A Type Theory for Probability Density Functions

Sooraj Bhat
Georgia Institute of Technology
sooraj@gatech.edu

Ashish Agarwal
New York University
ashish.agarwal@nyu.edu

Richard Vuduc, Alexander Gray
Georgia Institute of Technology
{richie,agray}@cc.gatech.edu

Abstract

There has been great interest in creating probabilistic programming languages to simplify the coding of statistical tasks; however, there still does not exist a formal language that simultaneously provides (1) continuous probability distributions, (2) the ability to naturally express custom probabilistic models, and (3) probability density functions (PDFs). This collection of features is necessary for mechanizing fundamental statistical techniques. We formalize the first probabilistic language that exhibits these features, and it serves as a foundational framework for extending the ideas to more general languages. Particularly novel are our type system for *absolutely continuous* (AC) distributions (those which permit PDFs) and our PDF calculation procedure, which calculates PDFs for a large class of AC distributions. Our formalization paves the way toward the rigorous encoding of powerful statistical reformulations.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages–Program analysis; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs–Mechanical verification; G.3 [*Probability and Statistics*]: Statistical computing

General Terms Theory, Languages

Keywords Continuous Probability, Probability Density Functions

1. Introduction

In the face of more complex data analysis needs, both the machine learning and programming languages communities have recognized the need to express probabilistic and statistical computations *declaratively*. This has led to a proliferation of probabilistic programming languages [7, 9, 12–14, 16, 18–22]. Program transformations on probabilistic programs are crucial: many techniques for converting statistical problems into efficient, executable algorithms are syntactic in nature. A rigorous language definition aids reasoning about the correctness of these program transformations.

However, several fundamental statistical techniques cannot currently be encoded as program transformations because current languages have weak support for probability distributions on continuous or hybrid discrete-continuous spaces. In particular, no existing language rigorously supports expressing the *probability density function* (PDF) of custom probability distributions. This is an impediment to mechanizing statistics; continuous distributions and their PDFs are ubiquitous in statistical theory and applications. Techniques such as maximum likelihood estimation (MLE), L_2 estimation (L2E), and nonparametric kernel methods are all formulated in terms of the PDF [3, 23, 24]. Specifically, we want the ability to naturally express a probabilistic model over a discrete, continuous or hybrid space and then mechanically obtain a usable form of its PDF. Usage of the PDF may entail direct numerical evaluation of the PDF or symbolic manipulation of the PDF and its derivatives. Continuous spaces pose some unique obstacles, however. First, the existence of the PDF is not guaranteed, unlike the discrete case. Second, stating the conditions for existence involves the language of measure theory, an area of mathematics renowned for nonconstructive results, suggesting that mechanization may not be straightforward. Notably, obtaining a PDF from its distribution is a non-computable operation in the general case [11]. In light of these issues, we make the following new contributions:

- We present a formal probability language with classical measure-theoretic semantics which allows naturally expressing a variety of useful probability distributions on discrete, continuous and hybrid discrete-continuous spaces, as well as their PDFs when they exist (Section 3). The language is a core calculus which omits functions and mutation.
- We define a type system for *absolutely continuous* probability distributions, *i.e.* those which permit a PDF. The type system does not require mechanizing σ-algebras, null sets, the Lebesgue measure, or other complex constructions from measure theory. The key insight is to analyze a distribution by how it transforms other distributions instead of using the "obvious" induction on the monadic structure of the distribution (Section 4).
- We define a procedure that calculates PDFs for a large class of distributions accepted by our type system. The design permits modularly adding knowledge about individual distributions with known PDFs (but which cannot be calculated from scratch), enabling the procedure to proceed with programs that use these distributions as subcomponents (Section 5).

We believe this is the first general treatment of PDFs in a language. We deliberately omit features that are not essential to the current investigation (*e.g.* expectation, sampling). Finally, we discuss the relation to existing and future work (Sections 6 and 7). In particular, we save a treatment of PDFs in the context of conditional probability for future work.

2. Background and motivation

We first introduce probability in the context of countable spaces to emphasize the complications that arise when moving to continuous spaces. We focus only on issues surrounding PDFs. We occasionally deviate from standard probability notation to circumvent imprecision in the standard notation and to create a harmonious notation throughout the paper. In this section we present a specialized ac-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.
Copyright © 2012 ACM 978-1-4503-1083-3/12/01. . . $10.00

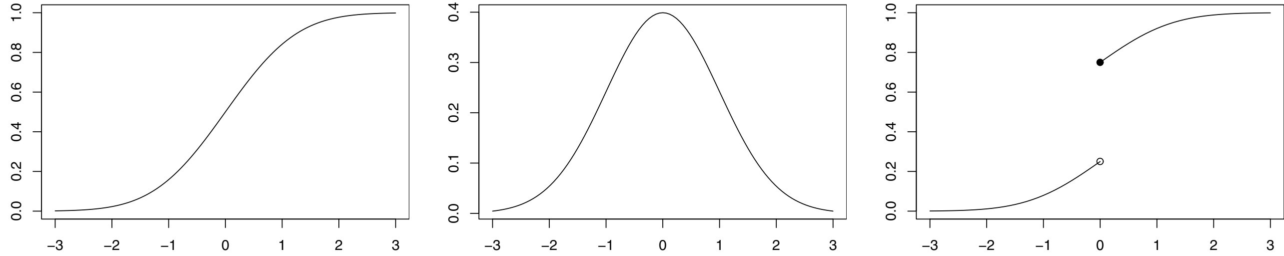

Figure 1. The CDF and PDF of a standard normal distribution and the CDF of a distribution that does not have a PDF.

count of probability for ease of exposition. We discuss the rigorous and generalized definitions in Section 3.3.

We use the term *discrete distribution* for distributions on discrete spaces (countable sets); *continuous distribution* for distributions on the continuous spaces $\mathbb{R}$ and $\mathbb{R}^n$; and *hybrid distribution* for distributions on products of discrete and continuous spaces that are themselves neither discrete nor continuous, such as $\mathbb{R} \times \mathbb{Z}$.

2.1 Probability on countable spaces

Consider a set of outcomes A. For now, let A be countable. It is meant to represent the possible states of the world we are modeling, such as the set of possible outcomes of an experiment or measurements of a quantity. An *event* is a subset of A, also understood as a predicate on elements of A. Events denote some occurrence of interest and partition the outcomes into those that exhibit the property and those that do not. A *probability distribution* $\mathbb{P}$ (or simply, *distribution*) on A is a function from events to $[0, 1]$ such that $\mathbb{P}(X) \geq 0$ for all events X, $\mathbb{P}(A) = 1$ and $\mathbb{P}(\bigcup_{i=0}^{\infty} X_i) = \sum_{i=0}^{\infty} \mathbb{P}(X_i)$ for countable sequences of mutually disjoint events X_i. Distributions tell us the probability that different events will occur. It is generally more convenient to work with a distribution's *probability mass function* (PMF) instead, defined $f(x) = \mathbb{P}(\{x\})$, which tells us how likely an individual outcome is. It satisfies

$$\mathbb{P}(X) = \sum_{x \in X} f(x)$$

for all events X on A. For example, if $\mathbb{P}$ is the distribution characterizing the outcome of rolling a fair die, its PMF is given by $f(x) = \frac{1}{6}$, where $x \in A$ and $A = \{1, 2, 3, 4, 5, 6\}$. The probability an even number is rolled is $\mathbb{P}(\{2, 4, 6\}) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$.

2.2 Moving to continuous spaces

A *probability density function* (PDF) is the continuous analog of the PMF. Unfortunately, although every distribution on a countable set has a PMF, not every distribution on a continuous space has a PDF. Consider distributions on the real line. We say that a function f is a PDF of a distribution $\mathbb{P}$ on $\mathbb{R}$ if for all events X,

$$\mathbb{P}(X) = \int_X f(x)\, dx, \tag{1}$$

which states that the probability of X is the integral of f on X (in the simplest case, X is an interval). This idea can be extended to more general spaces. This equation does not determine f uniquely, but any two solutions f_1 and f_2 are equal "almost everywhere" (see Section 3.3) and give identical results under integration. Thus, we often refer to a PDF as *the* PDF.

For the spaces we consider in this paper, the property of having a PDF is equivalent to being *absolutely continuous* (AC). Roughly speaking, a distribution is AC if it never assigns positive probability to events that have "size zero" in the underlying space. For instance, the standard normal distribution is absolutely continuous and has the PDF $\phi(x) = \exp(-x^2/2)/\sqrt{2\pi}$. On the other hand, the distribution of y in the model

$$\begin{aligned} z &\sim \text{CoinFlip}(1/2) \\ x &\sim \text{Normal}(0, 1) \end{aligned} \qquad y = \begin{cases} 0 & \text{if } z = \text{heads} \\ x & \text{if } z = \text{tails.} \end{cases}$$

does not have a PDF. We have used *random variables* to write the model; this is a commonly used informal notation that is shorthand for a more rigorous expression that defines the model. The model represents the following process: flip a fair coin; return 0 if it is heads, and sample from the standard normal distribution otherwise. We can see that it is not AC: the event $\{0\}$ occurs with probability $1/2$ (whenever the coin comes up heads), but has an interval length of zero. We use the *cumulative distribution function* (CDF) to visualize each distribution (Figure 1); the CDF F of a distribution $\mathbb{P}$ on $\mathbb{R}$ is $F(x) = \mathbb{P}(\ (-\infty, x]\ )$ and gives the probability that a sample from the distribution takes a value less than or equal to x. From Equation 1 we know that $\mathbb{P}$ has a PDF if and only if there exists a function f such that $F(x) = \int_{-\infty}^{x} f(t)\, dt$. Clearly, no such function exists for the CDF of y due to the jump discontinuity.

Mixing discrete and continuous types is not the only culprit. Consider the following process: sample a number u uniformly randomly from $[0, 1]$ and return the vector $x = (u, u) \in \mathbb{R}^2$. The distribution of x (a distribution on $\mathbb{R}^2$) is not AC: the event $X = \{(u, u) \mid u \in [0, 1]\}$ has probability 1, but X is a line segment and thus has zero area. Likewise, there is no PDF on $\mathbb{R}^2$ we could integrate to give positive mass on this zero area line segment.

2.3 Applications of the PDF

The PDF is often used to compute expectations (and probabilities, which are a special case of expectation). Expectation is a fundamental operation in probability and is used in defining quantities such as mean and variance. The expectation operation $\mathbb{E}$ of a distribution $\mathbb{P}$ on $\mathbb{R}$ is a higher-order function that satisfies

$$\mathbb{E}(g) = \int_{-\infty}^{\infty} g(x) \cdot f(x)\, dx$$

when $\mathbb{P}$ has a PDF, f, and the integral exists. Another application is *maximum likelihood estimation* (MLE), which addresses the problem of choosing the member of a family of distributions that "best explains" observed data. Let $\mathbb{P}(\,\cdot\,;\,\cdot\,)$ be a parameterized family of distributions, where $\mathbb{P}(\,\cdot\,;\theta)$ is the distribution for a given parameter θ. The MLE estimate θ^* of $\mathbb{P}$ for observed data x is given by

$$\theta^* = \arg\max_{\theta} f(x; \theta)$$

where $f(\,\cdot\,;\theta)$ is the PDF of $\mathbb{P}(\,\cdot\,;\theta)$. For example, x could be a set of points in $\mathbb{R}^n$ we wish to cluster, and θ^* could be the estimate of the locations of the cluster centroids. $\mathbb{P}$ would be the family of distributions we believe generated the clusters (a family parameterized by the positions of the cluster centroids), such as a mixture-of-Gaussians model. More details are available in [3].

2.4 Challenges for language design

Categorically, probability distributions form a monad [8, 20]. This structure forms the basis of many probabilistic languages because it is minimal, elegant, and presents many attractive features. First, it provides the look and feel of informal random variable notation, allowing us to express models as we normally would, while remaining mathematically rigorous. The monad structure affords formulating probability as an embedded domain specific language [7, 13, 18, 20] or as a mathematical theory in a proof assistant [2]. Additionally, many proofs about distributions expressed in the probability monad are greatly simplified by the monadic structure. We feel it is desirable to structure a language around the probability monad, and we investigate supporting PDFs specifically in such languages.

The probability monad consists of monadic return and monadic bind, as usual. Monadic return corresponds to the point mass distribution. We also provide the Uniform(0,1) distribution as a monadic value. These three combinators can be used to express a variety of distributions. The main issue when designing a type system for absolute continuity is that return creates non-AC distributions (on continuous types), yet—as a core member of the monadic calculus—it appears in the specification of nearly every distribution, even those that *are* AC. The "obvious" induction along the monadic structure is difficult to use to prove absolute continuity in the cases of interest. Consider for instance the joint distribution of two independent Uniform(0,1) random variables, written in our language as

$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (x, y). \qquad (2)$$

It is AC even though the subexpressions return (x, y) and var $y \sim$ random in return (x, y) are both *not* AC, where we treat x and y as real numbers, as dictated by the typing rule for bind. Also, as implementors we have found it difficult to "eyeball" the rules for absolute continuity. For example, only the first of these distributions is AC even though they are all nearly identical to Equation 2:

$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (x, x+y)$$
$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (x, y-y)$$
$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (x, y, x+y)$$

Clearly, what is needed is a principled analysis. We provide this in Section 4. A natural urge is wanting to remove return to create a language in which only AC distributions are expressible. We feel this is undesirable. Without return, we would not be able to express something as simple as adding two random variables (consider $(x + y)$ instead of (x, y) in Equation 2). Essentially, return allows us to express random variables as transformations of other random variables—a fundamental modeling tool we feel should be supported, allowing users to write down models that most naturally capture their domain. Without return we must extend the core calculus for each transformation we wish to use on random variables, and we must do so carefully if we want to ensure that non-AC distributions remain inexpressible. This extension of the core detracts from minimality and elegance, and it complicates developing the theory in a verification environment such as COQ, one of our eventual goals.

Finally, in addition to checking for existence, we would like to also calculate a usable form for the PDF. Many current probabilistic languages focus on distributions with only *finitely* many alternatives, which allows for implementing distributions as weighted lists of outcomes. The probability monad in this case is similar to the list monad, with some added logic describing how bind should propagate the weights. The weighted lists correspond directly to the PMF, but no such straightforward computational strategy exists for the PDF. We explore this further in Section 5.

Variables	x, y, z, u, v Literals l
Base types	$\tau ::= \mathsf{bool} \mid \mathsf{Z} \mid \mathsf{R} \mid \tau_1 \times \tau_2$
Types	$t ::= \tau \mid \mathsf{dist}\ \tau$
Expressions	$\varepsilon ::= x \mid l \mid op\ \ \varepsilon_1 ... \varepsilon_n \mid \mathsf{if}\ \varepsilon_1\ \mathsf{then}\ \varepsilon_2\ \mathsf{else}\ \varepsilon_3$
Primops	$op ::= + \mid * \mid \mathsf{neg} \mid \mathsf{inv} \mid = \mid < \mid (\cdot,\cdot) \mid \mathsf{fst} \mid \mathsf{snd}$ $\mid \mathsf{exp} \mid \mathsf{log} \mid \mathsf{sin} \mid \mathsf{cos} \mid \mathsf{tan} \mid \mathsf{R_of_Z}$
Distributions	$e ::= \mathsf{random} \mid \mathsf{return}\ \varepsilon \mid \mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2$
Programs	$p ::= \mathsf{pdf}\ e$
Contexts	$\Gamma ::= \emptyset \mid \Gamma, x : \tau$ $\Upsilon ::= \emptyset \mid \Upsilon, x : \tau \sim e$
Substitution	$e[x := \varepsilon]$ Free variables $FV(\cdot)$

Figure 2. The abstract syntax.

3. The language

In this section we present the abstract syntax, type system and semantics for our probabilistic language, except for the parts related to PDFs, which we cover in Section 4.

3.1 Abstract syntax

Figure 2 contains the syntax definitions. In addition to the standard letters for variables, we also use u and v when we want to emphasize that a random variable is distributed according to the Uniform(0,1) distribution. The syntactic category for literals includes Boolean (bool), integer (Z), and real number (R) literals. Types are stratified to ensure that distributions (dist τ) are only over base types. Integers are a distinct type from the reals; there is no subtyping in the language. We also stratify terms to simplify analysis. Expressions and primitive operations (primops) take their standard mathematical meaning, unless noted otherwise. For simplicity, we overload addition, multiplication, negation, and integer literals on the integers and reals, but fundamentally there is a $+$ for integers and a separate $+$ for reals, *etc*. Inversion inv denotes the reciprocal operation, and log is the natural logarithm. We give our semantics in terms of classical mathematics, so we do not concern ourselves with the issue of computation on the reals. Equality is defined on all base types in the usual way, and less-than is defined only on the numeric types. We write $(\varepsilon_1, \varepsilon_2, ..., \varepsilon_{n-1}, \varepsilon_n)$ as shorthand for $(\varepsilon_1, (\varepsilon_2, ...(\varepsilon_{n-1}, \varepsilon_n)...))$. The function R_of_Z injects an integer into the reals. The distribution random corresponds to the Uniform(0,1) distribution. The next two constructs correspond to monadic return and bind for the probability monad. The distribution return ε is the *point mass distribution*, which assigns probability 1 to the event $\{\varepsilon\}$. A random variable distributed according to return ε is in fact deterministic: there is no variation in the value it can take. The bind construct, var $x \sim e_1$ in e_2, is used to build complex distributions from simpler ones. It can be read: "introduce random variable x, distributed according to the distribution e_1, with scope in the distribution e_2". It is the only binding construct in the language. For simplicity, we have chosen to omit let-bindings and functions from our language, but we use both in our examples. We can use standard substitution rules to reduce such examples to the syntax of Figure 2. Examples include $\langle\varepsilon\rangle := \mathsf{if}\ \varepsilon\ \mathsf{then}\ 1\ \mathsf{else}\ 0$ (to inject Booleans into the reals), $\varepsilon_1 - \varepsilon_2 := \varepsilon_1 + \mathsf{neg}\ \varepsilon_2$, $\varepsilon_1/\varepsilon_2 := \varepsilon_1 * \mathsf{inv}\ \varepsilon_2$, and $\varepsilon_1 < \varepsilon_2 < \varepsilon_3 := \mathsf{if}\ \varepsilon_1 < \varepsilon_2\ \mathsf{then}\ \varepsilon_2 < \varepsilon_3\ \mathsf{else\ false}$. Finally, free variables, capture-avoiding substitution, and typing contexts (Γ) are defined in the usual way. The probability context Υ is used to additionally keep track of the distributions that random variables

are bound to. When we use Υ in places Γ is expected, the understanding is that the extra information carried by Υ is ignored.

3.2 Examples of expressible distributions

With just random, return, and bind, we can already construct a wide variety of distributions we might care to use in practice. Though we do not have a formal proof of this expressivity, existing work on sampling suggests that this is the case. Non-uniform random variate generation is concerned with generating samples from arbitrary distributions using only samples from Uniform(0,1) [6]. We can see the connection with our language if we view the constructs by a sampling analogy, which emphasizes understanding a distribution by its generating process: the phrase var $x \sim e_1$ in e_2 samples a value x from the sampling function e_1, which is used to create a new sampling function e_2; random samples from the Uniform(0,1) distribution; return ε always returns ε as its sample. For instance, the *standard normal distribution* can be defined in our language using the Box-Muller sampling method:

$$\mathsf{std_normal} := \begin{array}{l}\mathsf{var}\ u \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ v \sim \mathsf{random}\ \mathsf{in} \\ \quad \mathsf{return}\ (\mathsf{sqrt}(-2 * \log u) * \cos(2 * \pi * v))\end{array}$$

where $\mathsf{sqrt}\ \varepsilon := \exp\ ((1/2) * \log \varepsilon)$. In particular, our language is amenable to *inverse transform sampling*. Likewise, we can express other common continuous distributions:

```
uniform ε1 ε2 :=                       std_logistic :=
  var u ~ random in                      var u ~ random in
   return ((ε2 − ε1) ∗ u + ε1)            return (log (1/u − 1))

normal ε1 ε2 :=                        std_exponential :=
  var x ~ std_normal in                  var u ~ random in
   return (ε2 ∗ x + ε1)                   return (−log u)
```

These are the Uniform(a,b), standard logistic, standard exponential, and Normal(μ, σ) distributions. We intentionally parameterize the normal distribution by its standard deviation instead of its variance, for simplicity. We define it as a transformation of a standard normal random variable and require $\varepsilon_2 > 0$. We can also express discrete distributions, such as the coin flip distribution,

$$\mathsf{flip}\ \varepsilon := \mathsf{var}\ u \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (u < \varepsilon),$$

which takes the value true with probability ε. This is equivalent to the Bernoulli distribution. In fact, we can express any distribution with finitely many outcomes:

```
var u ~ random in
   return (if u < 1/3 then 10 else if u < 2/3 then 20 else 30)
```

Admittedly, a more satisfying definition would be possible if we had lists in the language. The reason we do not is that, although others have addressed recursion and iteration in the context of defining probability distributions [14], we have not yet fully reconciled recursion with PDFs. The absence of recursion also means that we do not support distributions in the style of *rejection sampling* methods, which resample values until a stopping criterion is met. Furthermore, we do not elegantly support *infinite* discrete distributions in the core language, many of which are naturally described using recursion. However, in Section 5 we describe how to add special support for any distribution with a known PDF.

We also define some higher-order concepts. The following functions are used to create joint distributions and mixture models:

```
join e1 e2 :=                 mix ε e1 e2 :=
  var x1 ~ e1 in                var z ~ flip ε in
    var x2 ~ e2 in                var x1 ~ e1 in
      return (x1, x2)               var x2 ~ e2 in
                                      return (if z then x1 else x2).
```

The mixture model is created by flipping a coin with the specified probability to determine which component distribution to sample from. For instance, a simple mixture-of-Gaussians is given by mix $(1/2)$ std_normal std_normal. We have defined discrete and continuous distributions, and now we can use join to define nontrivial hybrid distributions as well, such as join (flip $(1/2)$) random, which has type dist (bool $\times$ R). Essentially, if-expressions enable mixture models and tuples enable joint models. These two concepts are special cases of *hierarchical models*, which are models that are defined in stages. Distributions defined using nested instances of bind correspond to hierarchical models.

These examples are all AC, but we can also express non-AC distributions, such as the example from Section 2, written jumpy := mix $(1/2)$ (return 0) std_normal. In our language, jumpy will successfully type check as a distribution, but the program pdf jumpy will be rejected—as it should be—because jumpy is not absolutely continuous. The ability to represent non-AC distributions, even though they cannot be used in programs, is in anticipation of future language features such as expectation and sampling, which *can* be used with non-AC distributions.

3.3 Measure theory preliminaries

Measure theory is the basis of modern probability theory and unifies the concepts of discrete and continuous probability distributions. It is a precise way of defining the notion of volume. We develop our formalization within this framework. We give only a brief overview of the necessary concepts; details are available in [17].

Basics Let A be a set we wish to measure. A *σ-algebra* $\mathcal{M}$ on A is a subset of the powerset $\mathcal{P}(A)$ that contains A and is closed under complement and countable union. The pair $(A, \mathcal{M})$ is a *measurable space*. A subset X of A is *$\mathcal{M}$-measurable* if $X \in \mathcal{M}$. In the context of probability, A is the set of outcomes and $\mathcal{M}$ is the set of events. For a function $f : A \to B$, the *f-image* of a subset X of A, written $f[X]$, denotes the set $\{f(x) \mid x \in A\}$, and the *f-preimage* of a subset Y of B, written $f^{-1}[Y]$, denotes the set $\{x \in A \mid f(x) \in Y\}$. When f is on measurable spaces, we say f is *$(\mathcal{M}_A, \mathcal{M}_B)$-measurable* when the f-preimage of any $\mathcal{M}_B$-measurable set is $\mathcal{M}_A$-measurable. Measurable functions are closed under composition. We drop the prefix and say *measurable* (for functions or sets) when it is clear what the σ-algebras are. The σ-algebra machinery is needed to ensure a consistent theory; there are spaces which contain pathological sets that violate intuition about volume, *e.g.* the Banach-Tarski "doubling ball" paradox. Measure theory sidesteps these issues by preferring measurable sets and functions as much as possible. When A is countable, no such problems arise, and we can always take $\mathcal{P}(A)$ for $\mathcal{M}$.

Measures A nonnegative function $\mu : \mathcal{M} \to \mathbb{R}^+ \cup \{\infty\}$ is a *measure* if $\mu(\emptyset) = 0$, $\mu(X) \geq 0$ for all X in $\mathcal{M}$, and $\mu(\bigcup_{i=1}^{\infty} X_i) = \sum_{i=1}^{\infty} \mu(X_i)$ for all sequences of mutually disjoint X_i (*countable additivity*). The triple $(A, \mathcal{M}, \mu)$ is a *measure space*. If additionally $\mu(A) = 1$ then μ is a *probability measure* (conventionally written $\mathbb{P}$), and the triple is a *probability space*. We use the terms *probability measure*, *probability distribution* and *distribution* interchangeably. We use $\mathfrak{C}$ to denote the counting measure, which uses the number of elements of a set as the set's measure. We use $\mathfrak{L}$ to denote the Lebesgue measure on $\mathbb{R}$, which assigns the length $|b - a|$ to an open interval (a, b); the sizes of other sets can be understood by complements and countable unions of intervals. The *product measure* $\mu_A \otimes \mu_B$ of two measures μ_A and μ_B on measurable spaces $(A, \mathcal{M}_A)$ and $(B, \mathcal{M}_B)$ is the measure μ, on $A \times B$ and the product σ-algebra $\mathcal{M}_A \otimes \mathcal{M}_B$, such that

$$\mu(X \times Y) = \mu_A(X) \cdot \mu_B(Y)$$

for $X \in \mathcal{M}_A$ and $Y \in \mathcal{M}_B$. The measure is unique when μ_A and μ_B are σ-finite. The σ-finiteness condition is a technical condition that is satisfied by all measures we will consider in this paper and

requires that the space can be covered by a countable number of pieces of finite measure.

Null sets A measurable set X is *μ-null* if $\mu(X) = 0$; X is said to have μ-measure zero. The empty set is always null, the only $\mathfrak{C}$-null set is the empty set, and all countable subsets of $\mathbb{R}$ are $\mathfrak{L}$-null. A propositional function holds *μ-almost everywhere (μ-a.e.)* if the set of elements for which the proposition does not hold is μ-null. For instance, two functions of type $\mathbb{R} \to \mathbb{R}$ are equal $\mathfrak{L}$-almost everywhere if they differ at only a countable number of points. A measure space $(A, \mathcal{M}, \mu)$ is *complete* if all subsets of any μ-null set are $\mathcal{M}$-measurable. *Completion* is an operation that takes any measure space $(A, \mathcal{M}, \mu)$ and produces an "equivalent" complete measure space $(A, \mathcal{M}', \mu')$ such that $\mu'(X) = \mu(X)$ for $X \in \mathcal{M}$. Null sets are ubiquitous in measure theory, so it will be handy to work in spaces that support null sets as much as possible. Thus, completion makes measure spaces nicer to work with. The n-dimensional Lebesgue measure $\mathfrak{L}^n$ is the n-fold completed product of $\mathfrak{L}$. For measures μ and ν on a measurable space, ν is *absolutely continuous* with respect to μ if each μ-null set is also ν-null.

Integration A fundamental operation involving measures is the *abstract integral*, a generalization of the Riemann integral that avoids some of its deficiencies. The abstract integral of a measurable function $f : A \to \mathbb{R}$ w.r.t. a measure μ on A is written $\int f \; d\mu$. The integral is always defined for nonnegative f. The integral for arbitrary f is defined in terms of the positive and negative parts of f and may not exist; if it *does* we say f is μ-integrable. We write $\int_X f \; d\mu$ as shorthand for $\int \lambda x \,.\, \mathbf{1}_X(x) \cdot f(x) \; d\mu$, which restricts the integral to the subset X. We write $\mathbf{1}_X$ for the indicator function on X. *Expectation* refers to abstract integration w.r.t. a distribution. The abstract integral satisfies $\mu(X) = \int \mathbf{1}_X \; d\mu$ for all measurable X. In terms of probability, it says that the probability of X is the expectation of $\mathbf{1}_X$. Another consequence is that null sets cannot affect integration: two functions that are equal μ-a.e. give the same results under integration w.r.t. μ. Abstract integration w.r.t. $\mathfrak{C}$ and $\mathfrak{L}$ is ordinary (possibly infinite) summation and the ordinary Lebesgue integral, respectively. The Lebesgue integral agrees with the Riemann integral on Riemann-integrable functions.

Measurability Ordinarily, to conclude that a distribution such as $\mathsf{var}\ x \sim e\ \mathsf{in}\ \mathsf{return}\ (f\ x)$ is well-formed, we are obligated to verify that f is a measurable function. However, non-measurable sets and functions are actually quite pathological and constructing them requires the Axiom of Choice [25]. None of the constructs in our language are as powerful as the Axiom of Choice (though we do not have a formal proof of this), thus all constructible expressions represent measurable functions. This discharges the obligation, and we do not make any further mention of checking for measurability.

Stocked spaces For most applications, we often have a standard idea of how spaces are measured. We now formalize this practice. A space A is a *stocked space* if it comes equipped with a complete measure space $(A, \overline{\mathcal{M}}_A, \overline{\mu}_A)$, which is the *stock measure space* of A. We call $\overline{\mathcal{M}}_A$ the *stock σ-algebra* of A and $\overline{\mu}_A$ the *stock measure* of A. The abstract integral w.r.t. $\overline{\mu}_A$ is the *stock integral* of A. We define stock measure spaces for the spaces $\mathbb{B} = \{\text{true}, \text{false}\}$, $\mathbb{Z}$, $\mathbb{R}$, and product spaces between stocked spaces as follows:

$$\begin{aligned}
(\overline{\mathcal{M}}_{\mathbb{B}}, \overline{\mu}_{\mathbb{B}}) &= (\mathcal{P}(\mathbb{B}), \mathfrak{C}) \\
(\overline{\mathcal{M}}_{\mathbb{Z}}, \overline{\mu}_{\mathbb{Z}}) &= (\mathcal{P}(\mathbb{Z}), \mathfrak{C}) \\
(\overline{\mathcal{M}}_{\mathbb{R}}, \overline{\mu}_{\mathbb{R}}) &= (\text{the } \mathfrak{L}\text{-measurable sets}, \mathfrak{L}) \\
(\overline{\mathcal{M}}_{A \times B}, \overline{\mu}_{A \times B}) &= \text{completion}(\overline{\mathcal{M}}_A \otimes \overline{\mathcal{M}}_B, \overline{\mu}_A \otimes \overline{\mu}_B).
\end{aligned}$$

This definition matches what is used in practice: *e.g.* $\mathfrak{C}$ becomes the measure for countable spaces, and $\mathfrak{L}^n$ becomes the measure for $\mathbb{R}^n$. For the rest of the paper, we assume spaces are stocked, unless explicitly noted otherwise. We say that a distribution on A is AC if it is AC with respect to $\overline{\mu}_A$.

$$\frac{}{\Gamma \vdash \mathsf{random} : \mathsf{dist}\ \mathsf{R}}\ \text{T-RAND} \qquad \frac{\Gamma \vdash \varepsilon : \tau}{\Gamma \vdash \mathsf{return}\ \varepsilon : \mathsf{dist}\ \tau}\ \text{T-RET}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{dist}\ \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \mathsf{dist}\ \tau_2}{\Gamma \vdash \mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2 : \mathsf{dist}\ \tau_2}\ \text{T-BIND}$$

Figure 3. Standard monadic typing rules for distributions.

Densities A function f is a PDF of a distribution $\mathbb{P}$ on A if $\mathbb{P}(X) = \int_X f \; d\overline{\mu}_A$ for all measurable X. Expectation can be written using the PDF:

$$\int g \; d\mathbb{P} = \int \lambda x \,.\, g(x) \cdot f(x) \; d\overline{\mu}_A.$$

A *joint* PDF is the PDF of a *joint distribution*, which is simply a distribution on a product space. We later use the fact that the joint PDF f of a model such as $x_1 \sim \mathbb{P}_1, x_2 \sim \mathbb{P}_2(\cdot\,; x_1)$ can be written as the product of the individual (parameterized) PDFs: $f(x_1, x_2) = f_1(x_1) \cdot f_2(x_2; x_1)$.

3.4 Type system and semantics for distributions

We now discuss the type system and semantics for syntactic categories besides programs. The type system for expressions is ordinary. We assume an external mechanism for enforcing the preconditions necessary to ensure totality of functions, such as an automated theorem prover or the programmer themself. For instance, log must be applied to only positive real numbers. Distributions obey standard monadic typing rules (Figure 3). The "random variables" introduced by bind are really just normal variables and are typed as such; calling them random variables is a reminder about the role they play. The typing rules ensure that random variables are never used outside a probabilistic context.

We give our language a semantics based in classical mathematics with total functions. Base types have the usual meaning. The denotation of $\mathsf{dist}\ \tau$ is the set of distributions possible on τ:

$$\mathcal{T}[\![\mathsf{dist}\ \tau]\!] = \{\mathbb{P} \mid (\mathcal{T}[\![\tau]\!], \overline{\mathcal{M}}_\tau, \mathbb{P}) \text{ is a probability space}\}.$$

We overload stock measure space notation for types; thus, $\overline{\mathcal{M}}_\tau$ and $\overline{\mu}_\tau$ are shorthand for $\overline{\mathcal{M}}_{\mathcal{T}[\![\tau]\!]}$ and $\overline{\mu}_{\mathcal{T}[\![\tau]\!]}$. Let $\mathcal{E}[\![e]\!]\rho$ be the denotation of a distribution e under the environment ρ, also overloaded for expressions ε. Expressions have the semantics of their corresponding forms from classical mathematics. As stated before, random is the Uniform(0,1) distribution

$$\mathcal{E}[\![\mathsf{random}]\!]\rho = \lambda X \,.\, \mathfrak{L}(X \cap [0,1]),$$

which says that the probability of an event X is its "interval size" on [0,1]. Return is the point mass distribution

$$\mathcal{E}[\![\mathsf{return}\ \varepsilon]\!]\rho = \lambda X \,.\, \mathbf{1}_X(\mathcal{E}[\![\varepsilon]\!]\rho),$$

which gives an event X probability 1 as long as it includes the outcome ε. Bind expresses the Law of Total Probability,

$$\mathcal{E}[\![\mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2]\!]\rho = \lambda Y \,.\, \int \lambda x' \,.\, f(x')(Y) \; d\mathbb{P},$$

where $f(x') = \mathcal{E}[\![e_2]\!](\rho\{x \mapsto x'\})$ and $\mathbb{P} = \mathcal{E}[\![e_1]\!]\rho$. The family of distributions e_2 is parameterized by the variable x, in essence. The probability of an event Y is the "average opinion" (the $\mathbb{P}$-expectation) of what each member of the family thinks is the probability of Y. The integral exists because it is the expectation of a bounded function.

4. Type system and semantics for programs

The program $\mathsf{pdf}\ e$ is well-formed if the distribution e permits a PDF. The following theorem gives us a sufficient condition.

Theorem 4.1 (Radon-Nikodym). *For any two σ-finite measures μ and ν on the same measurable space such that ν is absolutely continuous w.r.t. μ, there is a function f such that $\nu(X) = \int_X f\ d\mu$.*

We call f a *Radon-Nikodym derivative* of ν with respect to μ, denoted $d\nu/d\mu$; pdf corresponds to the Radon-Nikodym operator. The condition is also necessary: given a satisfying f, ν is (trivially) AC. All stock measures we define and all distributions are σ-finite, so for our purposes absolute continuity is equivalent to possessing a PDF. Though not necessarily unique, Radon-Nikodym derivatives are equal μ-almost everywhere. When μ is the counting measure, the Radon-Nikodym derivative is a PMF. For hybrid spaces, it is a function which must be summed along one dimension and integrated along the other to obtain quantities interpretable as probabilities. Radon-Nikodym derivatives unify PMFs, PDFs and hybrids of the two. For this reason, we refer to all of these as PDFs. We use "PMF" when we want to emphasize its discrete nature.

Defining a type system for absolute continuity in terms of the straightforward induction on distribution terms proves unwieldy. Suppose we want to check if the distribution in Equation 2 is AC; we must verify that the probability of any $\mathfrak{L}^2$-null set Z is zero. A straightforward induction leads us to trying to show

$$\mathbb{P}(\{x \mid \left(\begin{array}{c}\mathsf{var}\ y \sim \mathsf{random\ in} \\ \mathsf{return}\ (x,y)\end{array}\right)(Z) \neq 0\}) = 0$$

where $\mathbb{P}$ is the Uniform(0,1) distribution, and we have abused notation slightly by mingling object language syntax with ordinary mathematics. This states that the body of the outermost bind assigns Z probability zero, $\mathbb{P}$-almost always. It is unclear how to proceed from here or how to remove concepts like null sets from the mechanization. We take an alternate approach based on the insight that we can reason about a distribution by examining how it transforms other distributions. Our approach, and outline of the following subsections, is as follows:

- We introduce the new notion of a *non-nullifying* function and prove a transformation theorem stating that when a random variable is transformed, the output distribution is AC if the input distribution is AC and the transformation is non-nullifying. We also prove some results about non-nullifying functions.
- We define the *random variables transform* of any distribution written in our language and show that for a large class of distributions the transformation theorem is applicable.
- We present a type system which defines absolute continuity of a distribution in terms of whether its RV transform is non-nullifying. As implementors, we have found it easier to come up with the rules for non-nullifying functions.

Measure-theoretic concepts like σ-algebras, null sets, and the Lebesgue measure, while present in the metatheory, do not need to be operationalized for implementing the type checker. Also, due to the measure-theoretic foundation, we correctly handle cases that are not typically explained, such as PDFs on hybrid spaces. We conclude the section with the semantics of programs.

4.1 Absolute continuity and non-nullifying functions

A function $h : A \to B$ is *non-nullifying* if the h-preimage of each $\overline{\mu}_B$-null set is $\overline{\mu}_A$-null; preimages of null sets are always null, and forward images of non-null sets are always non-null. A function that fails to be non-nullifying is called *nullifying*. The next theorem establishes the link between absolute continuity and non-nullity.

Theorem 4.2 (Transformation). *For a function $h : A \to B$ and an AC distribution $\mathbb{P}$ on A, the distribution*

$$\mathbb{Q}(Y) = \mathbb{P}(h^{-1}[Y]) \tag{3}$$

on B is AC if h is non-nullifying.

Proof. Let Y be a $\overline{\mu}_B$-null set. By the non-nullity of h, the set $h^{-1}[Y]$ is $\overline{\mu}_A$-null. By the absolute continuity of $\mathbb{P}$, we have $\mathbb{P}(h^{-1}[Y]) = 0$, implying that Y is also $\mathbb{Q}$-null. ☐

This style of defining $\mathbb{Q}$ may seem odd, but it actually underlies the use of random variables as a modeling language. For instance, the model $x \sim \mathbb{P},\ y = h(x)$ exhibits the relationship $\mathbb{Q}(Y) = \mathbb{P}(h^{-1}[Y])$, where $\mathbb{Q}$ is the distribution of y. In general, the reverse direction does not hold; h can be nullifying even if $\mathbb{Q}$ is AC. This happens when h has nullifying behavior only in regions of the space where $\mathbb{P}$ is assigning zero probability. This will be a source of incompleteness in the type system.

Lemma 4.3 (Discrete domain). *A function $h : A \to \mathbb{R}$ is nullifying if A is non-empty and countable.*

Proof. Let x be an element of A. The set $\{x\}$ has positive counting measure while its h-image, which is a singleton set, is $\mathfrak{L}$-null. ☐

This implies R_of_Z is nullifying, meaning that when we view an integer random variable as a real random variable, it loses its ability to have a PDF. This is desirable behavior; different spaces have different ideas of what it means to be a PDF. We would not want to mark an integer random variable as AC and later attempt to integrate its PMF in a context expecting a real random variable.

Lemma 4.4 (Discrete codomain). *A function $h : A \to B$ is non-nullifying if B is countable.*

Proof. The h-preimage of the empty set (the only $\mathfrak{C}$-null set) is the empty set, which is always null. ☐

This reasoning corroborates the fact that distributions on countable spaces always have a PMF.

Lemma 4.5 (Interval). *A function $h : \mathbb{R} \to \mathbb{R}$ is nullifying if it is constant on any interval.*

Proof. Let h be constant on (a, b); (a, b) is not $\mathfrak{L}$-null, but its h-image (a singleton set) is $\mathfrak{L}$-null. ☐

One way to visualize how this leads to a non-AC distribution is to observe that the transformation h takes all the probability mass along (a, b) and non-smoothly concentrates it onto a single point in the target space.

Lemma 4.6 (Inverse). *An invertible function $h : \mathbb{R} \to \mathbb{R}$ is non-nullifying if its inverse h^{-1} is an* absolutely continuous function.

Proof. We have discussed absolute continuity of measures; the absolute continuity of functions is a related idea. It is a stronger notion than continuity and uniform continuity. Absolutely continuous functions are well behaved in many ways; in particular, the images of null sets are also null sets. Coupled with the fact that an h-preimage is an h^{-1}-image, this proves the claim. More details on absolutely continuous functions can be found in [17]. ☐

This result shows that log, exp, and non-constant linear functions are non-nullifying. We believe the idea can be extended without much difficulty to show that functions with a countable number of invertible pieces, such as the trigonometric functions and non-constant polynomials, are also non-nullifying.

Lemma 4.7 (Piecewise)**.** *For functions $c : A \to \mathbb{B}$ and $f, g, h : A \to B$, where $h(x) = \text{if } c(x) \text{ then } f(x) \text{ else } g(x)$, h is non-nullifying if f and g are non-nullifying.*

Proof. Let Y be a $\overline{\mu}_B$-null set. The set $h^{-1}[Y]$ is a subset of $f^{-1}[Y] \cup g^{-1}[Y]$ and is thus $\overline{\mu}_A$-null, by non-nullity of f and g, and the countable additivity and completeness of $\overline{\mu}_A$. □

Lemma 4.8 (Composition)**.** *The set of non-nullifying functions is closed under function composition.*

Proof. Let $f : A \to B$ and $g : B \to C$ be non-nullifying functions and let $h = g \circ f$. The h-preimage of a $\overline{\mu}_C$-null set Z is given by $h^{-1}[Z] = f^{-1}[g^{-1}[Z]]$, and is thus $\overline{\mu}_A$-null, by the non-nullity of f and g. □

Lemma 4.9 (Projection)**.** *The function $h(x, y) = x$ of type $A \times B \to A$ is non-nullifying.*

Proof. Let X be a $\overline{\mu}_A$-null set. Its h-preimage is $X \times B$. By the properties of product measure, we have that

$$\overline{\mu}_{A\times B}(X \times B) = \overline{\mu}_A(X) \cdot \overline{\mu}_B(B) = 0 \cdot \overline{\mu}_B(B) = 0.$$

Even if $\overline{\mu}_B(B) = \infty$, the measure-theoretic definition of multiplication on extended nonnegative reals defines $0 \cdot \infty = 0$. □

Along these lines, we can show that returning a permutation of a subset of tuple components is also a non-nullifying function. The last two results permit us to ignore uninvolved arguments when reasoning about the non-nullity of the body of a function.

4.2 Distributions and RV transforms

A large class of distributions in our language can be understood by Equation 3. From the syntax we know that a distribution e must take the form of zero or more nested binds terminating in a body that is either random or $\mathsf{return}\ \varepsilon$. We focus on the latter, non-trivial case. The expression ε represents a transformation of the random variables x_i introduced by the binds. The function $\lambda(x_1, ..., x_n).\ \varepsilon$ is the *random variables transform (RV transform)* of the distribution e, where we use tuple pattern matching shorthand to name the components of a tuple argument. The correspondence between distributions in our language and Theorem 4.2 is as follows: let $\mathbb{Q}$ be the denotation of e, let h be the RV transform of e, and let $\mathbb{P}$ be the joint distribution of the random variables introduced on the spine of e. The class of distributions for which the theorem is applicable is given by the set of distributions for which each e_i is *parametrically* AC w.r.t. the random variables preceding it, where e_i is the distribution corresponding to x_i. In other words, the distribution for e_i must be AC while treating free occurrences of $x_1, ..., x_{i-1}$ as fixed, unknown constants. This ensures that the joint distribution is also AC; the joint PDF can be written as the product of the individual parameterized PDFs. This is a commonly used (implicit) assumption in practice. For example, the distribution

$$\mathsf{var}\ u \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ z \sim \mathsf{flip}\ u\ \mathsf{in}\ \mathsf{return}\ (u + \langle z \rangle)$$

has the RV transform $\lambda(u, z).\ u + \langle z \rangle$, which has type $\mathbb{R} \times \mathbb{B} \to \mathbb{R}$ and is transforming the joint distribution of random and $e_2 := \mathsf{flip}\ u$. The variable u appears free in e_2, making e_2 parametric in u; the restriction requires that e_2 is AC for all possible values of u, which is the case here. Two extensionally equivalent distributions may have different RV transforms and spines because of intensionally different representations. To show that this choice of $\mathbb{P}$, $\mathbb{Q}$, and h satisfies Equation 3, we appeal to the semantics of distributions (defined in Section 3.4). Consider the general case

$$\frac{}{\Upsilon; \Lambda \vdash \mathsf{random}\ \text{AC}}\ \text{AC-RAND} \qquad \frac{\Upsilon; \Lambda \vdash \varepsilon\ \text{NN}}{\Upsilon; \Lambda \vdash \mathsf{return}\ \varepsilon\ \text{AC}}\ \text{AC-RET}$$

$$\frac{\Upsilon; \emptyset \vdash e_1\ \text{AC} \qquad \Upsilon \vdash e_1 : \mathsf{dist}\ \tau \qquad \Upsilon, x : \tau \sim e_1; \Lambda, x \vdash e_2\ \text{AC}}{\Upsilon; \Lambda \vdash \mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2\ \text{AC}}\ \text{AC-BIND}$$

Figure 4. The absolute continuity judgment, $\Upsilon; \Lambda \vdash e$ AC.

$e := \overline{\mathsf{var}\ x_i\ \sim\ e_i\ \mathsf{in}}\ \mathsf{return}\ \varepsilon$, where we have used the bar as shorthand for nested binds. The denotation $\mathbb{Q}$ of e under an environment ρ is given by

$$\mathbb{Q}(Y) = \overline{\int d\mathbb{P}_i\ \lambda x'_i.}\ \mathbf{1}_Y(\mathcal{E}[\![\varepsilon]\!]\rho\overline{\{x_i \mapsto x'_i\}})$$

where $\mathbb{P}_i$ is the denotation of e_i (extending ρ as necessary) and we have again used the bar notation, to denote iterated expectation and the repeated extension of the environment ρ with variable mappings. We can now rewrite the expectations to use their corresponding PDFs f_i and then replace the iterated integrals with a single product integral using their joint PDF f:

$$\begin{aligned}
\mathbb{Q}(Y) &= \overline{\int d\overline{\mu}_{\tau_i} \lambda x'_i .f_i(x'_i; x'_1, ..., x'_{i-1})} \cdot \mathbf{1}_Y(\mathcal{E}[\![\varepsilon]\!]\rho\overline{\{x_i \mapsto x'_i\}}) \\
&= \int d\overline{\mu}_\tau\ \lambda\mathbf{x}.\ f(\mathbf{x}) \cdot \mathbf{1}_Y(h(\mathbf{x})) \\
&= \int d\mathbb{P}\ \lambda\mathbf{x}.\ \mathbf{1}_Y(h(\mathbf{x})) \\
&= \mathbb{P}(\{\mathbf{x} \mid h(\mathbf{x}) \in Y\})\ =\ \mathbb{P}(h^{-1}[Y])
\end{aligned}$$

where $\mathbf{x} = (x'_1, ..., x'_n)$, $h(\mathbf{x}) = \mathcal{E}[\![\varepsilon]\!]\rho\overline{\{x_i \mapsto \mathbf{x}_i\}}$, τ_i is the type of each x_i, and τ is their product. We have also used the fact that the expectation of the indicator function on a set is the probability of that set (the set here is $\{\mathbf{x} \mid h(\mathbf{x}) \in Y\}$, not Y). Replacing an iterated integral with a product integral is not always legal but is possible here because the integral is of a nonnegative function w.r.t. independent measures (see Tonelli's theorem, [17]).

4.3 Type system for programs

All judgments are defined modulo α-conversion. A program $\mathsf{pdf}\ e$ is well-formed if e is an AC distribution ($\emptyset \vdash e : \mathsf{dist}\ \tau$ holds for some τ and $\emptyset; \emptyset \vdash e$ AC holds). If the judgment $\Upsilon; \Lambda \vdash e$ AC (Figure 4) holds then e is an AC distribution under the probability context Υ and the active variable context Λ, where Λ is given by the grammar $\Lambda ::= \emptyset \mid \Lambda, x$. Variables in Λ are currently active and should be understood in a probabilistic sense, while those not in Λ are inactive and should be treated as fixed parameters. The contexts obey the following invariant: Λ is always the "prefix" of Υ, *i.e.* the variables in Λ correspond directly to the n most recent entries added to Υ, where n is the length of Λ. Rule AC-RAND asserts that the Uniform(0,1) distribution is AC. The main action of rules AC-BIND and AC-RETURN is to prepare a call to the non-nullity judgment. For Theorem 4.2 to be applicable, a distribution along the spine must be parametrically AC w.r.t. the random variables preceding it; thus, in AC-BIND we check that e_1 is AC without marking any current random variables as active. We reach the body of the RV transform in AC-RETURN. Roughly speaking, Λ (pointing into Υ) and ε correspond to $\mathbb{P}$ and h in Theorem 4.2.

Next is the non-nullity judgment (Figure 5). If $\Upsilon; \Lambda \vdash \varepsilon$ NN holds, then ε represents the body of a non-nullifying function under Υ and Λ. The variables in Λ are the arguments to the RV transform. Throughout this discussion, we implicitly use the composition and projection lemmas (Lemmas 4.8 and 4.9) to ignore uninvolved arguments during analysis. For example, in rule NN-VAR, we could

$$\frac{x \in \Lambda}{\Upsilon; \Lambda \vdash x \text{ NN}} \text{NN-VAR} \qquad \frac{\Upsilon \vdash \varepsilon : \tau \quad \tau \text{ countable}}{\Upsilon; \Lambda \vdash \varepsilon \text{ NN}} \text{NN-COUNT}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \text{ NN} \quad op \in \{\mathsf{neg}, \mathsf{inv}, \mathsf{log}, \mathsf{exp}, \mathsf{sin}, \mathsf{cos}, \mathsf{tan}\}}{\Upsilon; \Lambda \vdash op\ \varepsilon \text{ NN}} \text{NN-OP}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon_1 \text{ NN} \quad \Upsilon; \Lambda \vdash \varepsilon_2 \text{ NN}}{\Upsilon; \Lambda \vdash \mathsf{if}\ \varepsilon\ \mathsf{then}\ \varepsilon_1\ \mathsf{else}\ \varepsilon_2 \text{ NN}} \text{NN-IF}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \text{ NN} \quad op \in \{\mathsf{fst}, \mathsf{snd}\}}{\Upsilon; \Lambda \vdash op\ \varepsilon \text{ NN}} \text{NN-PROJ}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_2 \quad \Upsilon; \Lambda \vdash \varepsilon_1 \text{ NN} \quad \Upsilon; \Lambda \vdash \varepsilon_2 \text{ NN}}{\Upsilon; \Lambda \vdash (\varepsilon_1, \varepsilon_2) \text{ NN}} \text{NN-PAIR}$$

$$\frac{x_i \in \Lambda \quad x_1, ..., x_n \text{ are distinct}}{\Upsilon; \Lambda \vdash (x_1, ..., x_n) \text{ NN}} \text{NN-VARS}$$

$$\frac{\Upsilon; \Lambda \vdash (\varepsilon_1, \varepsilon_2) \text{ NN}}{\Upsilon; \Lambda \vdash \varepsilon_1 + \varepsilon_2 \text{ NN}} \text{NN-PLUS}$$

$$\frac{FV(\varepsilon_2) \cap \Lambda = \emptyset \quad \Upsilon; \Lambda \vdash \varepsilon_1 \text{ NN}}{\Upsilon; \Lambda \vdash \varepsilon_1 + \varepsilon_2 \text{ NN}} \text{NN-LINEAR}$$

$$\frac{\Upsilon; \Lambda \vdash (\varepsilon_1, \varepsilon_2) \text{ NN}}{\Upsilon; \Lambda \vdash \varepsilon_1 * \varepsilon_2 \text{ NN}} \text{NN-MULT} \qquad \frac{l \neq 0 \quad \Upsilon; \Lambda \vdash \varepsilon \text{ NN}}{\Upsilon; \Lambda \vdash l * \varepsilon \text{ NN}} \text{NN-SCALE}$$

Figure 5. The non-nullity judgment, $\Upsilon; \Lambda \vdash \varepsilon$ NN.

be analyzing a function with multiple inputs, but we can drop all of them but x, leaving us to analyze the function $\lambda x \,.\, x$, which is trivially non-nullifying. Under the hood, what we are actually doing is representing the original transform as the composition of a function that selects a single components of a tuple with the identity function $\lambda x \,.\, x$. The composition lemma is also the justification for being able to recurse into subexpressions. Rule NN-COUNT is merely an application of Lemma 4.4; the types bool, Z and products thereof define the countable types. Note that this covers the cases of $=$, $<$, integer neg, $+$ and $*$, and Boolean and integer literals. Rules NN-OP, NN-IF and NN-PROJ are direct translations of Lemmas 4.6, 4.7 and 4.9. The injection from integers into the reals is nullifying (Lemma 4.3), so there is no rule for R_of_Z. Rule NN-PAIR expresses the idea that the joint distribution of independent AC distributions is AC. If $\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_2$ holds then ε_1 and ε_2 represent independent distributions under Υ and Λ. Its definition is

$$\frac{\Lambda \cap \text{Anc}(\Upsilon, FV(\varepsilon_1)) \cap \text{Anc}(\Upsilon, FV(\varepsilon_2)) = \emptyset}{\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_2} \text{INDEP}$$

where $\text{Anc}(\Upsilon, X) = \bigcup_{x \in X} \text{anc}(\Upsilon, x)$. It states that ε_1 and ε_2 must not have any ancestors in common. The function $\text{anc}(\Upsilon, x)$ computes the ancestors of a random variable x. A random variable y is the parent of a random variable x if y appears free in the distribution that x is bound to. Rule NN-VARS corresponds to the corollary of Lemma 4.9 that states that you can drop and permute tuple components. The requirement that the variables are distinct is important; the distribution var $u \sim$ random in return (u, u) is not AC, as we saw in Section 2. We have multiple rules for addition because they each capture a different usage of plus. Rule NN-PLUS states that if the formation of the pair $(\varepsilon_1, \varepsilon_2)$ is non-nullifying, then $\varepsilon_1 + \varepsilon_2$ is also non-nullifying because it is the composition of tuple formation with $(+) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, where the latter is non-nullifying by corollary to Lemma 4.6. Rule NN-LINEAR represents the idea of composing with the non-nullifying function $\lambda x \,.\, x + c$, where c is a constant w.r.t. the arguments of the RV transform. There is an analogous rule for when the constant appears as the left operand. Rules NN-MULT and NN-SCALE are analogous. Note that NN-SCALE is slightly weaker than its counterpart NN-LINEAR, only because it needs to prove that the scaling coefficient is nonzero.

Discussion We believe our type system is sound; the only remaining case to rigorously prove is NN-PAIR. The soundness of the reduction to non-nullity is given by Theorem 4.2, and the soundness of the other cases in the non-nullity judgment are covered by the lemmas in Section 4.1. Stating the needed lemma for NN-PAIR essentially requires formalizing the idea that the conditional distribution of the second component conditioned on the first component should be AC. In non-nullity terms, the second component should still have a degree of freedom even after fixing the first. Rigorously stating this involves conditional probability, putting it outside the scope of the current work.

There are few sources of incompleteness in our type system. For instance, NN-PAIR conservatively requires ε_1 and ε_2 to be independent. The distribution

$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (\mathsf{exp}\ x, x + y)$$

is AC despite the fact that the tuple components are not independent: even if we know the value of exp x, the "residual" stochasticity in the quantity $x + y$ is still AC. The joint PDF is given by multiplying the marginal PDF of the first component by the conditional PDF of the second component conditioned on the first. This is a similar issue as the parametric AC requirement on spine distributions. Formulating this generalization of NN-PAIR is interesting future work. Likewise, NN-IF conservatively requires both branches of an if-expression to be non-nullifying. The distribution

$$\mathsf{var}\ x \sim \mathsf{std_normal}\ \mathsf{in}\ \mathsf{return}(\mathsf{if}\ x < 0\ \mathsf{then}\ \mathsf{min}\ x\ 0\ \mathsf{else}\ \mathsf{max}\ x\ 0)$$

is not accepted as AC because both branches ($\lambda x \,.\, \mathsf{min}\ x\ 0$ and $\lambda x \,.\, \mathsf{max}\ x\ 0$) are nullifying, even though the distribution is extensionally equivalent to var $x \sim$ std_normal in return x, which is AC. We define min and max in the usual way, using if. Finally, non-nullity is sufficient but not necessary for absolute continuity to hold. For instance, the RV transform of

$$\mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{return}\ (\mathsf{if}\ x < 100\ \mathsf{then}\ x\ \mathsf{else}\ 100)$$

is $\lambda x \,.\, \mathsf{if}\ x < 100\ \mathsf{then}\ x\ \mathsf{else}\ 100$, which is nullifying due to the constant portion, thus our type system does not accept this distribution as AC. However, x only takes values on $(0, 1)$, so the second branch is never entered, and thus the distribution is extensionally equivalent to the AC distribution var $x \sim$ random in return x.

4.4 Semantics of programs

The denotation of a program pdf e is that it is a member of the set of Radon-Nikodym derivatives of the distribution e:

$$[\![\mathsf{pdf}\ e]\!] \in \{f \mid \forall X,\ \mathbb{P}(X) = \int_X f\ d\overline{\mu}_\tau\}$$

where $\mathbb{P} = \mathcal{E}[\![e]\!]\{\}$ is the denotation of e under the empty environment and e has type dist τ. The procedure discussed in the next section calculates a member of this set.

5. Calculating density functions

The previous sections have defined a language in which it is possible to express PDFs. Our goal now is to mechanically obtain a usable form of the PDF for a given distribution. But what constitutes a usable form? We are motivated by applications of the PDF and the need to interface with existing software. For instance, we may want to use numerical optimization software to perform MLE, where the PDF appears in the objective function; we may also want to symbolically derive gradient information to improve the search. Or, we may want to use the PDF to calculate an expectation using a numerical integrator. Roughly speaking, we call a term "usable" if we can map it onto the capabilities of existing software in accordance with common practice. For example, the term $\lambda x \,.\, x + 5$ is

$$\begin{array}{rl}
\text{Target types} & \sigma ::= \tau \mid \sigma_1 \to \sigma_2 \\
\text{Target terms} & \delta ::= \varepsilon \mid \lambda x : \tau \,\textbf{.}\, \delta \mid \delta_1\ \delta_2 \mid \int \delta \\
\text{Typing} & \dfrac{\Gamma \vdash \delta : \tau \to \mathsf{R}}{\Gamma \vdash \int \delta : \mathsf{R}}\ \text{T-INT} \\
\text{Semantics} & \mathcal{E}[\![\int \delta]\!]\rho = \int \mathcal{E}[\![\delta]\!]\rho \ \ d\overline{\mu}_\tau
\end{array}$$

Figure 6. The target language.

usable; in practice, real addition is mapped to floating point addition. Likewise, $\int_0^5 x^2\ dx$ is usable; the integral is Riemannian and in a form accepted by computer algebra systems (CAS) and numerical integrators. On the other hand, terms like $\int g\ d\mathbb{P}$ and $d\mathbb{P}/d\mathfrak{L}$ make use of measure-theoretic operations such as abstract integration and the Radon-Nikodym derivative. Current software do not handle these operations (though, progress on mechanizing measure theory has been made [15]). Thus, the basic plan is to eliminate measure-theoretic concepts during PDF calculation. This means the constructs random, return, bind, and pdf should not appear in a PDF term because they involve measure theory, metatheoretically.

It will take some ingenuity to remove the Radon-Nikodym derivative (pdf). It has been shown that the Radon-Nikodym derivative is a non-computable operator: given a distribution, there is no general computable procedure for computing its PDF [11]. The discrete case at least enjoys the fact that the PMF has a straightforward definition in terms of its distribution; if $\mathbb{P}$ is an executable implementation of a discrete distribution, an executable implementation of its PMF $d\mathbb{P}/d\mathfrak{C}$ is given by $\lambda x \,\textbf{.}\, \mathbb{P}(\{x\})$. In general, however, we will need to tackle the calculation of PDFs with a collection of techniques. Our basic approach is as follows. First, we define a target language that defines what constitutes a usable form. Second, we provide a procedure that converts many distributions accepted as AC by our type system into PDFs expressed in the target language. Some RV transforms are mathematically inconvenient, so we will not be able to calculate certain PDFs from scratch; in particular, dependence between random variables makes the general case difficult. However, the design permits modularly adding knowledge about individual distributions with known PDFs, enabling the procedure to calculate PDFs for programs that use these distributions as subcomponents. This allows us to handle many useful cases.

5.1 The target language

The target language extends expressions with λ-abstraction, application, and the stock integral (Figure 6). We treat functions in a standard way. Notationally, we skip specifying τ in abstractions when the choice of τ is clear. Computing closed-form solutions for integrals is not always feasible or possible, so integrals cannot be completely eliminated from the target language. The integral is well-formed if its integrand is real-valued and summable (a function f is *μ-summable* if $\int f\ d\mu$ is finite). We require users of the target language (compiler writers) to manually ensure summability; this is reasonable for a back-end language. We have verified summability for each use of stock integration in the compilers presented in this section. Although a measure-theoretic concept, stock integration is close enough to the notion of integration used by numerical and symbolic solvers to be useful as a compilation target. Recall, stock integration over $\mathfrak{C}$ and $\mathfrak{L}$ is ordinary summation and Lebesgue integration, respectively. For most applications, Lebesgue integration will coincide with Riemann integration.

$$\begin{array}{l}
\mathsf{random}\ \$\ \delta \mapsto \int \lambda x : \mathsf{R} \,\textbf{.}\, \langle 0 < x < 1 \rangle * \delta\ x \\
\mathsf{return}\ \varepsilon\ \$\ \delta \mapsto \delta\ \varepsilon \\
\dfrac{e_2\ \$\ \delta \mapsto \delta' \qquad e_1\ \$\ \lambda x \,\textbf{.}\, \delta' \mapsto \delta''}{\mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2\ \$\ \delta \mapsto \delta''}
\end{array}$$

Figure 7. The probability compiler, $e\ \$\ \delta \mapsto \delta'$.

$$\begin{array}{l}
\dfrac{}{\Upsilon;\Lambda \vdash \mathsf{random} \curvearrowright \lambda x : \mathsf{R} \,\textbf{.}\, \langle 0 < x < 1 \rangle}\ \text{P-RAND} \\
\dfrac{\Upsilon \vdash e_1 : \mathsf{dist}\ \tau \qquad \Upsilon, x : \tau \sim e_1; \Lambda, x \vdash e_2 \curvearrowright \delta}{\Upsilon;\Lambda \vdash \mathsf{var}\ x \sim e_1\ \mathsf{in}\ e_2 \curvearrowright \delta}\ \text{P-BIND} \\
\dfrac{\Upsilon;\Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon;\Lambda \vdash \mathsf{return}\ \varepsilon \curvearrowright \delta}\ \text{P-RET}
\end{array}$$

Figure 8. The distribution-to-PDF converter, $\Upsilon;\Lambda \vdash e \curvearrowright \delta$.

5.2 The probability compiler

We need to calculate probabilities as a subroutine of PDF calculation. We achieve this by translating distributions into Kozen-style terms [14]. The probability compiler $e\ \$\ \delta \mapsto \delta'$ performs this translation (Figure 7). It takes a distribution e of type $\mathsf{dist}\ \tau$ and a function δ from τ to $[0, 1]$ and returns the expectation of δ w.r.t. e. When δ is the indicator function on a set X, δ' is the e-probability of X. For instance, suppose we want to know the probability that a sample from flip $(3/4)$ is true. We invoke the probability compiler with $e := \mathsf{flip}\ (3/4)$ and $\delta := \lambda z : \mathsf{bool} \,\textbf{.}\, \langle z \rangle$, producing

$$\int \lambda x : \mathsf{R} \,\textbf{.}\, \langle 0 < x < 1 \rangle * (\lambda u \,\textbf{.}\, (\lambda z \,\textbf{.}\, \langle z \rangle)\ (u < 3/4))\ x$$

for δ', which is equivalent to $\int_0^1 \langle x < 3/4 \rangle\ dx = 3/4$, as expected. Likewise, to derive the probability that a standard normal random variable stays within a standard deviation of its mean, we would invoke the probability compiler with $e := \mathsf{std_normal}$ and $\delta := \lambda x \,\textbf{.}\, \langle -1 < x < 1 \rangle$. Details on how this computes probabilities are given by Kozen and can also be understood by the expectation monad [20]. We also need the judgment $\Upsilon \vdash \varepsilon\ \$\ \delta \mapsto \delta'$, which invokes the probability compiler on the distribution corresponding to the RV transform body ε in the context Υ.

5.3 The PDF calculation procedure

We structure the PDF calculation procedure as we did the type system: the judgment on distributions prepares a call to the judgment on RV transforms. The PDF of a well-formed program pdf e is given by the δ satisfying $\emptyset; \emptyset \vdash e \curvearrowright \delta$. The judgment $\Upsilon;\Lambda \vdash e \curvearrowright \delta$ calculates the PDF δ of the distribution e under Υ and Λ (Figure 8). Rule P-RAND gives the PDF of Uniform(0,1): the indicator function on $(0, 1)$. Rules P-RET and P-BIND build the contexts and invoke the next compiler. The real work begins in the judgment $\Upsilon;\Lambda \vdash \varepsilon \rightsquigarrow \delta$, which computes the PDF δ corresponding to the RV transform body ε under Υ and Λ. We present this judgment in two parts, one each for univariate and multivariate transforms. The multivariate transforms must deal with the issue of dependence between inputs or between outputs of the transform.

Univariate transforms We use *univariate* for RV transforms between spaces that are not product spaces. The correctness of rules P-LOG, P-EXP, P-LINEAR, and P-SCALE is given by the following lemma.

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \mathsf{log}\, \varepsilon \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (\mathsf{exp}\, x) * \mathsf{exp}\, x} \text{ P-LOG}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \mathsf{exp}\, \varepsilon \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (\mathsf{log}\, x) * (1/x)} \text{ P-EXP}$$

$$\frac{FV(\varepsilon_2) \cap \Lambda = \emptyset \qquad \Upsilon; \Lambda \vdash \varepsilon_1 \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \varepsilon_1 + \varepsilon_2 \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (x - \varepsilon_2)} \text{ P-LINEAR}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta \qquad l > 0}{\Upsilon; \Lambda \vdash l * \varepsilon \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (x/l) * (1/l)} \text{ P-SCALE}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \mathsf{neg}\, \varepsilon \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (-x)} \text{ P-NEG}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \mathsf{inv}\, \varepsilon \rightsquigarrow \lambda x : \mathsf{R} \,.\, \delta\, (1/x) * (1/(x * x))} \text{ P-INV}$$

Figure 9. The transform-to-PDF converter, $\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta$, univariate cases.

Lemma 5.1. *For absolutely continuous distributions $\mathbb{P}$ and $\mathbb{Q}$ on $\mathbb{R}$ and a function $h : \mathbb{R} \to \mathbb{R}$ such that $\mathbb{Q}(Y) = \mathbb{P}(h^{-1}[Y])$, if h is strictly increasing, differentiable and invertible, then the function*

$$g(y) = f(h^{-1}(y)) \cdot \frac{d}{dy} h^{-1}(y).$$

is a PDF of $\mathbb{Q}$, where f is the derivative of the CDF F of $\mathbb{P}$.

Proof. The derivative of a CDF is a PDF. The CDF G of $\mathbb{Q}$ is

$$\begin{aligned} G(y) &= \mathbb{Q}(\,(-\infty, y]\,) \;=\; \mathbb{P}(h^{-1}[\,(-\infty, y]\,]) \\ &= \mathbb{P}(\,(-\infty, h^{-1}(y)]\,) \;=\; F(h^{-1}(y)), \end{aligned}$$

where we have used the fact that the h-preimage of $(-\infty, y]$ is $(-\infty, h^{-1}(y)]$ because h is strictly increasing and invertible. The claim follows from the fact that g is the derivative of G. □

The lemma is easily modified for P-NEG and also P-INV; an "extra" minus sign appears because they consist of strictly *decreasing* components. It is possible to define a version of P-SCALE for negative literals, as well as integer versions of P-NEG, P-LINEAR, and P-SCALE. With these rules (and P-VAR, discussed below) we can already compute some continuous PDFs. Consider the standard exponential from Section 3.2; we derive its PDF with $\emptyset; \emptyset \vdash \mathsf{std_exponential} \curvearrowright \delta$, which builds the contexts $\Lambda := u$ and $\Upsilon := u : \mathsf{R} \sim \mathsf{random}$ and invokes the chain

$$\begin{aligned} &\Upsilon; \Lambda \vdash -\mathsf{log}\, u \rightsquigarrow \delta && \delta'' := \lambda x'' \,.\, \langle 0 < x'' < 1 \rangle \\ &\Upsilon; \Lambda \vdash \mathsf{log}\, u \rightsquigarrow \delta' && \delta' := \lambda x' \,.\, \langle 0 < \mathsf{exp}\, x' < 1 \rangle * \mathsf{exp}\, x' \\ &\Upsilon; \Lambda \vdash u \rightsquigarrow \delta'' && \delta := \lambda x \,.\, \langle 0 < \mathsf{exp}\, (-x) < 1 \rangle * \mathsf{exp}\, (-x). \end{aligned}$$

We β-reduce for clarity. The chain ends with P-VAR, which gives the PDF of Uniform(0,1) for δ''; then, P-LOG and P-NEG produce δ' and δ. The latter is equivalent to $\lambda x \,.\, \langle 0 < x \rangle * \mathsf{exp}\, (-x)$, which is easily seen to be the PDF of the standard exponential. Likewise, the PDF of $\mathsf{uniform}\; \varepsilon_1\; \varepsilon_2$ is correctly calculated to be

$$\delta := \lambda x \,.\, \langle 0 < (x - \varepsilon_1)/(\varepsilon_2 - \varepsilon_1) < 1 \rangle * (1/(\varepsilon_2 - \varepsilon_1)),$$

which is equivalent to $\lambda x \,.\, \langle \varepsilon_1 < x < \varepsilon_2 \rangle * (1/(\varepsilon_2 - \varepsilon_1))$. We do not provide rules for sin, cos, and tan because we are unaware of any simple closed-form expression for the corresponding PDFs.

Multivariate transforms We use *multivariate* for RV transforms to or from a product space. The presence of multiple dimensions introduces the issue of dependence between the inputs or between the outputs of the transform, making it difficult to provide rules that work in the general case. As a result, some of the following rules introduce specific independence requirements.

$$\frac{\emptyset \vdash l : \tau \qquad \tau \text{ countable}}{\Upsilon; \Lambda \vdash l \rightsquigarrow \lambda x : \tau \,.\, \langle x = l \rangle} \text{ P-LIT}$$

$$\frac{\Upsilon \vdash \varepsilon : \mathsf{bool} \qquad \Upsilon; \Lambda \vdash \varepsilon \,\$\, \lambda x : \mathsf{bool} \,.\, \langle x \rangle \mapsto \delta}{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \lambda x : \mathsf{bool} \,.\, \mathsf{if}\; x \;\mathsf{then}\; \delta \;\mathsf{else}\; 1 - \delta} \text{ P-BOOL}$$

$$\frac{\{\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_i\}_{i=2,3} \qquad \{\Upsilon; \Lambda \vdash \varepsilon_i \rightsquigarrow \delta_i\}_{i=1,2,3}}{\begin{array}{c}\Upsilon; \Lambda \vdash \mathsf{if}\; \varepsilon_1 \;\mathsf{then}\; \varepsilon_2 \;\mathsf{else}\; \varepsilon_3 \rightsquigarrow \\ \lambda x \,.\, \delta_1\; \mathsf{true} * \delta_2\; x + \delta_1\; \mathsf{false} * \delta_3\; x\end{array}} \text{ P-IF}$$

$$\frac{\Lambda = \{x\} \sqcup \{y_1, ..., y_m\} \qquad \mathcal{J}(\Upsilon; \Lambda) \mapsto \delta}{\Upsilon; \Lambda \vdash x \rightsquigarrow \lambda x \,.\, \int \lambda(y_1, ..., y_m) \,.\, \delta} \text{ P-VAR}$$

$$\frac{\Lambda = \{x_1, ..., x_n\} \sqcup \{y_1, ..., y_m\} \qquad \mathcal{J}(\Upsilon; \Lambda) \mapsto \delta}{\Upsilon; \Lambda \vdash (x_1, ..., x_n) \rightsquigarrow \lambda(x_1, ..., x_n) \,.\, \int \lambda(y_1, ..., y_m) \,.\, \delta} \text{ P-VARS}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon \rightsquigarrow \delta}{\Upsilon; \Lambda \vdash \mathsf{fst}\, \varepsilon \rightsquigarrow \lambda x \,.\, \int \lambda y \,.\, \delta\, (x, y)} \text{ P-FST}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_2 \qquad \{\Upsilon; \Lambda \vdash \varepsilon_i \rightsquigarrow \delta_i\}_{i=1,2}}{\Upsilon; \Lambda \vdash (\varepsilon_1, \varepsilon_2) \rightsquigarrow \lambda(x_1, x_2) \,.\, \delta_1\, x_1 * \delta_2\, x_2} \text{ P-PAIR}$$

$$\frac{\Upsilon; \Lambda \vdash \varepsilon_1 \perp \varepsilon_2 \qquad \{\Upsilon; \Lambda \vdash \varepsilon_i \rightsquigarrow \delta_i\}_{i=1,2}}{\Upsilon; \Lambda \vdash \varepsilon_1 + \varepsilon_2 \rightsquigarrow \lambda x : \mathsf{R} \,.\, \int \lambda t : \mathsf{R} \,.\, \delta_1\, x * \delta_2\, (t - x)} \text{ P-PLUS}$$

Figure 10. The transform-to-PDF converter, multivariate cases.

$$\frac{}{\mathcal{J}(\Upsilon; \emptyset) \mapsto 1} \text{ J-NIL} \qquad \frac{\Upsilon; \emptyset \vdash e \curvearrowright \delta \qquad \mathcal{J}(\Upsilon; \Lambda) \mapsto \delta'}{\mathcal{J}(\Upsilon, x : \tau \sim e; \Lambda, x) \mapsto \delta\, x * \delta'} \text{ J-CONS}$$

Figure 11. The joint PDF body constructor, $\mathcal{J}(\Upsilon; \Lambda) \mapsto \delta$.

Rule P-LIT states that the PMF of a point mass distribution on l is simply the indicator function on $\{l\}$. The transforms corresponding to the rules in this section tend to be less obvious; the transform in question for P-LIT is the constant function on l, whose argument may be a tuple. Rule P-BOOL calculates the PMF of a Boolean random variable, which is a simple expression of the probability that the random variable is true. We thus invoke the probability compiler in the current context to compute this probability δ. This rule covers the cases for $<$ and $=$. The ability to represent the PMF of a Boolean random variable allows us to encode arbitrary probability queries. Rule P-IF computes the PDF of a mixture, which is a weighted combination of the component PDFs, where the mixing probability is the probability the if-condition is true. For this to be valid, the if-condition must be independent of its branches, as required. For instance, the PDF of

$\mathsf{var}\; x \sim \mathsf{random}\; \mathsf{in}$
$\quad \mathsf{var}\; y \sim \mathsf{uniform}\; 2\; 3\; \mathsf{in}\; \mathsf{return}\; (\mathsf{if}\; x < 1/2\; \mathsf{then}\; x\; \mathsf{else}\; y).$

is *not* equivalent to $\lambda x \,.\, (1/2) * \langle 0 < x < 1 \rangle + (1/2) * \langle 2 < x < 3 \rangle$, as would be calculated without the restriction (there should be no probability mass on $[1/2, 1]$).

Rule P-VAR is a special case of P-VARS. The transform corresponding to P-VARS is a function that returns a permutation of a subset of components of its tuple argument. We assume $x_1, ..., x_n$ and $y_1, ..., y_m$ are distinct, and we use $\sqcup$ to denote disjoint union. The resulting PDF is a marginal PDF. The *marginal* PDF of a joint PDF f on $A \times B$ is given by $g(x) = \int \lambda y \,.\, f(x, y)\; d\overline{\mu}_B$; g is a PDF on A whose density at x is given by adding up the contribution of the joint PDF along the other dimension, B. The corresponding process is one which generates tuples but then discards the second component, returning the first. We generalize to higher dimensions by integrating out random variables not appearing in the result tuple. When this set is empty ($m = 0$), the integral re-

duces to δ. The resulting PDF may be computationally inefficient due to a large number of nested integrals. More efficient schemes that take advantage of the graphical structure of the probabilistic model, such as *variable elimination*, are possible [26]. The judgment $\mathcal{J}(\Upsilon; \Lambda) \mapsto \delta$ constructs the body of the joint PDF of the active random variables (Figure 11). Rule J-CONS first computes the PDF of e, parametric in all of the preceding random variables (thus, invoking the distribution-to-PDF converter with no active random variables). It then constructs the product with the PDFs of the remaining active variables; the product of these parametric PDFs is the joint PDF. The terms δ and δ' in J-CONS have type $\tau \to \mathsf{R}$ and R, respectively. The judgment returns an open term and relies on the fact that the free variables will be bound appropriately by the invoking judgment. Rule P-FST is analogous to P-VARS; we ask for a PDF and compute the marginal PDF of the first component. We define an analogous rule for snd. Rules P-PAIR and P-PLUS state the well known results that the joint PDF and the PDF of the sum of independent random variables is the product of and convolution of their individual PDFs, respectively.

On the face of it, these rules handle mixture models and joint models, but where they really shine is on general hierarchical models. For example, the PDF of

$$\mathsf{hier} := \mathsf{var}\ x \sim \mathsf{random}\ \mathsf{in}\ \mathsf{var}\ y \sim \mathsf{uniform}\ 0\ x\ \mathsf{in}\ \mathsf{return}\ y$$

is not immediately obvious. The process is generated by sampling a value x uniformly from (0,1), and then sampling uniformly from $(0, x)$, discarding x. We calculate the PDF with $\emptyset; \emptyset \vdash \mathsf{hier} \curvearrowright \delta$, which builds $\Upsilon := y : \mathsf{R} \sim \mathsf{uniform}\ 0\ x, x : \mathsf{R} \sim \mathsf{random}$ and $\Lambda := y, x$ for $\Upsilon; \Lambda \vdash y \rightsquigarrow \delta$. Rule P-VAR then produces

$$\lambda y \,.\, \int \lambda x \,.\, (\langle 0 < (y-0)/(x-0) < 1\rangle * 1/(x-0)) * \langle 0 < x < 1\rangle * 1$$

for δ, where we have β-reduced for clarity. The body of the inner λ-abstraction is generated by the joint PDF body constructor; the two non-trivial multiplicands are the parametric PDF of uniform $0\ x$ and the PDF of random, respectively. With some manipulation we can show δ corresponds to $f(y) = \int_y^1 1/x\ dx = -\log(y)$ for $y \in (0, 1)$ and zero otherwise. The rules do not perform algebraic simplifications, but the benefit of automation can still be felt clearly.

Modularity Some RV transforms are inconvenient to work with, preventing us from calculating certain PDFs. For example, we cannot calculate the PDF of std_normal from scratch because its specification uses cos, which we do not handle. However, the design allows us to modularly address cases like this, where we want to specially handle the PDF for a specific distribution. We can add the rule $\Upsilon; \Lambda \vdash \mathsf{std_normal} \curvearrowright \phi$, where $\phi := \lambda x.\mathsf{exp}(-x*x/2)/\mathsf{sqrt}(2*\pi)$ is the PDF of the standard normal. This new rule is used by the joint body constructor whenever std_normal appears on the spine of a distribution, enabling the calculation of PDFs for hierarchical models using std_normal that were previously not compilable. For example, the PDF of normal $\mu\ \sigma$ can now be calculated as

$$\begin{array}{ll} \Upsilon; \Lambda \vdash \sigma * x + \mu \rightsquigarrow \delta & \delta'' := \lambda x'' \,.\, \phi\ x'' \\ \Upsilon; \Lambda \vdash \sigma * x \rightsquigarrow \delta' & \delta' := \lambda x' \,.\, \phi\ (x'/\sigma) * (1/\sigma) \\ \Upsilon; \Lambda \vdash x \rightsquigarrow \delta'' & \delta := \lambda x \,.\, \phi\ ((x-\mu)/\sigma) * (1/\sigma) \end{array}$$

using the rules P-VAR, P-SCALE, and P-LINEAR, where $\Lambda := x$ and $\Upsilon := x : \mathsf{R} \sim \mathsf{std_normal}$. We can see δ is equivalent to the classic formula for the normal PDF, $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2\sigma^2}(x-\mu)^2)$. Likewise, we can now handle distributions like the log-normal and mixture-of-Gaussians. To support an infinite discrete distribution with a known PDF, such as the Poisson distribution, we can add a new primitive to the core calculus (poisson ε) and handle it specially in the distribution-to-PDF converter.

6. Related Work

Our work builds on a long tradition of probabilistic functional languages, most connected to the probability monad in some way. They work by incorporating distributional semantics into a functional language, so that one can express values which represent a *distribution* over possible outcomes. The distribution can either be manifest (available to the programmer) or implicit (existing only in the metatheory). An early incarnation of the latter was given by Kozen in [14], in which he provides the semantics for an imperative language endowed with a random number primitive supplying samples from Uniform(0,1). Values of type A in the object language are given semantics in functions of type $(A \to [0, 1]) \to [0, 1]$ in the metatheory. These functions represent distributions over A and satisfy the expected laws for measures. Kozen's work is far-reaching and will continue to inspire future languages: it can accommodate continuous and hybrid distributions; it handles unbounded iteration (general recursion), a traditionally thorny issue for probabilistic languages; and it even provides a treatment of distributions on function types. However, PDFs are not addressed at all.

Though not explicitly cast as functional or monadic, Kozen's approach forms the basis for Audebaud and Paulin-Mohring's monadic development for reasoning about randomized algorithms in COQ [2]. Their focus is on verification, and they define the probability monad from first principles (modulo an axiomatization of arithmetic on the [0,1] interval), whereas we provide it axiomatically. We hope to inspire a cross-fertilization of ideas between the efforts as we bring our theory of PDFs into COQ.

While suitable for semantics and verification, Kozen's representation is not ideal for direct use in computing certain operations. For instance, it is unclear how to sample or compute general expectations efficiently given a term of type $(A \to [0, 1]) \to [0, 1]$. More recent works explore alternate concrete embodiments of the probability monad; Ramsey and Pfeffer discuss some of the possibilities [20]. A popular choice is to represent distributions as weighted lists or trees. This has the drawback that only distributions with finitely many outcomes are expressible (ruling out essentially all commonly used continuous distributions), and PMFs are the only supported form of PDFs. On the other hand, distributions can occur on arbitrary types, expectation and computing the PMF is straightforward, and the approach works well as an embedded domain-specific language (PFP [7], HANSEI [13], probability monads in Haskell [20]). Dedicated languages like IBAL [19] or Church [9] offer more scope for program analysis, which is crucial for escaping the limitations of an embedded approach and mitigates some of the fundamental drawbacks of the representation. Ultimately, however, these languages do not support continuous or hybrid distributions (nor their PDFs) in a general sense. Sampling functions are a fun alternative representation. They are used by $\lambda_{\bigcirc}$ [18] to support continuous and hybrid distributions in a true sense and also allow distributions on arbitrary types. Distributions are represented by sampling functions that return a sample from the distribution when requested. Sampling and sampling-based routines are the only supported operations, thus PDFs are not accommodated.

Another recent work also rigorously supports continuous and hybrid distributions by providing a measure transformer semantics for a core functional calculus [4]. The work does not provide PDFs but is novel for its ability to support conditional probability in the presence of zero probability events in continuous spaces, a feature necessary in many machine learning applications. Their formalization is similar to ours, as both are based in standard measure theory. They have independently recognized the importance of analyzing distributions by their transformations, doing so in the context of conditional probability, whereas we have developed the idea for PDFs. This hints that reasoning via transforms may be a technique

that is more broadly applicable to other program analyses for probabilistic languages.

The Hierarchical Bayes Compiler (HBC) is a toolkit for implementing hierarchical Bayesian models [5]. Its specification language represents a different point in the design space. Essentially, it removes `return` while adding a set of standard distributions (with PDFs) to the core calculus. This guarantees that all constructible models are AC. Many powerful models used in machine learning are expressible in HBC. However, something as basic as adding two random variables is not. Furthermore, if a distribution outside of the provided set is required, it must be added to the core. This is the fundamental tension surrounding `return`: with it, the core is minimal, expressivity is high, and PDFs are non-trivial; without it, PDFs are easily supported, but the core becomes large, and expressivity is crippled. HBC is not formally defined.

An entirely different tradition incorporates probabilistic semantics into logic programming languages (Markov Logic [21], BLOG [16], BLP [12], PRISM [22]). These languages are well suited for probabilistic knowledge engineering and statistical relational learning. In Markov Logic, for instance, programmers associate higher weights with logical clauses that are more strongly believed to hold. The semantics of a set of clauses is given by *undirected graphical models*, with the weights determining the potential functions (*e.g.* by Boltzmann weighting). Certain continuous distributions can be supported by manipulating the potential function calculation. Supporting PDFs in this context should not be problematic; the potential functions (essentially, unnormalized PDFs) always exist, by design. However, like HBC, it appears these languages are not quite as expressive as is possible in a probabilistic functional language.

The AutoBayes system [10] shares a key feature with our language in that PDFs are manifest in the object language. AutoBayes automates the derivation of maximum likelihood and Bayesian estimators for a significant class of statistical models, with a focus on code generation, and can express continuous distributions and PDFs. However, despite their focus on correctness-by-construction, the language is not formally defined. Furthermore, it is unclear how general the language actually is, *i.e.* how "custom" the models can be. Our work could serve as a formal basis for their system.

7. Conclusion

We have presented a formal language capable of expressing discrete, continuous and hybrid distributions and their PDFs. Our novel contributions include a type system for absolutely continuous distributions and a modular PDF calculation procedure. The type system uses the new ideas of RV transforms and non-nullifying functions. There are several interesting avenues for future work. The first is to address PDFs in the context of conditional probability, perhaps by incorporating our formalization of PDFs with the ideas presented in [4]. Secondly, to provide a complete account of continuous probability, one must support expectation. Generically supporting expectation requires a treatment of integrability or summability; reasoning via the RV transform may be a productive route. Finally, combining this work with a formal language for optimization such as [1] would create a true formal language for *statistics*, which would be able to express statistical problems in the object language itself. Current languages express *probability*; any notion of statistics is outside the language.

Acknowledgments

We thank Prof. Christopher Heil for valuable input on the idea of non-nullifying functions. We also thank the anonymous reviewers, whose thoughtful suggestions have greatly improved the paper.

References

[1] A. Agarwal, S. Bhat, A. Gray, and I. E. Grossmann. Automating Mathematical Program Transformations. In *Practical Aspects of Declarative Languages*, 2010.

[2] P. Audebaud and C. Paulin-Mohring. Proofs of Randomized Algorithms in Coq. In *Mathematics of Program Construction*, pages 49–68. Springer, 2006.

[3] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] J. Borgstram, A. D. Gordon, M. Greenberg, J. Margetson, and J. V. Gael. Measure Transformer Semantics for Bayesian Machine Learning. In *European Symposium on Programming*, pages 77–96, 2011.

[5] H. Daumé III. HBC: Hierarchical Bayes Compiler, 2007. URL `http://hal3.name/HBC`.

[6] L. Devroye. Non-Uniform Random Variate Generation, 1986.

[7] M. Erwig and S. Kollmansberger. Functional Pearls: Probabilistic Functional Programming in Haskell. *Journal of Functional Programming*, 16(01):21–34, 2005.

[8] M. Giry. A Categorical Approach to Probability Theory. *Categorical Aspects of Topology and Analysis*, 915:68–85, 1981.

[9] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: A Language for Generative Models. In *Uncertainty in Artificial Intelligence*, 2008.

[10] A. G. Gray, B. Fischer, J. Schumann, and W. Buntine. Automatic Derivation of Statistical Algorithms: The EM Family and Beyond. In *Advances in Neural Information Processing Systems*, 2003.

[11] M. Hoyrup, C. Rojas, and K. Weihrauch. The Radon-Nikodym operator is not computable. In *Computability & Complexity in Analysis*, 2011.

[12] K. Kersting and L. De Raedt. Bayesian Logic Programming: Theory and Tool. In *Introduction to Statistical Relational Learning*. 2007.

[13] O. Kiselyov and C. Shan. Embedded Probabilistic Programming. In *Working Conference on Domain Specific Languages*. Springer, 2009.

[14] D. Kozen. Semantics of Probabilistic Programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[15] T. Mhamdi, O. Hasan, and S. Tahar. On the Formalization of the Lebesgue Integration Theory in HOL. *Interactive Theorem Proving*, pages 387–402, 2010.

[16] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. BLOG: Probabilistic Models with Unknown Objects. In *International Joint Conference on Artificial Intelligence*, volume 19, 2005.

[17] O. Nielsen. *An Introduction to Integration and Measure Theory*. Wiley-Interscience, 1997.

[18] S. Park, F. Pfenning, and S. Thrun. A Probabilistic Language based upon Sampling Functions. In *Principles of Programming Languages*, pages 171–182. ACM New York, NY, USA, 2005.

[19] A. Pfeffer. IBAL: A Probabilistic Rational Programming Language. In *International Joint Conference on Artificial Intelligence*, 2001.

[20] N. Ramsey and A. Pfeffer. Stochastic Lambda Calculus and Monads of Probability Distributions. volume 37, pages 154–165. ACM, 2002.

[21] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1):107–136, 2006.

[22] T. Sato and Y. Kameya. PRISM: A Symbolic-Statistical Modeling Language. In *International Joint Conference on Artificial Intelligence*, pages 1330–1339, 1997.

[23] D. Scott. Parametric Statistical Modeling by Minimum Integrated Square Error. *Technometrics*, 43(3):274–285, 2001.

[24] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[25] R. Solovay. A Model of Set-Theory in Which Every Set of Reals is Lebesgue Measurable. *Annals of Mathematics*, pages 1–56, 1970.

[26] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2004.