd \mathtt{S}_1 \to \mathtt{T}_2}$$

$$\frac{\mathtt{S}_1 \lhd \mathtt{T} \quad \mathtt{S}_2 \lhd \mathtt{T}}{\mathtt{S}_1 \cap \mathtt{S}_2 \lhd \mathtt{S}_1} \qquad \frac{\mathtt{S}_1 \lhd \mathtt{T} \quad \mathtt{S}_2 \lhd \mathtt{T}}{\mathtt{S}_1 \cap \mathtt{S}_2 \lhd \mathtt{S}_2}$$

(c) Subtyping rules

Figure 4.2: Well-formed indexing terms, type-formation, and subtype rules

dependent types. All other types will be called **closed types**.

Since we have a limited supply of Π-types, we need to explicitly include function types $\mathtt{S} \to \mathtt{T}$, as opposed to considering these as Π-types themselves. The sampler types $\Sigma\,\mathtt{T}$ should not be confused with the Σ-types of dependent type theory, which do not feature in our language. As in chapter 3, $\Sigma\,\mathtt{T}$ is interpreted as the set of weighted streams of type $\mathtt{T}$. The **pullback types** ${}_s\mathtt{T}_t$ of chapter 3, while they retain a particular syntax, will now be interpreted as the dependent types they are. The only other dependent types we introduce are **vector types** and **matrix types**.

The language also includes a subtyping relation $\lhd$ described in fig. 4.2c. The purpose of this relation, as it was in chapter 3, is to encode topological information which will allow the interpretation of functions which are discontinuous for the standard topologies. The first rule introduces the ground types $f^{-1}(0) + f^{-1}(1)$ which will be interpreted as $\mathbb{R}^2$ equipped with the coarsest refinement of the usual topology making the comparison operator $f \in \{<, \leq, >, \geq, =, \neq\}$ continuous. The types $\mathtt{S_1} \cap \mathtt{S_2}$ will be interpreted as the coarsest common refinement of the topologies on $\mathtt{S_1}, \mathtt{S_2}$; they are as previously syntactic sugar for the pullback types ${}_{\mathtt{cast}\langle\mathtt{T}\rangle x_1:\mathtt{S_1}}\mathtt{T}_{\mathtt{cast}\langle\mathtt{T}\rangle x_2:\mathtt{S_2}}$.

Terms

The terms of our language are given by the grammar fig. 4.1d. The first seven lines, containing standard programming constructs and sampler operations, are unaltered from chapter 3. The last two lines contain our new vector and matrix operations, which will be essential for building samplers for the marginal distributions of stochastic processes. More specifically, $\mathtt{vec}(n)(t)$ (resp. $\mathtt{mat}(m,n)(t)$) constructs a vector of type $\mathtt{Vec}_n(\mathtt{T})$ (resp. a matrix of type $\mathtt{Mat}_{m\times n}(\mathtt{T})$) given a map $t : \mathtt{N} \to \mathtt{T}$ (resp. $t : \mathtt{N} \times \mathtt{N} \to \mathtt{T}$). The operation $\mathtt{get}(n)(i,v)$ (resp. $\mathtt{get}(m,n)(i,j,M)$) gets the i^{th} (resp. $(i,j)^{th}$) element of an n-dimensional vector v of type $\mathtt{T}$ (resp. $m \times n$-dimensional matrix M of type $\mathtt{T}$). The operation $\mathtt{reduce}(s)(f,t,v)$ repeatedly performs the binary operation f to the elements of an s-dimensional vector v starting with an element t.

The rules for well-formed terms are given in fig. 3.2a. Again, following [49], we have split our contexts in two: a context $\phi;\Gamma$ consists of an indexing context ϕ and a standard context Γ.

The rules on the first three lines of fig. 4.3a are standard, introducing constants, built-in functions, variables, weakening and casting. Note that the variable introduction and weakening rules ensure that a type must be constructed from the indexing context ϕ using the rules of fig. 4.2b before they can be introduced into the (standard) context. The rule on the third line is the "context-restriction" rule of chapter 3 adapted to our dependently-typed language, the notation $t^{-1}(\mathtt{S})$ being as previously syntactic sugar for the pullback type $_{\mathtt{cast}\langle \mathtt{I}\rangle x:\mathtt{S}}\mathtt{I}_t$. Next, the rules for pairing, pattern matching, let binding, lambda-abstractions and evaluations are the same as chapter 3 with the addition of the indexing context ϕ, as are the typing rules for sampler operations.

Only the rules for introduction and evaluation of Π-types, and for creating and manipulating vectors and matrices, contained within fig. 4.3c, are substantial alterations from the previous chapter. These are implemented as expected given the intended meaning sketched above.

4.1.2 Operational semantics.

The operational semantics of terms of dependent type is mostly straightforward, as they are naturally interpreted in the same way as λ-terms. For each of our built-in dependent operations, such as `get`, and each possible size $n \in \mathbb{N}$, we add a corresponding rule to the operational semantics – in this case, one which inputs a natural number $i \in \{1, \ldots, n\}$ and a vector of size n, and outputs the ith element of the vector. The operational semantics for these vector and matrix operations is given in fig. 4.4, the operational semantics for all other operations being unchanged from chapter 3.

4.1.3 Denotational semantics.

As explained in chapter 3, the key correctness criterion for a sampler is given by the concept of *targeting a probability measure*. Since targeting is defined in

$$\frac{}{\phi;\emptyset \vdash g : \mathtt{G}}\ g \in [\![\mathtt{G}]\!], \mathtt{G} \in \text{Ground}$$

$$\frac{\phi;\Gamma \vdash t_1 : \mathtt{G}_1 \quad \ldots \quad \phi;\Gamma \vdash t_n : \mathtt{G}_n}{\phi;\Gamma \vdash f(t_1,\ldots,t_n) : \mathtt{G}}\ \text{Func} \ni f : \mathtt{G}_1 \times \ldots \times \mathtt{G}_n \to \mathtt{G}$$

$$\frac{\phi;\Gamma,\Delta \vdash t : \mathtt{T} \quad \phi \vdash \mathtt{S} : \mathtt{Type}}{\phi;\Gamma,x : \mathtt{S},\Delta \vdash t : \mathtt{T}} \qquad \frac{\phi \vdash \mathtt{T} : \mathtt{Type}}{\phi;x : \mathtt{T} \vdash x : \mathtt{T}}\ x \in \text{Var}$$

$$\frac{\phi \vdash t : \mathtt{I}}{\phi;(x_1,\ldots,x_n) : t^{-1}(\mathtt{S}) \vdash t : \mathtt{S}}\ \mathtt{S} \lhd \mathtt{I}, \phi = x_1 : \mathtt{I}_1,\ldots,x_n : \mathtt{I}_n$$

$$\frac{\phi;\Gamma \vdash t : \mathtt{S} \times \mathtt{T}}{\phi;\Gamma \vdash \mathtt{fst}(t) : \mathtt{S}} \qquad \frac{\phi;\Gamma \vdash s : \mathtt{S} \quad \phi;\Gamma \vdash t : \mathtt{T}}{\phi;\Gamma \vdash (s,t) : \mathtt{S} \times \mathtt{T}} \qquad \frac{\phi;\Gamma \vdash t : \mathtt{S} \times \mathtt{T}}{\phi;\Gamma \vdash \mathtt{snd}(t) : \mathtt{T}}$$

$$\frac{\phi;\Gamma \vdash t : \sum_{i\in I} \mathtt{T}_i \quad \phi;\Gamma,x_i : \mathtt{T}_i \vdash s_i : \mathtt{T}}{\phi;\Gamma \vdash \mathtt{case}\ t\ \mathtt{of}\ \{(i,x_i) \Rightarrow s_i\}_{i\in I} : \mathtt{T}}$$

$$\frac{\phi;\Delta \vdash t : \mathtt{S}}{\phi;\Gamma \vdash \mathtt{cast}\langle \mathtt{T}\rangle t : \mathtt{T}}\ \mathtt{S} \lhd \mathtt{T}, \Gamma \lhd \Delta \qquad \frac{\phi;\Gamma \vdash t : \mathtt{T}_j}{\phi;\Gamma \vdash \mathtt{in}_j\,(t) : \sum_{i\in n} \mathtt{T}_i}\ j \in n$$

$$\frac{\phi;\Gamma,x : \mathtt{S} \vdash t : \mathtt{T}}{\phi;\Gamma \vdash \lambda x : \mathtt{S}\,.\,t : \mathtt{S} \to \mathtt{T}} \qquad \frac{\phi;\Gamma \vdash s : \mathtt{S} \quad \phi;\Gamma \vdash t : \mathtt{S} \to \mathtt{T}}{\phi;\Gamma \vdash t(s) : \mathtt{T}} \qquad \frac{\phi;\Gamma,x : \mathtt{S} \vdash t : \mathtt{T} \quad \phi;\Gamma \vdash s : \mathtt{S}}{\phi;\Gamma \vdash \mathtt{let}\ x = s\ \mathtt{in}\ t : \mathtt{T}}$$

(a) Typing rules for general programming constructs

$$\frac{\phi;\Gamma \vdash t : \Sigma\,\mathtt{T}}{\phi;\Gamma \vdash \mathtt{hd}(t) : \mathtt{T}} \qquad \frac{\phi;\Gamma \vdash t : \Sigma\,\mathtt{T}}{\phi;\Gamma \vdash \mathtt{tl}(t) : \Sigma\,\mathtt{T}}$$

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

$$\frac{\phi;\Gamma \vdash s : \mathtt{T} \to \mathtt{T} \quad \phi;\Gamma \vdash t : \mathtt{T}}{\phi;\Gamma \vdash \mathtt{prng}(s,t) : \Sigma\,\mathtt{T}} \qquad \frac{\phi;\Gamma \vdash n : \mathtt{N} \quad \phi;\Gamma \vdash t : \Sigma\,\mathtt{T}}{\phi;\Gamma \vdash \mathtt{thin}(n,t) : \Sigma\,\mathtt{T}}$$

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

(b) Typing rules for sampler-functions

$$\frac{\phi,i : \mathtt{I};\Gamma \vdash t : \mathtt{T}}{\phi;\Gamma \vdash (\lambda i : \mathtt{I}\,.\,t) : (\Pi\, i : \mathtt{I}\,.\,\mathtt{T})}\ i \notin FV(\Gamma) \qquad \frac{\phi;\Gamma \vdash t : \Pi\, i : \mathtt{I}\,.\,\mathtt{T} \quad \phi \vdash s : \mathtt{I}}{\phi;\Gamma \vdash t(s) : \mathtt{T}[i/s]}$$

$$\frac{\phi;\Gamma \vdash f : \mathtt{N} \to \mathtt{T} \quad \phi \vdash t : \mathtt{N}}{\phi;\Gamma \vdash \mathtt{vec}(t)(f) : \mathtt{Vec}_t(\mathtt{T})} \qquad \frac{\phi;\Gamma \vdash v : \mathtt{Vec}_t(\mathtt{T}) \quad \phi;\Gamma \vdash i : \mathtt{N}}{\phi;\Gamma \vdash \mathtt{get}(t)(i,v) : \mathtt{T}}$$

$$\frac{\phi;\Gamma \vdash f : \mathtt{T} \times \mathtt{T} \to \mathtt{T} \quad \phi;\Gamma \vdash t : \mathtt{T} \quad \phi;\Gamma \vdash v : \mathtt{Vec}_s(\mathtt{T})}{\phi;\Gamma \vdash \mathtt{reduce}(s)(f,t,v) : \mathtt{T}}$$

$$\frac{\phi;\Gamma \vdash f : \mathtt{N} \times \mathtt{N} \to \mathtt{T} \quad \phi \vdash s : \mathtt{N} \quad \phi \vdash t : \mathtt{N}}{\phi;\Gamma \vdash \mathtt{mat}(s,t)(f) : \mathtt{Mat}_{s\times t}(\mathtt{T})} \qquad \frac{\phi;\Gamma \vdash M : \mathtt{Mat}_{s\times t}(\mathtt{T}) \quad \phi;\Gamma \vdash i,j : \mathtt{N}}{\phi;\Gamma \vdash \mathtt{get}(s,t)(i,j,M) : \mathtt{T}}$$

(c) Typing rules for operations involving dependent types

Figure 4.3: Typing rules

$$\frac{(s,N)\to m \quad (t(1),N)\to v_1 \quad \ldots \quad (t(m),N)\to v_m}{(\mathtt{vec}(s)(t),N)\to (v_1,\ldots,v_m)}$$

$$\frac{(s,N)\to m \quad (t,N)\to i \quad (v,N)\to (v_1,\ldots,v_n)}{(\mathtt{get}(s)(t,v),N)\to v_i}\ 1\le i\le m$$

$$\frac{\begin{array}{lll}(s,N)\to m & (t,N)\to w_0 & (t',N)\to (v_1,\ldots,v_m)\\ (s'(w_0,v_1),N)\to w_1 & (s'(w_1,v_2),N)\to w_2 \quad \cdots & (s'(w_{m-1},v_m),N)\to w_m\end{array}}{(\mathtt{reduce}(s)(s',t,t'),N)\to w_m}$$

$$\frac{(s,N)\to m \quad (s',N)\to n \quad (t(1,1),N)\to v_{1,1} \quad \ldots \quad (t(m,n),N)\to v_{m,n}}{(\mathtt{mat}(s,s')(t),N)\to ((v_{1,1},\ldots,v_{1,n}),\ldots,(v_{m,1},\ldots,v_{m,n}))}$$

$$\frac{\begin{array}{ll}(s,N)\to m & (t,N)\to i\\ (s',N)\to n & (t',N)\to j\end{array} \quad (M,N)\to ((v_{1,1},\ldots,v_{1,n}),\ldots,(v_{m,1},\ldots,v_{m,n}))}{(\mathtt{get}(s,s')(t,t',M),N)\to v_{i,j}}\ \dagger$$

† Note the side condition $1\le i\le m, 1\le j\le n$.

Figure 4.4: Operational semantics of vector and matrix operations

terms of weak convergence, we must provide our language with a semantics in a category of topological spaces and continuous maps, as in chapter 3. As we have λ-abstractions, this semantics also needs a Cartesian closed structure; as in chapter 3, we choose the category **CG** of compactly-generated spaces (CG-spaces) [36]. This category is Cartesian closed, complete, and co-complete, and can in particular be used to interpret the coinductive types $\Sigma\,\mathtt{T}$.

However, **CG** is *not* locally Cartesian closed [50], and as a result, it cannot interpret a full dependently typed language [51]. Despite this, **CG** is rich enough to interpret a language with only a restricted class of dependent types. The addition of even a small collection of dependent types, though, will change the semantics of our sampling language quite considerably. Following [51], we now interpret judgements $\phi;\Gamma\vdash t:\mathtt{T}$ as morphisms in the comma category $\mathbf{CG}\downarrow[\![\phi]\!]$.

Indexing types, contexts and terms.

Indexing types. Indexing types are interpreted as objects in **CG**. The ground types `B` and `N` are interpreted as the sets 2 and $\mathbb{N}$ equipped with the

discrete topology, and the types $\mathtt{R}, \mathtt{R}^+$ are interpreted as $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ equipped with their usual topology. Product types are interpreted by the product in **CG** (which is not always the usual product of topological spaces, although it is for all the indexing types we allow [36]).

Indexing contexts. Indexing contexts are inductively interpreted in the usual way: $[\![\emptyset]\!] \triangleq 1 = \{*\}$, the terminal object in **CG**, and $[\![\phi, i : \mathtt{I}]\!] \triangleq [\![\phi]\!] \times [\![\mathtt{I}]\!]$, where the product is again taken in **CG**.

Well-formed indexing terms. Judgements $\phi \vdash t : \mathtt{I}$ derivable from fig. 4.2a are inductively interpreted as **CG**-morphisms $[\![t]\!] : [\![\phi]\!] \to [\![\mathtt{I}]\!]$ in the usual way.

Well-formed types.

A judgment $\phi \vdash \mathtt{T} : \mathtt{Type}$ will be interpreted as an object in $\mathbf{CG} \downarrow [\![\phi]\!]$. The underlying **CG**-morphism $[\![\mathtt{T}]\!] \to [\![\phi]\!]$ defining such an object will be referred to as the **indexing map**. Define $U : \mathbf{CG} \downarrow [\![\phi]\!] \to \mathbf{CG}$ as the forgetful functor $U([\![\mathtt{T}]\!] \to [\![\phi]\!]) = [\![\mathtt{T}]\!]$.

Ground types. Since $[\![\emptyset]\!] = 1$, $[\![\emptyset \vdash \mathtt{G} : \mathtt{Type}]\!]$ is interpreted as an object in $\mathbf{CG} \downarrow 1$, which is equivalent to **CG**. Therefore, we can simply use the interpretation of ground types defined above.

Index weakening. To interpret the index weakening rule of fig. 4.2b, we need to define the notion of **change of base functor**. Given two **CG**-objects A, B and a **CG**-morphism $f : A \to B$, there exists a functor $f^* : \mathbf{CG}{\downarrow}B \to \mathbf{CG}{\downarrow}A$ called the change of base functor which is defined by taking pullbacks along f. In particular, the projection $\pi_\phi : [\![\phi]\!] \times [\![\mathtt{I}]\!] \to [\![\phi]\!]$ defines a change of base functor $\pi_\phi^* : \mathbf{CG}{\downarrow}[\![\phi]\!] \to \mathbf{CG}{\downarrow}[\![\phi, i : \mathtt{I}]\!]$. We can now interpret index weakening by associating with $[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ the $\mathbf{CG} \downarrow [\![\phi, i : \mathtt{I}]\!]$-object

$$[\![\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}]\!] \triangleq \pi_\phi^* \left([\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \right).$$

Remark 4.1.1. *This rule shows that the interpretation of a type* $\mathtt{T}$ *changes*

with the indexing context. In particular, we have:

$$U\,[\![\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}]\!] \simeq U\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \times [\![\mathtt{I}]\!] \;\; \textit{when } i \notin FV(\mathtt{T}).$$

Products. Given two $\mathbf{CG} \downarrow [\![\phi]\!]$-objects $[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!]$ and $[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$, $[\![\phi \vdash \mathtt{S} \times \mathtt{T} : \mathtt{Type}]\!]$ is their product in $\mathbf{CG} \downarrow [\![\phi]\!]$, viz. the $\mathbf{CG}$-pullback

$$\begin{array}{ccc} U\,[\![\phi \vdash \mathtt{S} \times \mathtt{T} : \mathtt{Type}]\!] & \longrightarrow & U\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \\ \downarrow & \lrcorner & \downarrow \\ U\,[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!] & \longrightarrow & [\![\phi]\!] \end{array}$$

For example, $[\![n : \mathtt{N} \vdash \mathtt{Vec}_n(\mathtt{R}) \times \mathtt{Vec}_{2n}(\mathtt{R})]\!]$ is the space of pairs of vectors of lengths $(n, 2n), n \in \mathbb{N}$, sent to the same n by the indexing map by commutativity of the pullback square.

Coproducts. Colimits in $\mathbf{CG} \downarrow A$ are inherited from colimits in $\mathbf{CG}$: given a diagram $\mathscr{D} : J \to \mathbf{CG} \downarrow A$, $\operatorname{colim} \mathscr{D}$ is constructed by first constructing $\operatorname{colim}(U \circ \mathscr{D})$. Since A is, by definition, a cocone for this colimit, there exists a unique morphism $\operatorname{colim}(U \circ \mathscr{D}) \to A$ which defines $\operatorname{colim} \mathscr{D}$. For coproducts, if $[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!]$ and $[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ are $\mathbf{CG}\downarrow[\![\phi]\!]$-objects, then $[\![\phi \vdash \mathtt{S} + \mathtt{T} : \mathtt{Type}]\!]$ is given by the copairing $U\,[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!] + U\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \to [\![\phi]\!]$ in $\mathbf{CG}$.

Function types. As mentioned above, $\mathbf{CG}$ is *not* locally Cartesian closed [50], i.e. $\mathbf{CG} \downarrow A$ is in general not Cartesian closed. However, $\mathbf{CG} \downarrow A$ is Cartesian closed for sufficiently nice objects A, and these objects include the denotation of all indexing contexts that can arise in our language.

Theorem 4.1.2 ([52, 53])**.** *If A is (weak) Hausdorff then the category $\mathbf{CG}\downarrow A$ is Cartesian closed.*

Since 2, $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ are Hausdorff, and since the product of Hausdorff spaces is Hausdorff, theorem 4.1.2 allows us to define for every pair of $\mathbf{CG} \downarrow [\![\phi]\!]$-objects $[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!], [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ the object $[\![\phi \vdash \mathtt{S} \to \mathtt{T} : \mathtt{Type}]\!]$ as their internal hom in $\mathbf{CG} \downarrow [\![\phi]\!]$. This object is quite subtle [52, 1.3.11], and can informally be described as the space of partial maps from $U\,[\![\phi \vdash \mathtt{S} : \mathtt{Type}]\!]$ to $U\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ mapping a fibre indexed by $x \in [\![\phi]\!]$ in the domain

to the fibre in the codomain indexed by the same x. The indexing map $U\,[\![\phi \vdash \mathtt{S} \to \mathtt{T} : \mathtt{Type}]\!] \to [\![\phi]\!]$ naturally sends such a map to $x \in [\![\phi]\!]$. See *op. cit.* for information about how this space is defined and topologised. As an example, $[\![m : \mathtt{N}, n : \mathtt{N} \vdash \mathtt{Mat}_{m\times n}(\mathtt{R}) \to \mathtt{Vec}_{2m}(\mathtt{R})]\!]$ contains the continuous maps from the space of all real $m \times n$-matrices to that of all real $2m$-dimensional vectors – for a fixed choice $m, n \in \mathbb{N}^2$. Such a map is not defined on $m' \times n'$-matrices if $m \neq m'$ or $n \neq n'$ and is sent to (m, n) by the indexing map.

Sampler types. Let $[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ be a $\mathbf{CG} \downarrow [\![\phi]\!]$-object. We define $[\![\phi \vdash \Sigma\,\mathtt{T} : \mathtt{Type}]\!]$ as the terminal coalgebra for an endofunctor on $\mathbf{CG} \downarrow [\![\phi]\!]$, in analogy to how we had defined it in chapter 3. Define the functor

$$F^{\mathtt{T}}(A) \triangleq [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \times \left[\!\left[\phi \vdash \mathtt{R}^+ : \mathtt{Type}\right]\!\right] \times A$$

where the product above is taken in $\mathbf{CG}\downarrow[\![\phi]\!]$ (and is thus a pullback in $\mathbf{CG}$).

Note that $\mathbf{CG} \downarrow [\![\phi]\!]$ has as terminal object $\mathrm{id}_{[\![\phi]\!]} : [\![\phi]\!] \to [\![\phi]\!]$, and is complete because $\mathbf{CG}$ is. Moreover, since the functor $F^{\mathtt{T}}$ is defined by a product, and limits commute with limits, $F^{\mathtt{T}}$ preserves all limits. We can therefore define $[\![\phi \vdash \Sigma\,\mathtt{T} : \mathtt{Type}]\!] \triangleq \nu F^{\mathtt{T}}$, the terminal coalgebra for $F^{\mathtt{T}}$, by using the terminal sequence construction of theorem 3.1.6, just as in the previous chapter. The carrier of $[\![\Sigma\,\mathtt{T}]\!]$ is then given by the limit (in $\mathbf{CG}$) of the following diagram, which guarantees that all the components of streams are indexed consistently.

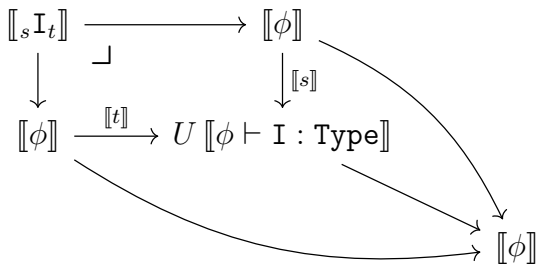
$$\begin{array}{ccccccc} [\![\phi]\!] & \longleftarrow & UF^{\mathtt{T}}1 & \longleftarrow & UF^{\mathtt{T}}(F^{\mathtt{T}}1) & \longleftarrow & \dots \\ & \searrow & \downarrow & \swarrow & & & \\ & & [\![\phi]\!] & & & & \end{array}$$

For example, $\Sigma\,\mathtt{Vec}_n(\mathtt{R})$ contains the streams of weighted vectors of identical lengths; a sampler in $\Sigma\,\mathtt{Vec}_n(\mathtt{R})$ cannot return vectors of different lengths because they are not in the limit.

Example 4.1.3. *As discussed in the introduction to this chapter, choosing to consider stochastic process samplers as programs returning sample paths of*

type $\mathtt{T} \to \mathtt{S}$ *leads to programs which cannot be computationally realised. Nevertheless the type of such programs is available to us as* $\phi \vdash \Sigma\,(\mathtt{T} \to \mathtt{S}) : \mathtt{Type}$. *Elements in the carrier of* $[\![\phi \vdash \Sigma\,(\mathtt{T} \to \mathtt{S}) : \mathtt{Type}]\!]$ *are weighted sequences of continuous functions (sample paths) that share the same domain and codomain fibres indexed by some* $x \in [\![\phi]\!]$.

Pullback types. As their name suggests, the semantics of pullback types is straightforwardly given by the pullbacks



Vector and matrix types. Consider $[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ in $\mathbf{CG}{\downarrow}[\![\phi]\!]$. We interpret the vector type $\mathtt{Vec}_s(\mathtt{T})$; the case of matrices is treated in a similar way. An indexing judgement $\phi \vdash t : \mathtt{N}$ is interpreted as a $\mathbf{CG}$-morphism $[\![t]\!] : [\![\phi]\!] \to \mathbb{N}$. To define $[\![\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}]\!]$, we note that any element $x \in [\![\phi]\!]$ defines a morphism $x : 1 \to [\![\phi]\!]$ and therefore a change of base functor $x^* : \mathbf{CG}\downarrow[\![\phi]\!] \to \mathbf{CG}\downarrow 1$. With this definition, and using the cocompleteness of $\mathbf{CG}$, we define

$$[\![\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}]\!] : \coprod_{x \in [\![\phi]\!]} (x^*\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!])^{[\![t]\!](x)} \to [\![\phi]\!]\,,\, \mathrm{in}_x(y) \mapsto x$$

The products and coproduct are taken in $\mathbf{CG}\downarrow 1$, i.e. in $\mathbf{CG}$, so the above defines an object in $\mathbf{CG}\downarrow[\![\phi]\!]$. Intuitively, x^* substitutes the variables in ϕ with the concrete values provided by x in the dependent type $\mathtt{T}$, i.e. constructs '$\mathtt{T}[\phi/x]$'.

As an example of this complex-seeming definition, consider the type

$$[\![x : \mathtt{N}, y : \mathtt{N} \vdash \mathtt{Vec}_{2y}(\mathtt{Mat}_{x \times y}(\mathtt{R}))]\!]\,.$$

For each $(m,n) \in \mathbb{N}^2$, we have $(m,n)^*(\llbracket \mathtt{Mat}_{x\times y}(\mathtt{R}) \rrbracket) = \mathbb{R}^{m\times n}$, and therefore

$$\llbracket x : \mathtt{N}, y : \mathtt{N} \vdash \mathtt{Vec}_{2y}(\mathtt{Mat}_{x\times y}(\mathtt{R})) \rrbracket = \coprod_{(m,n)\in\mathbb{N}^2} \left(\mathbb{R}^{m\times n}\right)^{2n}$$

with the indexing map sending $2n$-tuples of $m \times n$ matrices to (m,n).

Π-types. Consider a type $\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}$ interpreted as an object in $\mathbf{CG} \downarrow \llbracket \phi, i : \mathtt{I} \rrbracket$. The projection map $\pi_\phi : \llbracket \phi \rrbracket \times \llbracket \mathtt{I} \rrbracket \to \llbracket \phi \rrbracket$ defines a change of base functor $\pi_\phi^* : \mathbf{CG} \downarrow \llbracket \phi \rrbracket \to \mathbf{CG} \downarrow \llbracket \phi, i : \mathtt{I} \rrbracket$. Under the same assumptions as theorem 4.1.2, namely that the object defining the comma category is Hausdorff, it can be shown that this functor has a right adjoint $\Pi_{\pi_\phi} : \mathbf{CG} \downarrow \llbracket \phi, i : \mathtt{I} \rrbracket \to \mathbf{CG} \downarrow \llbracket \phi \rrbracket$. We then define

$$\llbracket \phi \vdash \Pi\, i : \mathtt{I}\,.\,\mathtt{T} \rrbracket \triangleq \Pi_{\pi_\phi} \llbracket \phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type} \rrbracket .$$

Example 4.1.4. *A particularly important Π-type for our purpose is the type of samplers on $\llbracket \mathtt{T} \rrbracket$-indexed $\llbracket \mathtt{S} \rrbracket$-valued stochastic processes*

$$\mathtt{Marginal}(\mathtt{T},\mathtt{S}) \triangleq \Pi\, n : \mathtt{N}\,.\,\mathtt{Vec}_n(\mathtt{T}) \to \Sigma\,(\mathtt{Vec}_n(\mathtt{S})).$$

whose development was motivated in the introduction to this chapter. Assuming for simplicity's sake that S *and* T *are closed types, this type is interpreted as*

$$\begin{aligned}\llbracket \vdash \mathtt{Marginal}(\mathtt{T},\mathtt{S}) : \mathtt{Type} \rrbracket &= \Pi_{!:\mathbb{N}\to 1}\left(\llbracket i : \mathtt{N} \vdash \mathtt{Vec}_i(\mathtt{T}) \to \Sigma\,(\mathtt{Vec}_i(\mathtt{S})) \rrbracket\right) \\ &= \prod_{n\in\mathbb{N}} \{ f : \llbracket \mathtt{T} \rrbracket^n \to (\llbracket \mathtt{S}^n \rrbracket \times \mathbb{R}_{\geq 0})^\omega \textit{ continuous} \}\end{aligned}$$

An element of $U\,\llbracket \vdash \mathtt{Marginal}(\mathtt{T},\mathtt{S}) : \mathtt{Type} \rrbracket$ is thus a sequence of functions $(f_n)_{n\in\mathbb{N}}$, each of which associates to any tuple $(t_1,\ldots,t_n)$ a sampler on n copies of S*.*

Remark 4.1.5. *The reader may be wondering at this point: why not instead introduce stochastic process samplers to the language as functional samplers of*

type $s : \Sigma(\mathtt{T} \to \mathtt{S})$*, each sample of which is a full trajectory, i.e. a function* $f : \mathtt{T} \to \mathtt{S}$*? In short, this is conceptually possible, subject to concerns about continuity, but less practical. Regarding continuity, recall that our language limits us to considering continuous functions* $f : \mathtt{T} \to \mathtt{S}$*, which means that the only stochastic processes that will type are those whose sample paths are all continuous.*

Note that these functional samplers have a natural relationship to our marginal samplers: clearly, given a functional sampler $s : \Sigma(\mathtt{T} \to \mathtt{S})$*, we can construct its corresponding finite-dimensional variant:*

```
λn : N . λt : Vec_n(T) .
  map(λf : T → S .
    vec(n)(λi : N . f(get(n)(i, t))),
    s)
  : Marginal(T,S)
```

Listing 4.1: Projection from functional to marginal samplers

However, there is no clear procedure by which we can do the reverse on uncountably large index sets $\mathtt{T}$*, i.e. use a system of marginal samplers to construct a functional sampler. While functional samplers for those processes are perfectly well-defined, nontrivial samplers targeting such processes cannot be produced by any practical means. Consider the Wiener process: preparing a full, single trajectory* $f : \mathbb{R} \to \mathbb{R}$ *would require specifying its value at uncountably many inputs.*

Subtyping. Just as in chapter 3, the subtyping relation $\triangleleft$ defined by the rules of fig. 4.2c is meant to carry topological information, and will always be interpreted as an identity map. Its formal development proceeds the same as in the previous chapter as well. For every boolean-valued comparison operator $f \in \{<, \leq, >, \geq, =, \neq\}$ the ground type $f^{-1}(0) + f^{-1}(1)$ is interpreted as $[\![f]\!]^{-1}(0) + [\![f]\!]^{-1}(1)$ where $+$ is the coproduct in **CG**, which is a CG-space by Proposition 3.1.5. Recall that $\mathtt{S}_1 \cap \mathtt{S}_2$ is syntactic sugar for $\mathtt{cast}\langle \mathtt{T} \rangle x_1 : \mathtt{S}_1\, \mathtt{T}\, \mathtt{cast}\langle \mathtt{T} \rangle x_2 : \mathtt{S}_2$, i.e. $[\![\mathtt{S}_1 \cap \mathtt{S}_2]\!]$ has the same carrier as $[\![\mathtt{S}_1]\!]$ and $[\![\mathtt{S}_2]\!]$ and the coarsest common refinement of their topologies, making both identity maps $[\![\mathtt{S}_1 \cap \mathtt{S}_2 \triangleleft \mathtt{S}_1]\!]$ and

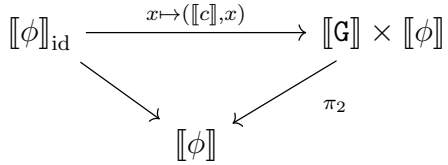
$[\![\mathtt{S}_1 \cap \mathtt{S}_2 \lhd \mathtt{S}_2]\!]$ continuous. And since the subtyping relation is interpreted via identity maps, the interpretation of $\mathtt{S} \lhd \mathtt{T}$ in $\mathbf{CG}$ transfers trivially to $\mathbf{CG}{\downarrow}[\![\phi]\!]$.

Well-formed terms-in-contexts

The semantics of a judgement $\phi; \Gamma \vdash t : \mathtt{T}$ will be given by a $\mathbf{CG}{\downarrow}[\![\phi]\!]$-morphism $[\![\phi; \Gamma \vdash t : \mathtt{T}]\!] : [\![\phi; \Gamma]\!] \to [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$. Apart from sampler operations, all operations will be interpreted using completely standard categorical constructions, albeit in the less-familiar comma categories $\mathbf{CG} \downarrow [\![\phi]\!]$.

Contexts. As can be seen from fig. 3.2a, contexts are constructed using either the variable introduction rule or the weakening rule. In particular, the indexing context ϕ is fixed. We can therefore interpret contexts recursively using the product of $\mathbf{CG} \downarrow [\![\phi]\!]$ in the usual way, viz. $[\![\phi; \emptyset]\!] = 1$, the terminal object in $\mathbf{CG}{\downarrow}[\![\phi]\!]$ given by $\mathrm{id}_{[\![\phi]\!]} : [\![\phi]\!] \to [\![\phi]\!]$, and $[\![\phi; \Gamma, x : \mathtt{T}]\!] = [\![\phi; \Gamma]\!] \times [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$.

Constants, variable introduction and built-in functions. The index weakening rule allows us to interpret the judgement $\phi \vdash \mathtt{G} : \mathtt{Type}$ as $!^*_\phi [\![\emptyset \vdash \mathtt{G} : \mathtt{Type}]\!]$ where $!_\phi : [\![\phi]\!] \to 1$. This interpretation is just the $\mathbf{CG} \downarrow [\![\phi]\!]$-object $[\![\mathtt{G}]\!] \times [\![\phi]\!] \to [\![\phi]\!]$ projecting away $[\![\mathtt{G}]\!]$. With this, we define $[\![\phi; \emptyset \vdash c : \mathtt{G}]\!]$ by



We define $[\![\phi; x : \mathtt{T} \vdash x : \mathtt{T}]\!]$ as the $\mathbf{CG} \downarrow [\![\phi]\!]$-identity $\mathrm{id} : [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!] \to [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$. Finally, each built-in function f is given a $\mathbf{CG}$-interpretation $[\![f]\!] : [\![\mathtt{G}_1]\!] \times \ldots \times [\![\mathtt{G}_n]\!] \to [\![\mathtt{G}]\!]$ which, in turn, defines a $\mathbf{CG}{\downarrow}[\![\phi]\!]$-interpretation of the same formal type. Since products in $\mathbf{CG}{\downarrow}[\![\phi]\!]$ are (wide) pullbacks in $\mathbf{CG}$, input elements in $[\![\mathtt{G}_1]\!] \times \ldots \times [\![\mathtt{G}_n]\!]$ are tuples $((g_1, x), \ldots, (g_n, x))$, $g_i \in [\![\mathtt{G}_i]\!]$ for a common x of $[\![\phi]\!]$. We can thus unambiguously send $((g_1, x), \ldots, (g_n, x))$ to $([\![f]\!](g_1, \ldots, g_n), x)$; this defines the $\mathbf{CG}{\downarrow}[\![\phi]\!]$-interpretation $[\![f]\!]$. With this we can define the usual interpretation

$$[\![\phi; \Gamma \vdash f(t_1, \ldots, t_n) : \mathtt{G}]\!] = [\![f]\!] \circ \langle [\![t_1]\!], \ldots, [\![t_n]\!] \rangle.$$

Standard constructs. Casting, (co)products, (co)projections, let bindings, lambda-abstraction and function application are interpreted in the usual way. The last two operations rely on theorem 4.1.2 and the fact that indexing types are interpreted as Hausdorff spaces.

Π-type abstraction and evaluation. The map $\pi_\phi : [\![\phi]\!] \times [\![\mathtt{I}]\!] \to [\![\phi]\!]$ defines the change of base functor $\pi_\phi^* : \mathbf{CG}\downarrow[\![\phi]\!] \to \mathbf{CG}\downarrow[\![\phi; i : \mathtt{I}]\!]$ and its right adjoint $\Pi_{\pi_\phi} : \mathbf{CG}\downarrow[\![\phi; i : \mathtt{I}]\!] \to \mathbf{CG}\downarrow[\![\phi]\!]$. Using this functor we define

$$[\![\phi; \Gamma \vdash (\lambda i : \mathtt{I}\,.\,t) : (\Pi\, i : \mathtt{I}\,.\,\mathtt{T})]\!] = \Pi_{\pi_\phi} [\![\phi, i : \mathtt{I}; \Gamma \vdash t : \mathtt{T}]\!]$$

For the Π-type evaluation rule, since the premise has a Π-type, it must be that $[\![\phi; \Gamma \vdash t : \Pi\, i : \mathtt{I}\,.\,\mathtt{T}]\!]$ has the shape $\Pi_{\pi_\phi} [\![\phi, i : \mathtt{I}; \Gamma \vdash t' : \mathtt{T}]\!]$ for some term $t' : \mathtt{T}$. In other words,

$$[\![t]\!] \in \hom_{\mathbf{CG}\downarrow[\![\phi]\!]} \left([\![\phi; \Gamma]\!] , \Pi_{\pi_\phi} [\![\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}]\!] \right).$$

By the adjunction, this corresponds to a unique $\mathbf{CG}\downarrow[\![\phi, i : \mathtt{I}]\!]$-arrow

$$\widehat{[\![t]\!]} \in \hom_{\mathbf{CG}\downarrow[\![\phi, i:\mathtt{I}]\!]} \left(\pi_\phi^* [\![\phi; \Gamma]\!] , [\![\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}]\!] \right).$$

The index term $\phi \vdash s : \mathtt{I}$ defines $\langle \mathrm{id}, [\![s]\!] \rangle : [\![\phi]\!] \to [\![\phi]\!] \times [\![\mathtt{I}]\!]$ which provides a concrete value of type $\mathtt{I}$ and satisfies $\pi_\phi \circ \langle \mathrm{id}, [\![s]\!] \rangle = \mathrm{id}$. Using this map we define

$$[\![\phi; \Gamma \vdash t(s) : \mathtt{T}[i/s]]\!] \triangleq \langle \mathrm{id}, [\![s]\!] \rangle^* \widehat{[\![t]\!]} : [\![\phi; \Gamma]\!] \to \langle \mathrm{id}, [\![s]\!] \rangle^* [\![\phi, i : \mathtt{I} \vdash \mathtt{T} : \mathtt{Type}]\!]$$

Intuitively, $\langle \mathrm{id}, [\![s]\!] \rangle^*$ substitutes the free variable i in $\mathtt{T}$ with the concrete value computed by s and leaves all other free variables unchanged, as suggested by the notation $\mathtt{T}[i/s]$.

Vector and matrix operations. The semantics is intuitively simple: given $f : \mathbb{N} \to A$, $\mathtt{vec}(n)(f)$ builds the n-dimensional vector with entries in A given by $(f(1), \ldots, f(n))$. Matrices are defined analogously: given a map

$f : \mathbb{N} \times \mathbb{N} \to A$, the matrix $\mathtt{mat}(m,n)(f)$ is given by $(f(i,j))_{1\leq i\leq m, 1\leq j\leq n}$. The difficulty is in showing that these simple constructions are compatible with the indexed structure of the semantics. We only consider the case of vectors, as the case of matrices is treated in the same way. Assuming

$$[\![f]\!] : [\![\phi; \Gamma]\!] \to [\![\phi \vdash \mathtt{N} \to \mathtt{T} : \mathtt{Type}]\!] \qquad \text{and} \qquad [\![t]\!] : [\![\phi]\!] \to \mathbb{N}$$

in $\mathbf{CG} \downarrow [\![\phi]\!]$, we define

$$[\![\mathtt{vec}(t)(f)]\!] : [\![\phi; \Gamma]\!] \to [\![\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}]\!]$$

via the **CG**-triangle

$$\begin{array}{ccc} U\,[\![\phi;\Gamma]\!]_g & \xrightarrow{U[\![\mathtt{vec}(t)(f)]\!]} & U\,[\![\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}]\!] \\ & \searrow \quad \swarrow_{\mathrm{in}_x(y)\mapsto x} & \\ & [\![\phi]\!] & \end{array}$$

where $U\,[\![\mathtt{vec}(t)(f)]\!]\,(\gamma) \triangleq \Big(U\,[\![f]\!]\,(\gamma)(1), \ldots, U\,[\![f]\!]\,(\gamma)\big([\![t]\!](g(\gamma))\big)\Big)$.

Lemma 4.1.6. *The triangle above commutes, i.e. the semantics of* $\mathtt{vec}(t)(f)$ *is well-defined.*

Proof. By the definition of the internal hom in $\mathbf{CG} \downarrow [\![\phi]\!]$, $[\![f]\!]\,(\gamma)$ takes a function from the $g(\gamma)$-indexed copy of $\mathbb{N}$ in the domain to the fibre in $U\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$ indexed by $g(\gamma)$. It follows that $U\,[\![\mathtt{vec}(t)(f)]\!]\,(\gamma) \triangleq \Big(U\,[\![f]\!]\,(\gamma)(1), \ldots, U\,[\![f]\!]\,(\gamma)\big([\![t]\!](g(\gamma))\big)\Big)$ belongs to the summand

$$\big(g(\gamma)^*\,[\![\phi, n : \mathtt{N} \vdash \mathtt{T} : \mathtt{Type}]\!]\big)^{[\![t]\!](g(\gamma))}$$

in the coproduct defining $[\![\phi \vdash \mathtt{Vec}_t(\mathtt{T}) : \mathtt{Type}]\!]$, and thus gets indexed by $g(\gamma)$ as desired. □

The operation $\mathtt{get}(t)(i,v)$ deconstructs a vector, choosing its ith element; its output is not indexed. Given

$$[\![v]\!] : [\![\phi; \Gamma]\!] \to [\![\phi \vdash \mathtt{N} \to \mathtt{T} : \mathtt{Type}]\!], \qquad [\![i]\!] : [\![\phi; \Gamma]\!] \to \mathbb{N}, \qquad [\![t]\!] : [\![\phi]\!] \to \mathbb{N},$$

having suppressed the forgetful functor U, define

$$[\![\mathtt{get}(t)(i,v)]\!] : [\![\phi;\Gamma]\!] \to [\![1 \vdash \mathtt{T} : \mathtt{Type}]\!]$$

as simply $[\![\mathtt{get}(t)(i,v)]\!] = [\![v]\!](\gamma)([\![t]\!](\gamma))_{[\![i]\!](\gamma)}$, i.e. choosing the ith element of the vector v, assuming the choice is valid.

The operation `reduce` is also straightforwardly defined. It applies a binary operation $f : \mathtt{T} \times \mathtt{T} \to \mathtt{T}$ to a vector $v : \mathtt{Vec}_t(\mathtt{T})$, starting from the initial value $s : \mathtt{T}$, until all that remains is one value $\mathtt{reduce}(t)(f,v,s)$ of type $\mathtt{T}$. And so, given

$$[\![v]\!] : [\![\phi;\Gamma]\!] \to [\![\phi \vdash \mathtt{N} \to \mathtt{T} : \mathtt{Type}]\!], \qquad [\![f]\!] : [\![\phi;\Gamma]\!] \to [\![\phi \vdash \mathtt{T} \times \mathtt{T} \to \mathtt{T} : \mathtt{Type}]\!],$$
$$[\![s]\!] : [\![\Phi;\Gamma]\!] \to [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!], \qquad [\![t]\!] : [\![\Phi;\Gamma]\!] \to \mathbb{N}$$

define

$$[\![\mathtt{reduce}(t)(f,v,s)]\!] : [\![\phi;\Gamma]\!] \to [\![1 \vdash \mathtt{T} : \mathtt{Type}]\!]$$

as

$$[\![\mathtt{reduce}(t)(f,v,s)]\!](\gamma) = [\![f]\!](\gamma)([\![v]\!](\gamma)([\![t]\!](\gamma)), \ldots, [\![f]\!](\gamma)([\![v]\!](\gamma)(1), [\![s]\!](\gamma)))$$

For example, in the case that $[\![f]\!](\gamma)(x,y) = x + y$, $[\![t]\!](\gamma) = 3$, $[\![s]\!](\gamma) = v_0$, and $[\![v]\!](\gamma) = (v_1, \ldots, v_3)$, $[\![\mathtt{reduce}(t)(f,v,s)]\!] = v_3 + (v_2 + (v_1 + v_0))$.

Sampler operations. Remember that sampler operations were defined purely coinductively in chapter 3. Since judgements $[\![\phi \vdash \Sigma\,\mathtt{T} : \mathtt{Type}]\!]$ are analogously defined as terminal coalgebras in $\mathbf{CG}{\downarrow}[\![\phi]\!]$, all the definitions of *op. cit.* transfer immediately to our indexed semantics. For example, $[\![\phi;\Gamma \vdash \mathtt{hd}(t) : \mathtt{T}]\!]$ is defined from the usual `head` operation ($F^{\mathtt{T}}!$), then projecting away the weight

$$[\![\phi;\Gamma \vdash t : \Sigma\,\mathtt{T}]\!] \xrightarrow{F^{\mathtt{T}}!} F^{\mathtt{T}}1 \xrightarrow{\pi_1} [\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!].$$

4.2 Sampler equivalence

Figure 4.5 extends the equivalence calculus of section 3.2 with two additional rules pertaining to our new built-in vector and matrix operations, whose proof (whether in the operational or denotational semantics) is immediate. These two simple rules are the only additional rules we will need in order to justify the results in section 4.3.

Note that all of the proofs of our equivalence calculus given in section 3.2 extend immediately to this setting as well. These proofs, aside from those that are trivially true in our denotational semantics, worked by leveraging the coinductive definitions of our sampler operations. While the addition of dependent types has added additional structure to our denotational semantics of terms and types, our coinductive definitions of sampler operations are in fact unchanged, and so the purely coinductive proofs of section 3.2 carry over with no changes.

$$\phi\ ;\ \Gamma \vdash \mathtt{T} \vdash \mathtt{get}(n)(i, \mathtt{vec}(n)(f)) \approx f(i) : \mathtt{T}$$
$$\phi\ ;\ \Gamma \vdash \mathtt{get}(m,n)(i,j, \mathtt{mat}(m,n)(f)) \approx f(i,j) : \mathtt{T}$$

Figure 4.5: Equivalence rules for vector and matrix operations

The `get`-reduction rules of fig. 4.5 follow immediately from the denotational semantics of `vec` and `get` given in section 4.1; it can also be easily demonstrated in the operational semantics.

Proof of soundness for fig. 4.5.

$$\begin{aligned}
[\![\mathtt{get}(n)(i, \mathtt{vec}(n)(f))]\!]\,(\gamma) &= [\![\mathtt{get}(n)(i, ([\![f]\!]\,(\gamma)(1), \ldots, [\![f]\!]\,(\gamma)([\![n]\!]\,(\gamma))))]\!] \\
&= ([\![f]\!]\,(\gamma)(1), \ldots, [\![f]\!]\,(\gamma)([\![n]\!]\,(\gamma)))_{[\![i]\!](\gamma)} \\
&= [\![f]\!]\,(\gamma)([\![i]\!]\,(\gamma))
\end{aligned}$$

assuming that the selector $[\![i]\!]\,(\gamma)$ is valid. The same argument holds for `get` over `mat`. □

```
λn : N .
  λ(v1, v2) : Vec_n(R) × Vec_n(R) .
    vec(n)(λi : N . get(n)(v1, i) + get(n)(v2, i))
: Π n : N . Vec_n(R) × Vec_n(R) → Vec_n(R)
```

Listing 4.2: Definition of `vadd`

```
λn : N .
  λv : Vec_n(R) .
    reduce(n)(λ(x, y) : R × R . x + y, 0, v)
: Π n : N . Vec_n(R) → R
```

Listing 4.3: Definition of `vsum`

4.3 Verification

In this section, we apply the language we developed in section 4.1 to demonstrate the soundness of a number of techniques which are commonly used in the literature for constructing and transforming stochastic processes. In particular, we will identify sets of programs which provably construct stochastic process samplers, and which provably transform one stochastic processes sampler into another.

4.3.1 Vector and matrix operations

In this section, we include implementations of several vector and matrix operations used within the example programs we will consider in the coming sections. As the purpose of this chapter is to discuss the verification of samplers for stochastic processes, rather than to to discuss the verification of linear algebra and list operations, we restrict attention to only a few simple, mostly self-explanatory operations that are necessary for our chosen examples.

The purpose of each of these programs should be self-explanatory. `vadd` (implemented in listing 4.2) adds two vectors of the same size elementwise; `vsum` (implemented in listing 4.3) sums the elements of one vector; `vprod` (implemented in listing 4.4) multiplies two vectors of the same size elementwise; and so on.

```
λn : N .
  λ(v1, v2) : Vec_n(R) × Vec_n(R) .
    vec(n)(λi : N . get(n)(v1, i)*get(n)(v2, i))
: Π n : N . Vec_n(R) × Vec_n(R) → Vec_n(R)
```

Listing 4.4: Definition of `vprod`

```
λn : N .
  λ(v1, v2) : Vec_n(R) × Vec_n(R) .
    vsum(n)(vprod(n)(v1,v2))
: Π n : N . Vec_n(R) × Vec_n(R) → R
```

Listing 4.5: Definition of `dot`

```
λn : N .
  λ(v, x) : Vec_n(T) × T .
    vec(n + 1)(λi : N .
      if i = n + 1 then
        x
      else
        get(n)(i, v))
: Π n : N . Vec_n(T) × T → Vec_{n+1}(T)
```

Listing 4.6: Definition of `cons`

```
λ(m, n) : N × N .
  λv : Vec_m(T) .
    vec(n)(λi : N . get(m)(i, v))
: Π (m, n) : N^2 . Vec_m(T) → Vec_n(T)
```

Listing 4.7: Definition of `take`

```
λn : N . λ(i, M) : N × Mat_{m×n}(T) .
    vec(n)(λj : N . get(m, n)(i, j, M))
: Π (m, n) : N^2 . Mat_{m×n}(T) → Vec_n(T)
```

Listing 4.8: Definition of `row`

```
λn : N . λ(j, M) : N × Mat_{m×n}(T) .
    vec(n)(λi : N . get(m, n)(i, j, M))
: Π (m, n) : N^2 . Mat_{m×n}(T) → Vec_m(T)
```

Listing 4.9: Definition of `col`

```
λ(m, n) : N × N . λ(M, v) : Mat_{m×n}(R) × Vec_n(R) .
    vec(m)(λi : N . dot(n)(row(m, n)(i, M), v))
: Π (m, n) : N^2 . Mat_{m×n}(R) × Vec_n(R) → Vec_m(R)
```

Listing 4.10: Definition of `matvec`

4.3.2 Targeting for dependent samplers

We first recall how standard (finite-dimensional) samplers can be shown to produce samples which asymptotically behaves as if they were drawn from a given distribution. As in chapter 3, we formalise this idea using weak convergence. Let X be a topological space; a sequence of weighted samples $(x_n, w_n)_{n\in\mathbb{N}}$ converges weakly to the probability measure P on X if, for all continuous bounded functions $f : X \to \mathbb{R}$, we have $\lim_{N\to\infty} \frac{\sum_{n=1}^{N} w_n f(x_n)}{\sum_{n=1}^{N} w_n} = \int_X f(x)\, dP(x)$.

Let $\mathcal{P} : \mathbf{Top} \to \mathbf{Top}$ be the functor defined by $\mathcal{P}X =$ probability measures on the Borel sets of X equipped with the topology of weak convergence and $\mathcal{P}(f : X \to Y) = f_* : \mathcal{P}X \to \mathcal{P}Y$, the pushforward operation $P \mapsto f_*P$ (we do not know if $\mathcal{P}X$ is compactly generated when X is, so we define $\mathcal{P}$ on **Top** rather than **CG**). Next, define $\mathcal{P}_\perp = \mathcal{P} + 1$, the coproduct of $\mathcal{P}$ with the singleton space $\{\perp\}$ which will represent divergence. Finally, we define the **empirical transformation** as the collection of maps $\varepsilon_X : (X \times \mathbb{R}_{\geq 0})^{\mathbb{N}} \to \mathcal{P}_\perp X$ each of which takes a weighted sequence of points to its limit in the weak topology if it exists, and to $\perp$ if it doesn't. As shown by Example 3.3.2, this transformation is not natural.

We now recall the definition of the *targeting relation* between samplers and probability measures from the previous chapter; we will then adapt it to our dependently-typed setting. In section 3.1.3, the denotational semantics of a sampler-in-context $\Gamma \vdash s : \Sigma\, \mathtt{S}$ was given by a **CG**-morphism $[\![s]\!] : [\![\Gamma]\!] \to ([\![\mathtt{S}]\!] \times \mathbb{R}_{\geq 0})^\omega$ and the sampler s **targets a $[\![\Gamma]\!]$-labelled measure** $P : [\![\Gamma]\!] \to \mathcal{P}([\![\mathtt{S}]\!])$ when $\varepsilon_{[\![\mathtt{S}]\!]} \circ [\![s]\!] = P$. Since we now have dependent types, our denotational semantics takes place in slice categories $\mathbf{CG}{\downarrow}[\![\phi]\!]$ instead of **CG**, and we must therefore adapt the definition of targeting to the indexed setting.

For a topological space X, we extend the functor $\mathcal{P}$ from **Top** to $\mathbf{Top}{\downarrow}X$ by working fibre-wise. Given $p : Y \to X$, define

$$\mathcal{P}^X(p) \triangleq \coprod_{x\in X} (\mathcal{P}(p^{-1}(x)) \xrightarrow{!} 1 \xrightarrow{x} X);$$

the coproduct is taken in $\mathbf{Top}\downarrow X$, and $p^{-1}(x) \subseteq X$ is the fibre at x with the induced topology (in particular, it is closed whenever X is Hausdorff). The functor is defined on morphisms in the obvious way via fibre-wise pushforwards, and $\mathcal{P}_\perp^X$ is defined analogously. With the notation we used to define the terminal coalgebra interpreting sampler types, we can define for each $\mathbf{Top}\downarrow X$-object $p : Y \to X$ the **X-indexed-empirical transformation** $\varepsilon_p^X : \nu F^Y \to \mathcal{P}_\perp^X Y$ as the map

$$\varepsilon_p^X((y_1, w_1), (y_2, w_2), \ldots) = \begin{cases} \mathrm{in}_x\left(\lim_{n\to\infty} \sum_{i=1}^n \frac{w_n \delta_{y_n}}{\sum_{i=1}^n w_n}\right) & \text{if it exists} \\ \mathrm{in}_x(\perp) & \text{else} \end{cases}$$

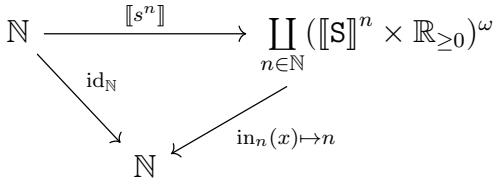
where $x = p((y_n, w_n)_n)$ is the common index of all samples (by construction of νF^Y), thus defining a $\mathbf{Top}\downarrow X$-morphism.

With this is place we can formalise our indexed version of targeting. Given a sampler $\phi; \Gamma \vdash s : \Sigma\,\mathtt{T}$ and a $[\![\phi;\Gamma]\!]$-labelled measure $P : [\![\phi;\Gamma]\!] \to \mathcal{P}^{[\![\phi]\!]}\,[\![\phi \vdash \mathtt{T} : \mathtt{Type}]\!]$, we say that s **targets** P, notation $\phi;\Gamma \vdash s \rightsquigarrow P$, if $\varepsilon^{[\![\phi]\!]}_{[\![\mathtt{T}]\!]} \circ [\![s]\!] = P$ (as $\mathbf{Top}\downarrow [\![\phi]\!]$-morphisms this time).

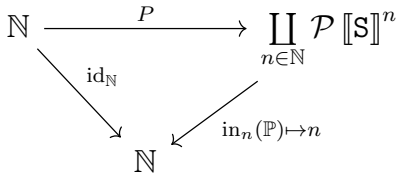
Example 4.3.1. *Let s be a closed sampler $\vdash s : \Sigma\,\mathtt{S}$, $[\![s]\!] \in ([\![\mathtt{S}]\!] \times \mathbb{R}_{\geq 0})^\omega$, and consider the dependently typed sampler*

$$n : \mathtt{N}; \emptyset \vdash s^n : \Sigma\,\mathtt{Vec}_n(\mathtt{S})$$

whose interpretation is given by the commutative **CG***-triangle*



with $[\![s^n]\!](n) = (([\![s]\!](1), \ldots, [\![s]\!](n)), ([\![s]\!](n+1), \ldots, [\![s]\!](2n)), \ldots)$. *An* $[\![n : \mathtt{N}; \emptyset]\!]$*-labelled measure* $P : 1 \to \mathcal{P}^{\mathbb{N}}\,[\![n : \mathtt{N} \vdash \mathtt{Vec}_n(\mathtt{S}) : \mathtt{Type}]\!]$ *is given by a commutative* **CG***-triangle*



It now follows that the dependently-typed sampler targets P, i.e. $n : \mathtt{N} \vdash s^n : \mathtt{Vec}_n(\mathtt{S}) \rightsquigarrow P$, iff, for all $n \in \mathbb{N}$,

$$\varepsilon^{\mathbb{N}}_{[\![\mathtt{S}]\!]^n} \circ [\![s]\!]\,(n) = P(n) = P(1) \otimes \ldots \otimes P(1);$$

that is, iff the sampler s is n-equidistributed with respect to the probability measure $P(1)$ for all $n \in \mathbb{N}$. (Recall from section 3.3.2 that this condition holds in the event that the sampler is weak-mixing.)

By working fibre-wise like in Example 4.3.1, the targeting calculus of the previous chapter (given in fig. 3.6) and its underlying proofs carry over to our current setting with no changes, other than the fact that the types (e.g. $\mathtt{Vec}_n(\mathtt{S})$) can now be dependent on an indexing context.

4.3.3 Constructing stochastic processes

Consider a stochastic process sampler $\vdash s : \mathtt{Marginal}(\mathtt{T}, \mathtt{S})$. (We assume a closed term for simplicity's sake; everything that follows extends easily to a sampler-in-context, but the notation becomes unnecessarily burdensome.) This is the type which *all* samplers for $[\![\mathtt{T}]\!]$-indexed, $[\![\mathtt{S}]\!]$-valued stochastic processes inhabit – and yet most programs of this type do not sample from any stochastic process. To argue that a particular sampler s does, we must show that taking samples obeys

$$n : \mathtt{N}; t : \mathtt{Vec}_n(\mathtt{T}) \vdash s(n)(t) : \Sigma\,(\mathtt{Vec}_n(\mathtt{S})) \rightsquigarrow P_t \tag{4.2}$$

for a $[\![n : \mathtt{N}; t : \mathtt{Vec}_n(\mathtt{T})]\!]$-labelled measure

$$P : [\![n : \mathtt{N}; t : \mathtt{Vec}_n(\mathtt{T})]\!] \to \mathcal{P}^{\mathbb{N}}\,[\![n : \mathtt{N} \vdash \mathtt{Vec}_n(\mathtt{S}) : \mathtt{Type}]\!] \tag{4.3}$$

which corresponds to the marginals of the desired process.

When conditions eq. (4.2) and eq. (4.3) are satisfied, and eq. (4.3) are the finite-dimensional marginals of a stochastic process with law ν, we write $\vdash s : \texttt{Marginal}(\texttt{T}, \texttt{S}) \overset{SP}{\rightsquigarrow} \nu$, and we say that s **targets the stochastic process with law** ν.

As we discussed in section 2.8, $s \overset{SP}{\rightsquigarrow} \nu$ iff the targeted marginals satisfy either the consistency conditions of Kolmogorov's extension theorem, or the projective limit conditions of Bochner's theorem. Note that there is nothing in the definition of the type $\texttt{Marginal}(\texttt{T}, \texttt{S})$ which guarantees these consistency conditions: in order to capture these type-theoretically, our system would need to contain something like "projective limit types". We choose instead to isolate the programs of type $\texttt{Marginal}(\texttt{T}, \texttt{S})$ which sample from a stochastic process using a meta-theoretical verification (targeting) calculus, like that in the previous chapter.

To enforce the consistency conditions between marginal samplers, we use the construction of processes via Bochner's theorem, and in particular its encoding of Kolmogorov's extension theorem. Recall that Bochner's theorem constructs $[\![\texttt{T}]\!]$-indexed, $[\![\texttt{S}]\!]$-valued processes via

$$\varprojlim_{t \in I} \mathcal{P}^0 S_t \cong \mathcal{P}^0(\varprojlim_t S_t) = \mathcal{P}^0(S^T). \tag{4.4}$$

where the directed set $(I, \leq)$ is defined by $I = \bigcup_n [\![\texttt{T}]\!]^n$ and for every $t = (t_1, \ldots, t_m)$ and $t' = (t'_1, \ldots, t'_n)$, $t \leq t'$ whenever $\{t_1, \ldots, t_m\} \subseteq \{t'_1, \ldots, t'_n\}$. Every pair $t \leq t'$ defines an injection $i_{t,t'} : m \to n$ mapping the position of each t_i in the tuple t to its position in the tuple t'. The projective system $(S_t)_{t \in I}$ is now defined as $[\![\texttt{S}]\!]_t = [\![\texttt{S}]\!]^n$ and for $\pi_{t',t} : [\![\texttt{S}]\!]^n \to [\![\texttt{S}]\!]^m, (s_1, \ldots, s_n) \mapsto (s_{i_{t,t'}(1)}, \ldots, s_{i_{t,t'}(m)})$.

Note that the indexing set I of the projective system is *precisely* the carrier of $[\![n : \texttt{N} \vdash \texttt{Vec}_n(\texttt{T}) : \texttt{Type}]\!]$. This means that we can extract from eq. (4.3) a

sequence

$$(P_t)_{t\in I} \qquad \text{where } P_t = \varepsilon^{\mathbb{N}}_{[\![S]\!]^n} \circ [\![s]\!](n)(t) \quad \text{by eq. (4.2)}$$

This sequence defines a process $\nu \in \mathcal{P}^0([\![\mathtt{S}]\!]^{[\![\mathtt{T}]\!]}) = \varprojlim_t \mathcal{P}^0 [\![\mathtt{S}]\!]_t$ if for all $t' \leq t$ the following consistency condition holds:

$$P_{t'} = \mathcal{P}\pi_{t,t'}(P_t)\,. \tag{4.5}$$

Writing $\pi_t : \varprojlim_t \mathcal{P}^0 [\![\mathtt{S}]\!]_t \to \mathcal{P}^0 [\![\mathtt{S}]\!]_t$ for the obvious projections, this gives us our first introduction rule for the stochastic process targeting calculus.

$$\frac{}{\vdash s : \mathtt{Marginal}(\mathtt{T},\mathtt{S}) \overset{SP}{\rightsquigarrow} \nu} \; \textit{eq. (4.2), eq. (4.5) hold}, P_t = \mathcal{P}\pi_t\nu \tag{4.6}$$

That this rule is sound follows from Bochner's theorem.

Remark 4.3.2. *It is the use of dependent types in our language which makes this construction rule semantically natural and easy to present. The projective system of measures $(P_t)_{t\in T}$, exactly what is required to construct a stochastic process, is obtained easily thanks to the presence of the type* $\mathtt{Marginal}(\mathtt{T},\mathtt{S})$ *in our language. This pleads strongly in favour of dependently-typed probabilistic languages when dealing with stochastic processes.*

Remark 4.3.3. *The way in which a family of marginal samplers s 'targets' the law μ of a stochastic process is formally unlike the notion of targeting discussed in chapter 3, in that μ is a measure on a product algebra $[\![\mathtt{S}]\!]^{[\![\mathtt{T}]\!]}$ for some types $\mathtt{S},\mathtt{T}$, but s is* not *a sampler on the function type $\Sigma(\mathtt{S} \to \mathtt{T})$. Nevertheless, there is a sense in which the two notions of targeting are related. If an $\mathtt{S}$-valued sampler s targets the probability distribution μ, then s can compute the expectations under μ of all continuous bounded functions $f : [\![\mathtt{S}]\!] \to \mathbb{R}$; if the marginal samplers s target the law μ, then s can compute the expectations under μ of all continuous bounded functionals $f : [\![\mathtt{S}]\!]^{[\![\mathtt{T}]\!]} \to \mathbb{R}$ which factor through a finitary evaluation functional $\mathrm{ev}_t : [\![\mathtt{S}]\!]^{[\![\mathtt{T}]\!]} \to [\![\mathtt{S}]\!]^n$ for some $t \in [\![\mathtt{T}]\!]^n$.*

Example 4.3.4 (White noise)**.** *Let* $\texttt{WhiteNoise} : \Sigma\,\texttt{S} \to \texttt{Marginal}(\texttt{T}, \texttt{S})$ *represent the function*

$$s : \Sigma\,\texttt{S} \vdash \lambda n : \texttt{N} \,.\, \lambda t : \texttt{Vec}_n(\texttt{T}) \,.\, s^n : \texttt{Marginal}(\texttt{T}, \texttt{S}).$$

and fix a closed sampler $\vdash s : \Sigma\,\texttt{S}$*. Recall first from Example 4.3.1 that, in order for eq.* (4.2) *to hold for* $\texttt{WhiteNoise}(s)(n)(t)$*, it must be the case that* $P_t = P \otimes \ldots \otimes P$*, the* n*-fold product of a fixed probability measure* P*, and that the sampler* s *is* n*-equidistributed.*

Since each P_t *is the product of a fixed univariate distribution* P*, the consistency condition eq.* (4.5) *follows almost immediately (from the commutativity of multiplication on the reals). Assuming that* s *is* n*-equidistributed for each* $n \in \mathbb{N}$*, we can apply the introduction rule eq.* (4.6) *to show* $\vdash \texttt{WhiteNoise}(s) : \texttt{Marginal}(\texttt{T}, \texttt{S}) \overset{SP}{\rightsquigarrow} \nu$*, where* ν *is the white noise process defined by the univariate probability measure* $P = \varepsilon^{\mathbb{N}}_{[\![\texttt{S}]\!]} \circ \texttt{WhiteNoise}(s)(1)(t)$*.*

Example 4.3.5 (Gaussian process)**.** *For an arbitrary Gaussian process (see section 2.8) on* $\mathbb{R}$ *with mean function* $\mu : \texttt{T} \to \texttt{R}$ *and covariance kernel* $\kappa : \texttt{T} \times \texttt{T} \to \texttt{R}^+$*, which together make up the context* Γ*, the desired marginals* P *are explicitly given by*

$$P(\mu, \kappa)(n)(t_1, \ldots, t_n) = \mathrm{N}\left(\begin{bmatrix} \mu(t_1) \\ \vdots \\ \mu(t_n) \end{bmatrix}, \begin{bmatrix} \kappa(t_1, t_1) & \cdots & \kappa(t_1, t_n) \\ \vdots & \ddots & \vdots \\ \kappa(t_n, t_1) & \cdots & \kappa(t_n, t_n) \end{bmatrix} \right).$$

That is, each marginal is a Gaussian distribution whose mean is given by applying μ *to* $(t_1, \ldots, t_n)$*, and whose covariance is the Gram matrix obtained by applying the kernel* κ *to* $(t_1, \ldots, t_n)$*. In order for the above marginals to be well-defined, we must assume that* κ *is a* positive-definite *kernel, i.e. that these Gram matrices are positive-definite for all* $t_1, \ldots, t_n \in [\![\texttt{T}]\!]$*. That these marginals obey the consistency conditions of eq.* (4.5) *follows immediately from a well-known property of the Gaussian distribution: for any linear*

$A : \mathbb{R}^m \to \mathbb{R}^n$, *if* $x \sim \mathrm{N}(\mu, \Sigma)$, *then* $Ax \sim \mathrm{N}(A\mu, A\Sigma A^T)$. *Therefore, writing a program which samples from the finite-dimensional marginals of an arbitrary Gaussian process is immediate,* provided *that one can sample from a multivariate Gaussian distribution with arbitrary mean and covariance.*

Such samplers are easily implemented using standard techniques. It follows from the aforementioned property of Gaussians that, if $z = (z_1, \ldots, z_n)$ *where* $z_i \sim \mathrm{N}(0, 1)$ *are i.i.d., the variable* $\mu + Lz$ *is distributed as* $\mathrm{N}(\mu, LL^T)$. *Given a positive-definite* Σ, *the unique lower-triangular* L *such that* $LL^T = \Sigma$ *is known as its Cholesky decomposition. Assume that functions* `cholesky`, `matvec`, *and* `vadd` *have been implemented, which, respectively, compute the Cholesky decomposition, multiply a matrix by a vector, and add two vectors*[1]*; listing 4.11 uses these to define* `MVN`, *a family of samplers for arbitrary multivariate Gaussian distributions, given a Gaussian sampler* $s : \Sigma\,\mathtt{R}$.

```
λs : Σ R  .  λn : N  .  λ(μ, Σ) : Vec_n(R) × Mat_{n×n}(R)  .
  map (λz : Vec_n(R)  .
       vadd(n)(μ, matvec(n, n)(cholesky(n)(Σ), z)), s^n)
  :  Σ R → (Π n : N . Vec_n(R) × Mat_{n×n}(R) → Σ Vec_n(R))
```

Listing 4.11: Definition of `MVN`

To show the validity of this family of samplers in a context Γ *is to demonstrate the truth of the implication:*

$$\frac{n : \mathtt{N}\ ;\ \Gamma \vdash s^n : \Sigma\,(\mathtt{R}^n) \rightsquigarrow \mathrm{N}(0,1)^n \quad n : \mathtt{N}\ ;\ \Gamma \vdash \mu : \mathtt{Vec}_n(\mathtt{R}) \quad n : \mathtt{N}\ ;\ \Gamma \vdash \Sigma : \mathtt{Mat}_{n\times n}(\mathtt{R})}{n : \mathtt{N}\ ;\ \Gamma \vdash \mathtt{MVN}(s)(\mu, \Sigma) : \Sigma\,(\mathtt{Vec}_n(\mathtt{R})) \rightsquigarrow \mathrm{N}(\mu, \Sigma)} \tag{4.7}$$

That is, on the assumption that the sampler s *is* n*-equidistributed with respect to a standard Gaussian distribution, it follows that* `MVN` *generates a sample from the desired Gaussian distribution.*

As verifying eq. (4.7) *falls more under the purview of verifying linear*

[1] Simple implementations of `matvec` and `vadd` are given in section 4.3.1. We eschew an implementation of `cholesky` here, as its implementation is fairly wordy, and is most naturally given recursively, which would require additional language features. Our primary aim here is the verification of stochastic process samplers, not of linear algebra.

algebra (in particular, the validity of the Cholesky decomposition and of matrix-vector multiplication) than verifying sampling, we do not focus on it here.

By applying the function MVN *to a suitable Gaussian sampler* $s : \Sigma\,\mathtt{R}$*, listing 4.12 quickly constructs a Gaussian process sampler for arbitrary mean function* μ *and covariance kernel* κ*.*

```
λs : ΣR . λ(μ, κ) : (T → R) × (T × T → R⁺) .
  λn : N . λt : Vec_n(T) .
    let mean = vec(n)(λi : N . μ(get(n)(i, t))) in
    let cov = mat(n, n)(λ(i, j) : N × N .
                  κ(get(n)(i, t), get(n)(j, t)) in
                    MVN(s)(n)(mean , cov)
  : ΣR → ((T → R) × (T × T → R⁺)) → Marginal(T, R)
```

Listing 4.12: Definition of `GaussianProcess`

Applying eq. (4.7) *in this case, assuming that its premises hold, gives* $n : \mathtt{N}\ ;\ \Gamma \vdash \mathtt{MVN}(s)(\mathtt{mean}, \mathtt{cov}) : \Sigma\,(\mathtt{Vec}_n(\mathtt{R})) \rightsquigarrow \mathrm{N}(\llbracket \mathtt{mean} \rrbracket, \llbracket \mathtt{cov} \rrbracket)$*. Therefore, provided only that (in context)*

$$\llbracket \mathtt{mean} \rrbracket = \begin{bmatrix} \mu(t_1) \\ \vdots \\ \mu(t_n) \end{bmatrix}, \qquad \llbracket \mathtt{cov} \rrbracket = \begin{bmatrix} \kappa(t_1, t_1) & \cdots & \kappa(t_1, t_n) \\ \vdots & \ddots & \vdots \\ \kappa(t_n, t_1) & \cdots & \kappa(t_n, t_n) \end{bmatrix},$$

it follows that, writing `GaussianProcess` *for listing 4.12,*

$$n : \mathtt{N}; \Gamma \vdash \mathtt{GaussianProcess}(s)(\mu, \kappa)(n)(t) \rightsquigarrow P(\mu, \kappa, n)(t)$$

with $P(\mu, \kappa)(n)(t)$ *as defined above. Finally, as eq.* (4.5) *holds for this system of probability measures, the introduction rule eq.* (4.6) *gives us*

$$\Gamma \vdash \mathtt{GaussianProcess}(s)(\mu, \kappa) \overset{SP}{\rightsquigarrow} \mathrm{GP}(\mu, \kappa).$$

$$\frac{}{\vdash s : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \nu} \; eq.\ (4.2), eq.\ (4.5) \text{ hold}, P_t = \mathcal{P}\pi_t\nu$$

$$\frac{\phi\ ;\ \Gamma \vdash s \approx t : \texttt{Marginal}(\texttt{T},\texttt{S}) \quad \phi\ ;\ \Gamma \vdash s : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \mu}{\phi\ ;\ \Gamma \vdash t : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \mu}$$

$$\frac{\phi\ ;\ \Gamma \vdash s : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \nu \quad \phi\ ;\ \Gamma \vdash f : \texttt{S} \to \texttt{S}'}{\phi\ ;\ \Gamma \vdash \texttt{spmap}(f,s) : \texttt{Marginal}(\texttt{T},\texttt{S}') \overset{SP}{\rightsquigarrow} \gamma \mapsto (u \mapsto [\![f]\!](\gamma) \circ u)_*\nu(\gamma)}$$

$$\frac{\phi\ ;\ \Gamma \vdash s : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \nu \quad \phi\ ;\ \Gamma \vdash f : \texttt{S} \to \texttt{R}^+ \quad \phi;\Gamma \vdash t : \texttt{T}}{\phi\ ;\ \Gamma \vdash \texttt{spreweight}(f,t,s) : \texttt{Marginal}(\texttt{T},\texttt{S}) \overset{SP}{\rightsquigarrow} \gamma \mapsto (\lambda u.\, [\![f]\!](\gamma)(u([\![t]\!](\gamma)))) \cdot \nu(\gamma)}\ \dagger$$

† Note the side condition $\int [\![f]\!](\gamma)(u([\![t]\!](\gamma)))d\nu(\gamma) \in (0,\infty)$.

Figure 4.6: Rules for stochastic process targeting.

4.3.4 Transforming stochastic processes

We consider two important transformations $\texttt{Marginal}(\texttt{T},\texttt{S}) \to \texttt{Marginal}(\texttt{T},\texttt{S}')$ between stochastic process samplers. They are the infinite-dimensional counterparts of the `map` and `reweight` operations on samplers.

We start by defining `spmap`, the stochastic process version of `map`, in listing 4.13. Given $f : \texttt{S} \to \texttt{S}'$, it applies f separately and element-wise to each dimension of the marginals produced by s. For example, in the event that $\texttt{S} = \texttt{R} \times \texttt{R}, \texttt{S}' = \texttt{R}$, and $f : \texttt{R} \times \texttt{R} \to \texttt{R}$ is addition, $\texttt{spmap}(f,s)$ essentially inputs a $\mathbb{R}^2$-valued stochastic process (sampler) and outputs a $\mathbb{R}$-valued stochastic process (sampler) by adding the two dimensions.

```
λ(f, s) : (S → S′) × Marginal(T, S) . λn : N . λt : Vec_n(T) .
  map(λv : Vec_n(S) . vec(n)(λi : N . f(get(n)(i, v))), s(n)(t))
: (S → S′) × Marginal(T, S) → Marginal(T, S′)
```

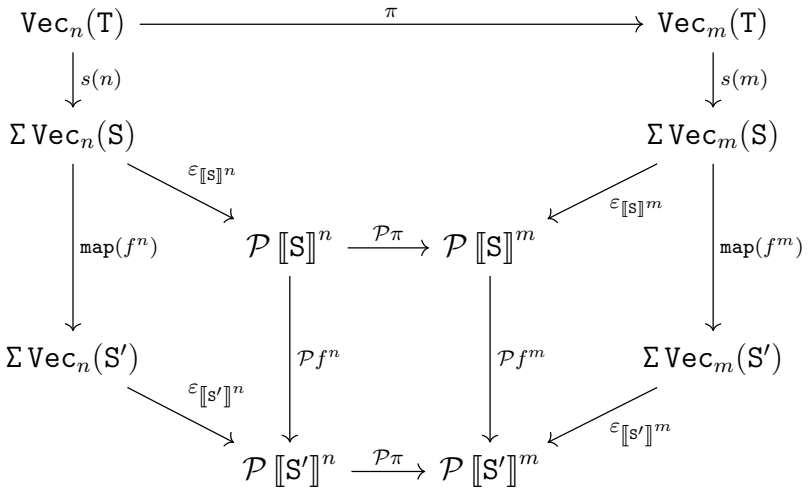
Listing 4.13: Definition of `spmap`

The rule for `spmap` found in fig. 4.6 shows that stochastic process targeting can be pushed through `spmap`.

Theorem 4.3.6. *The rule for* `spmap` *is sound.*

Proof of soundness for `spmap` *rule.* Let $\pi = \pi_{t,t'}$ be any projection-permutation

composition as described in section 4.3.3. The proof hinges on proving the commutativity of the following diagram.



The left- and right-most vertical paths select the marginal distributions targeted by the sampler $\texttt{spmap}(f, s)$. Therefore, if it can be shown that this diagram commutes, then it follows that eq. (4.5) holds, and so that $\texttt{spmap}(f, s)$ targets a valid stochastic process.

The top hexagon commutes by assumption: we assume that s targets a valid stochastic process. The right- and left-bottom faces commute by the pushforward, i.e. `map`, rule of fig. 3.6. Finally, the middle bottom square commutes because π is a projection-permutation composition, and thus $\pi \circ f^n = f^m \circ \pi$, which is preserved by functoriality of $\mathcal{P}$. Therefore, `spmap` is sound. □

This rule allows use to modify a valid stochastic sampler (and products thereof), in any arbitrary *pointwise* way, and recover another valid stochastic sampler.

The next transformation rule is crucial for implementing Bayesian conditioning on sampled values from stochastic processes; it is the stochastic process counterpart of the `reweight` rule for standard samplers. The transformation listing 4.14 reweights a stochastic process sampler $s : \texttt{Marginal}(\texttt{T}, \texttt{S})$ depending on the value of its marginals at a given point t_0, using a given reweighting function f. For example, in the event that f is a Gaussian density with lo-

cation y and scale σ^2, and s targets the stochastic process $X(t)$, listing 4.14 constructs, using importance sampling, a sampler which targets the posterior process conditioned on the noisy observation $y \mid X(t_0) \sim \mathrm{N}(X(t_0), \sigma^2)$. The simple vector operations `take` and `cons` are defined in section 4.3.1.

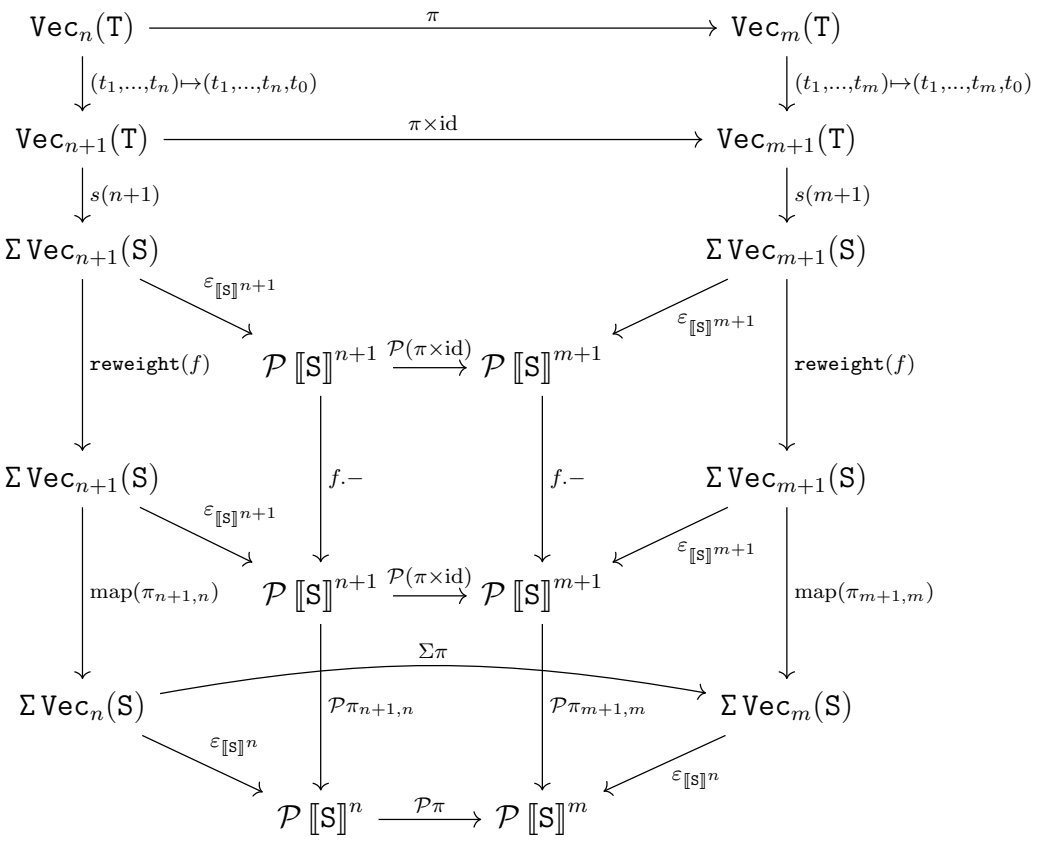
```
λ(t₀, f, s) : T × (S → R⁺) × Marginal(T, S) . λn : N . λt : Vecₙ(T).
  map(λt′ : Vecₙ₊₁(S) . take(n + 1)(n, t′),
    reweight(λt′ : Vecₙ₊₁(S) . f(get(n + 1)(n + 1, t′)),
      s(n + 1)(cons(n)(t, t₀))))
: T×(S→ R⁺) × Marginal(T, S) → Marginal(T, S)
```

Listing 4.14: Definition of `spreweight`

The rule associated with `spreweight` is given in fig. 4.6.

Theorem 4.3.7. *The rule for* `spreweight` *is sound.*

Proof of soundness for `spreweight` *rule.* Let $\pi = \pi_{t',t}$ be any projection-permutation composition as described in section 4.3.3. The proof hinges on proving the commutativity of the following diagram.



As in the proof of validity of `spmap`, the left- and right-most vertical paths select the marginal distributions of the sampler targeted by $\texttt{spreweight}(f, s)$. To show the commutativity of this diagram is to show eq. (4.5), and thus to show that $\texttt{spreweight}(f, s)$ targets a valid stochastic process.

The bottom trapezium commutes by the `map`, i.e. pushforward, rule, as do the bottom face involving the projection $\pi_{n+1,n}(t_1, \ldots, t_n, t_{n+1}) = (t_1, \ldots, t_n)$ and the bottom face involving the projection $\pi_{m+1,m}$. The bottom middle face commutes trivially by definition of the maps involved and the functoriality of $\mathcal{P}$. The middle square involving the operation $f.-$ commutes trivially by definition of the reweighting function (which only involves $f(t_0)$), and the two faces on either side commute by the `reweight` rule. Finally, the top hexagon commutes because s defines a valid sampler and because $\pi \times \mathrm{id}$ is a projection-permutation composition and the top diagram commutes trivially by definition. Therefore, `spreweight` is sound. □

By chaining several applications of this rule it is possible to prove that conditioning, say a Gaussian process (constructed in Example 4.3.5) on any finite number of observations produces a valid stochastic process implementing a posterior distribution. This kind of construction is central to non-parametric Bayesian techniques in probabilistic programming languages.

Example 4.3.8 (Wiener process conditioning)**.** *For a simple example of conditioning a stochastic process, consider the Wiener process $W(t)$, which is the $\mathbb{R}$-valued, $\mathbb{R}^+$-indexed Gaussian process with mean function $\mu(t) = 0$ and covariance function $\kappa(t, t') = \min(t, t')$. Listing 4.15 first constructs this Gaussian process, building on Example 4.3.5, and then conditions on noisy observations $y_1 \sim \mathrm{N}(W(1), \sigma^2)$ and $y_2 \sim \mathrm{N}(W(2), \sigma^2)$, for some noise level σ. This returns another stochastic process on $\mathbb{R}$, conditioned on having approximately $W(1) \approx y_1, W(2) \approx y_2$ (for sufficiently small observation noise σ^2, that is).*

Example 4.3.9 (Products of stochastic processes)**.** *In the previous chapter, we defined a natural product $\otimes$ of samplers, and noted that product samplers $s \otimes s'$ jointly targeting a product measure $P \times P'$ is a strictly stronger condition*

```
λs : ΣR .
  let μ = λt : R . 0 in
  let κ = λ(t,t') : R × R . min(t,t') in
  let W = GaussianProcess(s)(μ, κ) in
  let f = λx,y : R . 1/(2*π*σ**2) * exp(-0.5*(y-x)**2) in
    spreweight(λx : R . f(x,y₂), 2,
      spreweight(λx : R . f(x,y₁), 1, W))
  : ΣR → Marginal(R⁺,R)
```

Listing 4.15: Wiener process: noisy observation

than s targeting P and s' targeting P' separately. A similar result applies in the case of stochastic process samplers. Given two S*-valued,* T*-indexed stochastic process samplers* s : Marginal(T, S), s' : Marginal(T, S′)*, define their product* spprod(s, s') : Marginal(T, S × S′) *as given in listing 4.16.*

```
λ(s,s') : Marginal(T,S) × Marginal(T,S') .
  λt : Vecₙ(T) .
    map(λ(x,y) : Vecₙ(S) × Vecₙ(S') .
      vec(n)(λi : N . (get(n)(i,x), get(n)(i,y)),
      s(n)(t) ⊗ s'(n)(t)))
: Marginal(T,S) × Marginal(T,S') → Marginal(T,S × S')
```

Listing 4.16: Definition of spprod

By a straightforward application of the equivalence rules of fig. 3.5 and fig. 4.5, it can be verified that

$$\phi; \Gamma \vdash \texttt{spmap}(\texttt{fst}, \texttt{spprod}(s, s')) \approx s : \texttt{Marginal}(\texttt{T}, \texttt{S})$$

and likewise

$$\phi; \Gamma \vdash \texttt{spmap}(\texttt{snd}, \texttt{spprod}(s, s')) \approx s' : \texttt{Marginal}(\texttt{T}, \texttt{S}').$$

As a result, applying the equivalence and spmap *rules from fig. 4.6, it follows that*

$$\frac{\phi; \Gamma \vdash \texttt{spprod}(s, s') \overset{SP}{\rightsquigarrow} \nu}{\phi; \Gamma \vdash s \overset{SP}{\rightsquigarrow} \gamma \mapsto (u \mapsto \pi_1 \circ u)_* \nu(\gamma)}$$

and

$$\frac{\phi;\Gamma \vdash \texttt{spprod}(s,s') \overset{SP}{\rightsquigarrow} \nu}{\phi;\Gamma \vdash s' \overset{SP}{\rightsquigarrow} \gamma \mapsto (u \mapsto \pi_2 \circ u)_*\nu(\gamma)}$$

though the converse clearly does not hold.

One might also define the sum of two real-valued stochastic processes, `spsum`*, as in listing 4.17.*

```
λ(s, s′) : Marginal(T, R) × Marginal(T, R) .
  spmap (λ(x, y) : R × R . x + y,
    spprod (s, s′))
: Marginal(T, R) × Marginal(T, R) → Marginal(T, R)
```

Listing 4.17: Definition of `spsum`

The above rules for transforming stochastic process samplers are far from complete[2], and might be extended with a number of transformations on stochastic processes seen in the literature. More interestingly, though, rules of this sort can only contain at most a subset of the transformations on stochastic processes often considered: those which can be expressed in terms of transformations on the process' marginals. The literature does contain a number of transformations which cannot: for example, given an almost surely continuous stochastic process $X(t) \in \mathbb{R}$, consider the stochastic process $Y(t) = \int_0^t X(s)\,ds$; while the finite-dimensional marginals of $Y(t)$ can be *approximately* expressed in terms of those of $X(t)$, a precise expression is impossible.

[2]Completeness is not reasonable to expect in this setting for the same reasons discussed in the previous chapter; see Remark 3.3.11.

Chapter 5

Discussion

Contributions. In this thesis, we introduced a probabilistic programming language for reasoning about sampling tasks, featuring deterministic semantics. We presented an operational and denotational semantics in this setting, and showed an adequacy result relating them. We gave an effective system for simplifying samplers using equivalence rules. We introduced a targeting calculus that relates these samplers to the measures that they target, and which includes rules corresponding to the most common techniques for constructing samplers. We then broadened this language to include dependent types, completely reworking its denotational semantics, in order to discuss families of samplers which target the marginals of a stochastic process. Finally, we developed an analogous targeting calculus for reasoning about systems of samplers which target (the marginal distributions of) a stochastic process.

This thesis aimed to, by weaving together advancements which have been made in a number of fields since von Neumann's oft-quoted remark about pseudorandom number generation, present a formal theory of sampling, samplers, and their verification. We draw inspiration and techniques from fields that were either nascent or nonexistent at the dawn of computing, including in particular the modern fields of formal semantics and verification, to progress towards this goal.

In the few pages that remain, we situate our approach within a few adjacent and overlapping fields of study, compare and contrast with a few other

approaches to formal sampler verification, and discuss natural extensions of this work.

5.1 Related work

Trace semantics. Trace semantics, introduced in [10] and further developed in [54, 55, 56], give probabilistic programs an essentially deterministic operational semantics. Each built-in distribution is paired with an infinite sequence of samples. When a sample for each distribution is requested, the head of this sequence is popped and used in the computation, and the tail of the sequence retained for further sampling.

Trace semantics is philosophically in line with our approach to semantics, but we do not assume that samples are 'pre-generated' in this fashion. Instead, we introduce samplers as first-class objects, and include the process of (pseudo-)random number generation within our language. We do this in order to address what we see as two problems with the standard trace semantics approach.

First, the assumption that all random quantities are generated ahead-of-time, while mathematically convenient, is not realistic and is incompatible with pseudorandom generation of random variables. This issue exposes a gap between an operational semantics based on trace semantics and a denotational semantics based on probability measures (or similar algebraic objects). For a simple example, consider a sequence of samples $(x_0, x_1, \ldots)$ produced by iterating a computable function $x_{n+1} = T(x_n)$, and consider the program $T(\texttt{sample}()) - \texttt{sample}()$. Under a reasonable denotational semantics, this program would map to a nontrivial probability measure, and yet under our operational semantics, it always evaluates to zero. In order to rule out the possibility of such adversarial programs, one must assume that the sequence $(x_0, x_1, \ldots)$ is *Martin-Löf random* [57] – though unfortunately, all Martin-Löf random sequences are uncomputable. We instead allow pseudorandom number generation under far weaker conditions that Martin-Löf randomness, and so

obtain adequacy between our operational and denotational semantics.

Second, we find the fact that trace semantics models the operation of sampling as inherently effectful to be problematic for verification. Our primary interest in verification is enabling techniques which construct samplers in a compositional manner, building more complex samplers from simpler ones. If two samplers cannot be soundly composed together in this manner to yield a valid sampler, we would prefer to model this as a fact about those samplers rather than a fact about the state of the machine. To that end, we introduce a type representing samplers, and we model the operation of our samplers without side effects, so that their verification can be undertaken with no reference to state.

Denotational semantics. As alluded to earlier, [10] presented a semantics for an imperative probabilistic programming language, though this approach does not immediately extend to a higher-order setting due to Remark 2.1.7. Subsequent approaches on denotational semantics for probabilistic programming languages have incorporated continuous distributions and higher-order functions in particular, by, essentially, replacing the probability measure with a more general algebraic object. More concretely, a denotational universe of types is chosen (ideally one with nice structure, e.g. a Cartesian closed category), and an equivalent to the Giry monad constructed within that category – thereby generalising the notion of 'measure' and 'measurable space'.

In particular, [12] chooses as its types countable-product-preserving presheaves using a Yoneda embedding, and lifts the Giry monad to its left Kan extension, a monad on these presheaves; relatedly, Quasi-Borel space semantics [21] replace measurable spaces with quasi-Borel spaces, and [13] extends this approach to include recursion. [14] instead constructs a Giry monad over regular ordered Banach spaces, while [58] instead considers the category of complete cones.

Our approach differs substantially from these approaches in that our semantics is deterministic: the semantics of a probabilistic program is an infinite

stream of samples. These streams are related to measures in a natural way, by what we call the 'empirical transformation' (see section 3.3.1), but they are sufficiently algebraically different from measures that a concern analogous to Remark 2.1.7 does not arise. (Instead it is continuity that we must be careful about; see section 3.1.3.)

Markov chain semantics. Markov chain semantics [59, 60, 61, 62, 63, 64, 65] are primarily applied to probabilistic λ-calculi to model the probability (or probability density) of a probabilistic program evaluating to a particular value. These are, from our perspective, closer to a denotational semantics than an operational semantics – unless a trace of generated samples is included within the semantics, in which case they are similar to trace semantics.

Pseudorandom number generation, quasi-Monte Carlo, and derandomisation. While we do not meaningfully contribute to the field of pseudorandom number generation, we draw motivation and terminology from it. The concept of k-equidistribution [40, 41, 42] is particularly important in the development of our targeting calculus.

We will also briefly acknowledge that our work is not intended to meaningfully contribute to the array of derandomised approaches for producing 'optimal' samples targeting a certain distribution by deterministic means, such as quasi-Monte Carlo [43, 44] and kernel herding [66]. While such methods can certainly be used to create samplers in our language to the extent that they possess the relevant convergence criteria (though see the remarks in section 5.2 about goodness-of-fit for a discussion of using convergence criteria other than weak convergence), devising new methods for generating 'optimal' samples is not the direction of our research.

Algorithmic randomness. Similarly, we are inspired by the study of algorithmic randomness, but we make no meaningful contributions to it. The concept of Martin-Löf randomness [57], as well as Schnorr's reformulation [67] and Hoyrup's recent generalisation to arbitrary metric spaces [68], guided our thinking and pushed us towards requiring weaker statistical guarantees for our

samplers.

Quantum computation. Our discussion of sampling and samplers is purely classical. Quantum methods can certainly be used to produce high-quality 'true' random numbers [69], and incorporated into our targeting calculus as axioms, as shown in fig. 3.6. Naïvely, one would expect that random numbers produced by quantum measurement for classical usage would have good statistical properties, such as k-equidistribution for all k.

Verifying statistical properties of probabilistic programs. A wide literature exists on an array of verification tasks involving probabilistic programming languages. However, the primary type of verification task that we are interested in is verifying that the outputs of probabilistic programs have certain statistical properties, such as being distributed (asymptotically or otherwise) according to a desired distribution – for the purpose of verifying implementations of sampling techniques such as importance and rejection sampling. Particularly important comparisons include [56] and [70].

In [56], a sampling language is constructed in which the denotation of a sampler consists of a *sampling function*, i.e. a mapping from a countable number of uniform random variables specifying a means by which countably many uniform samples might be transformed into samples from the desired distribution. This enables the compositional verification of samplers, assuming, as in the case of trace semantics, an inexhaustible source of independent uniform random numbers. As was extensively discussed in chapter 3, our approach differs from this approach in that we allow for pseudorandom number generation, and explicitly keep track of the independence guarantees necessary in order to obtain weak convergence.

[70] is perhaps the most directly focused on verifying the correctness of statistical inference (in particular, Bayesian inference). This thesis models the inference *process* as a program transformation which inputs an intractable specification of a posterior distribution and outputs a tractable approximation to it (e.g. via one of many Monte Carlo sampling techniques, such as

sequential Monte Carlo). The semantics is set in the category of quasi-Borel spaces, and sampling is described as a monadic operation. While, like the previous approaches, this differs significantly from our semantics of sampling, and [70] focuses primarily on justifying Bayesian inference, our aims – to construct languages in which certain sampling techniques are easily seen to be compositionally valid – are very similar.

Finally, our approach to constructing and verifying stochastic process samplers is informed by the design of probabilistic programming languages. These languages have implemented stochastic process samplers since the development of some of the earliest higher-order probabilistic programming languages [48, 46], particularly to enable nonparametric Bayesian inference. (Indeed, it is a clear demonstration of the utility of the higher-order probabilistic programming language that it is natively capable of this.)

Most of the literature on stochastic process samplers focuses on their implementation and on applications to Bayesian nonparametrics, rather than focusing on stochastic processes as a datatype and on verification. A recent exception is [11], which implements stochastic processes lazily and recursively; our approach differs in that our language is higher-order but without recursion.

5.2 Further work

We conclude this thesis by listing a few potential avenues for extending the work we have presented here, along with some preliminary ideas on how one might proceed.

Recursion. In Remark 3.1.1, we discussed the reasons for our language's lack of explicit recursion. There, we hinted at the difficulty involved in formalising a general notion of 'recursive sampler' that lends itself to verification, and we asserted that these samplers can in fact be specified in our language without adding any additional features. Here we will expand on those remarks.

As discussed previously, incorporating unrestricted recursion certainly does not serve our purposes. Our aim is to construct programming languages

in which the construction of a sampler quickly provides its verification, by incorporating rules in our targeting calculus for each sampling technique used in constructing a sampler. Verifying probabilistic properties of samples produced by unrestricted recursive procedures is clearly intractable in general, so no such rule is possible. Moreover, the vast majority of these recursive procedures are meaningless from a probabilistic perspective. If recursion were to be allowed, it would have to be a heavily restricted type of recursion that lends itself to verification.

As far as we are aware, a general characterisation of the notion of 'recursive sampler' that lends itself to verification does not exist. Consider, for example, the particular case of the recursive geometric sampler, which generates samples from the geometric distribution with parameter p as follows. Initialise n, the number of trials, at zero; at each step, flip a biased coin with probability p of landing heads; if that coin lands tails, then increment n and flip another coin; if that coin lands heads, then return n. This sampler is a recursive procedure that incorporates a changing state, the number of trials n, and yet its treatment of this state is quite constrained. A notion of 'recursive sampler' should generalise this type of sampler with internal state in a way that makes general-purpose verification feasible.

One might naturally define a 'recursive sampler', or at least a family of them, as essentially a while-loop which includes an internal state z_i, initialised to z_1, as follows. At each step, generate a sample $x_i \sim Q$ from some fixed distribution Q on X; if the condition $w(z_i, x_i)$ is false, halt and returns a function of the internal state $g(z_i)$; if the condition is true, then update the internal state $z_{i+1} = f(z_i, x_i)$ and loop. This procedure clearly includes our previous geometric sampler as a special case, and it is amenable to standard probabilistic analysis: provided that f, w, g are measurable and that this procedure's stopping time is almost surely finite, inductive strategies for proving that the return values $y = g(z_i)$ are distributed according to some target P are natural (which is not to say they are easy).

The deterministic verification of such samplers is more challenging, but almost obtainable: we must show that, subject to the assumption that a sampler s which generates our samples x_i and targets Q is infinitely equidistributed, i.e. $\forall n, s^n \rightsquigarrow Q^n$, the samples $y = g(z_i)$ generated upon halting $w(z_i, x_i) = 0$ target a desired probability measure P. Some ideas of how this might be accomplished follow. Summing over halting times τ, this is the requirement that for any bounded continuous f,

$$\int_Y f(y)\, dP(y) = \sum_{\tau=1}^{\infty} \int_{X^\tau} f(g(z_\tau(x_{1:\tau-1}))) \left[\prod_{i=1}^{\tau} w(z_i(x_{1:i-1}), x_i)\right] (1 - w(z_\tau(x_{1:\tau-1}), x_\tau))\, dQ^\tau(x_{1:\tau})$$

where we abbreviate $z_2(x_1) = f(z_1, x_1)$, $z_3(x_{1:2}) = f(z_2(x_1), x_2) = f(f(z_1, x_1), x_2)$, and so on. Expanding these integrals over X^τ to common integrals over X^ω and pulling the sum inwards, provided this can be justified, gives a more standard integral $\int_Y f(y)\, dP(y) = \int_{X^\omega} \varphi(x_{1:\infty})\, dQ^\omega(x_{1:\infty})$ where the form of φ follows. Unfortunately, the while loop $w(z_i, x_i)$ involved ensures that this function φ is certainly not continuous, which means that we would have to keep track of the discontinuities that it imposes during our verification task – which seems challenging.

We didn't pursue this direction because, as noted in Remark 3.1.1, samplers can typically be implemented in a non-recursive manner – but verifying a subset of recursive samplers by construction, and adding a corresponding rule to our targeting calculus, would be a natural next step.

Finally, note that our language already contains the ability to implement such recursive samplers by making somewhat unusual use of its existing language features. For example, given an unweighted sampler $s : \Sigma\, \mathtt{B}$ targeting a Bernoulli distribution with success probability p, and with sufficient equidistribution properties, listing 5.1 implements the recursive geometric sampler mentioned above.

This sampler increments a counter after every `False` sample from s, and resets the counter after `True` is drawn. After each sample is taken, the first element of the sampler s is popped. Intermediate samples in which `False` is

```
map(λ(b, s, i) : B×ΣB×N . i,
  reweight(λ(b, s, i) : B×ΣB×N . if b then 1 else 0,
    prng(λ(b, s, i) : B×ΣB×N .
         (hd(s), tl(s), if b then 1 else i + 1),
       (False, s, 0)))) : ΣN
```

Listing 5.1: Recursive geometric sampler

drawn are assigned weight zero, and the internal variables b and s dropped, yielding a geometric sampler on the natural numbers.

Of course, our targeting calculus cannot be directly used to prove the convergence of this sampler, as it is not immediately clear to the author how to show that this use of `prng`, which produces a sampler of type $\Sigma\,(\mathtt{B}\times\Sigma\,\mathtt{B}\times\mathtt{N})$, gives a well-defined ergodic system. Nevertheless, it demonstrates that recursive samplers can be implemented with no additional language features.

Markov chain Monte Carlo. As in the case of recursion, basic Metropolis-Hastings samplers *can* be implemented in our sampling language via highly nonstandard use of the construct `prng`. For example, *listing* 5.2 implements a sketch of a symmetric Metropolis sampler on the real numbers with proposal q, targeting the (unnormalised) density p.

```
map(λz : T. fst(cast⟨R × ΣR × ΣR+⟩(z)),
  prng(λ(x, q, u) : T .
       let x′ = x + hd(q) in
       let α = p(x′) / p(x) in
       let accept = hd(u) ≤ α in
         if accept then
           (x′, tl(q), tl(u))
         else
           (x, tl(q), tl(u)),
     (x1, q, u)))
  : ΣR
```

Listing 5.2: Random-walk Metropolis sampling

Note first that the continuity conditions necessary to verify listing 5.2, i.e. the form of the subtype $\mathtt{T} \triangleleft \mathtt{R}\times\Sigma\,\mathtt{R}\times\Sigma\,\mathtt{R}^{+}$, will be quite complex. Next, while this procedure uses standard operations in our language, it is not immediately clear to the author whether uses of `prng` like the above, which define a dynamical system over samplers, can straightforwardly be shown to be ergodic, and so

whether it is possible to argue naturally using our targeting calculus that they give asymptotically valid samples. That studying these dynamical systems over samplers would yield a tractable way in which to port the standard proof of convergence of the Metropolis-Hastings algorithm to a deterministic setting was not clear to the author.

Sequential Monte Carlo. Sequential Monte Carlo is perhaps a less natural fit for our framework. While a sketch can easily be implemented using our vector operations, the discontinuities usually present in resampling will pose challenges. Resampling, in our setting, can be understood as an operation $\texttt{resample} : \Pi n : \texttt{N} \,.\, \texttt{Vec}_n(\Sigma\,\texttt{X}) \times \Sigma\,\texttt{R}^+ \to \texttt{Vec}_n(\Sigma\,\texttt{X})$, inputting n weighted samplers of type `X` and a (typically) unweighted uniform sampler, and outputting (typically) unweighted samplers of type `X`. As this operation transforms the samples and weights of its input samplers jointly in a unique manner, it must be implemented as a built-in sampler operation. The challenge for us in integrating resampling is that standard multinomial resampling as typically interested is a discontinuous operation, which will complicate its definition, typing, and verification. Continuity aside, though, demonstrating a corresponding inference rule for resampling in the style of fig. 3.6 does not seem to be challenging.

Functional stochastic process samplers. In Remark 4.1.5, we introduced the possibility of formalising samplers for stochastic processes as samplers of function type, rather than as systems of marginal samplers. We dismissed that possibility as not reasonable from a computational perspective, but despite its impracticality, it is nevertheless a useful perspective to take on stochastic process samplers. For example, our operations `spmap` and `spreweight` correspond simply to `map` and `reweight` operations on these functional samplers – in the sense of satisfying a commuting square, completed by the natural projection map from a functional sampler to the corresponding marginal sampler shown in listing 4.1. This correspondence hints at the possibility of characterising a broader class of operations on marginal samplers, as precisely those operations

which are obtainable as the finite-dimensional equivalents of probabilistically sensible operations on functional samplers.

Convergence, black-box verification, and goodness of fit. Our targeting calculus centres around weak convergence – that is, showing that the samples produced by our samplers can asymptotically estimate expectations of any continuous function. We made this choice because weak convergence has convenient topological properties and, crucially, is guaranteed by most pseudorandom number generators. However, studying convergence with respect to other metrics, such as uniform convergence over a well-chosen class of functions (e.g. a kernelised maximum mean discrepancy (MMD) that metrises weak convergence; see [71]) can provide a natural way to link white-box and black-box verification, using a well-designed one-sample goodness-of-fit test.

Consider the problem of generating samples which target a probability measure P on the Polish space Z. Choose a space of measurable functions $\mathcal{F} \subseteq [X, \mathbb{R}]$ equipped with a measure μ. Then, choose a $p \in [1, \infty]$, and consider the (in general extended) pseudometric

$$d(P,Q) = \begin{cases} \left(\int_{\mathcal{F}} |P(f) - Q(f)|^p \, d\mu(f)\right)^{1/p} & p < \infty \\ \operatorname{ess\,sup}_{f \in \mathcal{F}} |P(f) - Q(f)| & p = \infty \end{cases}$$

on the space of measures $\mathcal{P}X$, where $P(f) = \int_X f(x)\, dP(x)$. If this class of functions $\mathcal{F}$ and the desired target distribution P jointly have certain properties studied in empirical process theory (in particular the Glivenko-Cantelli and Donsker properties; see e.g. [72]), then it follows that $d(P, \hat{P}_n)$, where $\hat{P}_n = \frac{1}{n}\sum_{i=1}^n \delta_{x_i}$ is the empirical measure produced by i.i.d. sampling $x_i \sim P$ from the target distribution, converges in probability to zero, with a corresponding central limit theorem. This enables us to compute p-values for the goodness-of-fit test[1] under the null hypothesis of 'perfect sampling' $x_i \sim P$

[1]The majority of goodness-of-fit tests used – from the chi-squared test on discrete probability measures, to the Kolmogorov-Smirnov and Cramer-von Mises tests on measures on $\mathbb{R}$, to kernelised Stein discrepancies [73] and other kernelised MMDs – arise as special cases of this construction. The primary examples of tests that are not of this form are tests arising

from the target distribution, which can serve as a proxy for a sampler's black-box correctness.

However, characterising the convergence of $d(P, \hat{P}_n)$ under 'perfect' i.i.d. sampling does *not* necessarily indicate that the outputs of any particular approximate sampler should be expected to pass the corresponding statistical test. Justifying the application of goodness-of-fit tests to the outputs of approximate sampling techniques such as sequential Monte Carlo, Markov chain Monte Carlo, or even simple importance sampling, appears challenging. (Rejection sampling, at least, is straightforward: in general, provided that its proposal distribution passes goodness-of-fit tests, its returned samples must as well.) Sensibly applying goodness-of-fit tests to such samplers would require a proof that the samples produced satisfy a generalised Donsker theorem, i.e. functional central limit theorem, with respect to a chosen class of test functions – a difficult proposition. The question of when functional central limit theorems hold for dependent samples is deep and complex; for an overview of results in this area, see [74].

Regardless of whether goodness-of-fit tests for approximate sampling techniques can be justified in this manner, though, compositionality is still not guaranteed. For example, consider the case of an MMD-style discrepancy, i.e. $p = \infty$: knowing that a sampler s produces samples x_n which approximate the target distribution P in the sense that $\lim_{N\to\infty} \sup_{f\in\mathcal{F}} \left| \frac{1}{N} \sum_{n=1}^{N} f(x_n) - \int_X f(x)\, dP(x) \right| = 0$, does not give a corresponding result relating $g(x_n)$ to the pushforward $g_*(P)$ unless it is known that the class $\mathcal{F}$ is closed under precomposition by g (and the same applies to the Donsker property). The natural choice of the 1-Wasserstein distance, for example, in which case $\mathcal{F}$ is the collection of functions with Lipschitz constant less than 1, makes $g(x) = \exp(x)$ and $g(x) = x^2$ inadmissible. Thus, verification using such a procedure would be far from automatic. A class of test functions $\mathcal{F}$ must be chosen by proceeding backwards from how samples

from f-divergences, such as the Kullback-Leibler divergence.

from a given sampler will be used, and the assumptions necessary for a generalised Donsker theorem proven, in order to enable black-box verification of that sampler. A general-purpose approach like the one taken here, aimed at producing samplers which are correct by construction under only weak assumptions about their intended usage, may not be possible unless the built-in functions expressible in the language are heavily restricted (e.g. to Lipschitz functions only).

Bibliography

[1] John von Neumann. *Various techniques for use in connection with random numbers: Notes by G. E. Forsythe*, volume 12 of *National Bureau of Standards: Applied Mathematics Series*. U. S. Government Printing Office, 1951.

[2] Roger Eckhardt. Stan Ulam, John von Neumann, and the Monte Carlo method. *Los Alamos Science*, 15:131–136, 1987.

[3] Andrew Thomas. BUGS: a statistical modelling package. *RTA/BCS Modular Languages Newsletter*, 2:36–38, 1994.

[4] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[5] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, IFL 2016, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv e-prints*, page arXiv:1404.0099, March 2014.

[7] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[8] David Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. WinBUGS - a Bayesian modeling framework: Concepts, structure and extensibility. *Statistics and Computing*, 10:325–337, 10 2000.

[9] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

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

[11] Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. Affine monads and lazy structures for Bayesian programming. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023.

[12] Sam Staton, Frank Wood, Hongseok Yang, Chris Heunen, and Ohad Kammar. 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)*, pages 1–10. IEEE, 2016.

[13] Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

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

[15] Fredrik Dahlqvist, Alexandra Silva, and William Smith. Deterministic stream-sampling for probabilistic programming: semantics and verification. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2023.

[16] Paul R Halmos. *Measure theory*. Springer, 2013.

[17] Andrey N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Pub Co, 2 edition, June 1960.

[18] H. Lebesgue. Sur une généralisation de l'intégrale définie. *Comptes rendus de l'Académie des Sciences*, 132:1025–1027, 1901.

[19] Alexander S. Kechris. Classical descriptive set theory. *Graduate Texts in Mathematics*, 156:1490–1491, 2012.

[20] Robert J. Aumann. Borel structures for function spaces. *Illinois Journal of Mathematics*, 5:614–630, 1961.

[21] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[22] M. Capinski and P.E. Kopp. *Measure, Integral and Probability*. Springer Undergraduate Mathematics Series. Springer London, 2013.

[23] Walter Rudin. *Real and complex analysis, 3rd ed.* McGraw-Hill, Inc., USA, 1987.

[24] Subhashis Ghosal and Aad van der Vaart. *Fundamentals of Nonparametric Bayesian Inference*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2017.

[25] Marcelo Viana and Krerley Oliveira. *Foundations of Ergodic Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2016.

[26] Richard M Dudley. *Real analysis and probability*. CRC Press, 2018.

[27] H. Weyl. Über die gibbs'sche erscheinung und verwandte konvergenzphänomene. *Rendiconti del Circolo Matematico di Palermo*, 330:377–407, 1910.

[28] P. Bohl. Über ein in der theorie der säkularen störungen vorkommendes problem. *J. reine angew. Math*, 135:189–283, 1909.

[29] W. Sierpinski. Sur la valeur asymptotique d'une certaine somme. *Bull Intl. Acad. Polonaise des Sci. et des Lettres*, Series A:9–11, 1910.

[30] Olav Kallenberg. *Foundations of modern probability*. Springer, 1997.

[31] Paul André Meyer. Les processus stochastiques de 1950 à nos jours. 2000.

[32] Bernt Øksendal. *Stochastic Differential Equations (5th Ed.): An Introduction with Applications*. Springer-Verlag, Berlin, Heidelberg, 2010.

[33] Peter Orbanz. Conjugate projective limits. *arXiv: Statistics Theory*, 2010.

[34] Michel Métivier. Limites projectives de mesures. Martingales. Applications. *Annali di Matematica Pura ed Applicata*, 63(1):225–352, 1963.

[35] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

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

[37] Michael C McCord. Classifying spaces and infinite symmetric products. *Transactions of the American Mathematical Society*, 146:273–298, 1969.

[38] Lemoine Gaunce Lewis. *The stable category and generalized Thom spectra. Appendix A*. PhD thesis, University of Chicago, Department of Mathematics, 1978.

[39] Kyriakos Keremedis, Cenap Özel, Artur Piękosz, Mohammed Al Shumrani, and Eliza Wajch. Compact complement topologies and k-spaces. *arXiv preprint arXiv:1806.10177*, 2018.

[40] Sebastiano Vigna. Further scramblings of Marsaglia's xorshift generators. *Journal of Computational and Applied Mathematics*, 315:175–181, 2017.

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

[42] M. Fushimi and Shu Tezuka. The k-distribution of generalized feedback shift register pseudorandom numbers. *Communications of the ACM*, 26:516–523, 07 1983.

[43] Gunther Leobacher and Friedrich Pillichshammer. *Introduction to quasi-Monte Carlo integration and applications*. Springer, 2014.

[44] Lauwerens Kuipers and Harald Niederreiter. *Uniform distribution of sequences*. Courier Corporation, 2012.

[45] Patrick Billingsley. *Convergence of probability measures*. John Wiley & Sons, 2013.

[46] Ulrich Schaechtle, Ben Zinberg, Alexey Radul, Kostas Stathis, and Vikash K. Mansinghka. Probabilistic programming with Gaussian process memoization. *CoRR*, abs/1512.05665, 2015.

[47] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[48] Daniel Roy, Vikash Mansinghka, and Noah Goodman. A stochastic programming perspective on nonparametric Bayes. 09 2010.

[49] Hongwei Xi. Dependent ML: an approach to practical programming with dependent types. *Journal of Functional Programming*, 17(2):215–286, 2007.

[50] Francesca Cagliari, Sandra Mantovani, and EM Vitale. Regularity of the category of Kelley spaces. *Applied Categorical Structures*, 3:357–361, 1995.

[51] Robert AG Seely. Locally cartesian closed categories and type theory. In *Mathematical proceedings of the Cambridge philosophical society*, volume 95, pages 33–48. Cambridge University Press, 1984.

[52] J Peter May and Johann Sigurdsson. *Parametrized homotopy theory*. Number 132. American Mathematical Soc., 2006.

[53] Peter I Booth and Ronald Brown. Spaces of partial maps, fibred mapping spaces and the compact-open topology. *General Topology and its Applications*, 8(2):181–195, 1978.

[54] Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors. *Foundations of Probabilistic Programming*. Cambridge University Press, 2020.

[55] Fredrik Dahlqvist, Dexter Kozen, and Alexandra Silva. *Semantics of Probabilistic Programming: A Gentle Introduction*, pages 1–42. Cambridge University Press, 2020.

[56] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based upon sampling functions. *ACM SIGPLAN Notices*, 40(1):171–182, 2005.

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

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

[59] Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 46(3):413–450, 2012.

[60] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 309–320, 2014.

[61] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic PCF. *Journal of the ACM (JACM)*, 65(4):1–44, 2018.

[62] Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

[63] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 154–165, 2002.

[64] Claire Jones and Gordon D Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 186–187. IEEE Computer Society, 1989.

[65] Johannes Borgström, Ugo Dal Lago, Andrew D Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. *ACM SIGPLAN Notices*, 51(9):33–46, 2016.

[66] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, page 109–116, Arlington, Virginia, USA, 2010. AUAI Press.

[67] Claus-Peter Schnorr. A unified approach to the definition of random sequences. *Mathematical systems theory*, 5(3):246–258, 1971.

[68] Mathieu Hoyrup and Cristòbal Rojas. Computability of probability measures and Martin-Löf randomness over metric spaces. *Information and Computation*, 207(7):830–847, 2009.

[69] Cameron Foreman, Richie Yeung, and Florian J. Curchod. Statistical testing of random number generators and their improvement using randomness extraction. *Entropy*, 26(12), 2024.

[70] Adam Ścibior. *Formally justified and modular Bayesian inference for probabilistic programs*. PhD thesis, University of Cambridge, UK, 2019.

[71] Carl-Johann Simon-Gabriel, Alessandro Barp, Bernhard Schölkopf, and Lester Mackey. Metrizing weak convergence with maximum mean discrepancies. *Journal of Machine Learning Research*, 24(184):1–20, 2023.

[72] Monroe D Donsker. Justification and extension of Doob's heuristic approach to the Kolmogorov-Smirnov theorems. *The Annals of mathematical statistics*, pages 277–281, 1952.

[73] Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 276–284, New York, New York, USA, 20–22 Jun 2016. PMLR.

[74] Jérôme Dedecker and Clémentine Prieur. An empirical central limit theorem for dependent sequences. *Stochastic Processes and their Applications*, 117(1):121–142, January 2007.