On Probabilistic Termination of Functional Programs with Continuous Distributions

Raven Beutner
University of Oxford, and
Saarland University
UK and Germany

Luke Ong
University of Oxford
UK

Abstract

We study termination of higher-order probabilistic functional programs with recursion, stochastic conditioning and sampling from continuous distributions.

Reasoning about the termination probability of programs with continuous distributions is hard, because the enumeration of terminating executions cannot provide any non-trivial bounds. We present a new operational semantics based on *traces of intervals*, which is sound and complete with respect to the standard sampling-based semantics, in which (countable) enumeration can provide arbitrarily tight lower bounds. Consequently we obtain the first proof that deciding almost-sure termination (AST) for programs with continuous distributions is Π^0_2-complete (for CbN). We also provide a compositional representation of our semantics in terms of an intersection type system.

In the second part, we present a method of proving AST for *non-affine programs*, i.e., recursive programs that can, during the evaluation of the recursive body, make *multiple* recursive calls (of a first-order function) from *distinct* call sites. Unlike in a deterministic language, the number of recursion call sites has direct consequences on the termination probability. Our framework supports a proof system that can verify AST for programs that are well beyond the scope of existing methods.

We have constructed prototype implementations of our methods for computing lower bounds on the termination probability, and AST verification.

CCS Concepts: • **Theory of computation → Operational semantics**; **Program analysis**; **Program verification**.

Keywords: almost-sure termination, probabilistic programs, sampling-style operational semantics, intersection types, random walk

PLDI '21, June 20–25, 2021, Virtual, Canada

ACM ISBN 978-1-4503-8391-2/21/06…$15.00
https://doi.org/10.1145/3453483.3454111

ACM Reference Format:
Raven Beutner and Luke Ong. 2021. On Probabilistic Termination of Functional Programs with Continuous Distributions. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21), June 20–25, 2021, Virtual, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3453483.3454111

1 Introduction

Probabilistic (or randomised) programs have long been recognised as essential to the efficient solution of many algorithmic problems [46, 48, 52]. Recently, in probabilistic programming [26, 53, 58], probabilistic programs, augmented with stochastic conditioning constructs, have been used as a means of expressing generative models whose posterior probability can be computed by general-purpose inference engines. Though sampling from discrete distributions (such as binary probabilistic branching) can be considered algorithmically adequate[1] for probabilistic computation, the generation of real-world data—a basic capability expected of generative models—requires expressivity of the whole gamut of continuous distributions. For this reason, sampling from continuous distributions is an essential feature of probabilistic programming languages. (See e.g. Church [25], Stan [12], Anglican [56], Gen [18], Pyro [5], Edward [57] and Turing [24].)

In this work we study a central property of probabilistic programs: *termination*. In non-probabilistic (possibly nondeterministic) computation, termination is a purely qualitative, boolean property. However, with randomness in the control flow, termination is characterised by a scalar quantity: the *probability of termination*. We say that a program is *almost-surely terminating* (AST) if a run of it terminates with probability 1.

Guarantees and bounds on the probability of termination are important both when viewing probabilistic programs as algorithmic solutions but also in the emerging field of probabilistic programming. When a probabilistic program implements a solution to an algorithmic problem, one naturally requires the computation to terminate with a high (lower bounded) probability, usually 1. In probabilistic programming, lower bounds and guarantees of AST are equally

[1] in the sense that they are enough to make any Turing complete programming language universal for probabilistic Turing machine [37, 55]

important. Indeed, it is standard for designers and implementors of probabilistic programming systems to regard non-AST programs as defining invalid models, and hence inadmissible (see e.g. [53, §4.3.2] and [25]). Moreover [41] have recently shown that AST programs have density (a.k.a. weight) functions that are differentiable almost everywhere. This is significant, because the latter property is a precondition for the correctness of some of the most scalable inference algorithms, such as Hamiltonian Monte Carlo [49, 59] and reparameterised gradient variational inference [39]. AST is thus a precondition for the correctness of inference algorithms and important both in theory and practice.

In this paper we tackle two key questions: computation of lower bounds on the probability of termination, and AST verification. While there has been much progress in the termination analysis of probabilistic programs with discrete distributions [8, 29, 33], programs with continuous distributions have received comparatively little attention. Many methods and proofs hinge on the countable nature inherent to discrete distributions [9, 30, 33, 36, 43, 44, 50]. It is not at all obvious if they can be extended to systems with continuous distributions.

Using an idealised functional language with continuous samples and stochastic conditioning, we provide partial answers to these questions. On the one hand, we give a definitive answer to the lower bound problem, and precisely determine the complexity of various termination problems in the arithmetic hierarchy. On the other hand, we provide a sound (but incomplete) proof method for AST which can be seen as orthogonal to [36].

1.1 High Level Overview

Lower Bound Computation. In languages with discrete distributions, evaluation can be seen as a step-indexed probability mass on terms [21, 33, 36]. By enumerating terminating executions, we can iteratively compute arbitrarily tight lower bounds on the probability of termination. As a direct consequence, AST is a decision problem in Π^0_2 [29], the second level of the arithmetic hierarchy [32][2]. In languages that admit continuous distributions, we cannot assign probability mass to terms directly. Rather, by viewing a probabilistic program as a deterministic program parameterised by an *execution trace* (or simply, trace) (i.e. the sequence of random draws made during the execution), we can organise such traces into a measure space [6, 34]. The probability of termination can then be defined as the measure of all traces on which the program terminates [41]. However, in general, a single terminating execution (or even a countable set thereof) cannot be assigned any positive probability measure. This leaves open problems such as sound computation of lower bounds, and the exact complexity of deciding AST.

We approach these problems by introducing a novel operational semantics based on *interval traces*, which are a summarisation of the relevant traces. We show soundness and completeness w.r.t. the sampling-style semantics [6]. Instead of analysing a program using uncountably many traces, we work with interval traces, where only countably many such traces suffice. This yields an effective procedure to compute lower bounds on termination probability, enabling the first proof that deciding AST in the presence of continuous distributions is Π^0_2-complete (under mild assumptions on the primitive functions). Further, we show that positive almost sure termination (PAST) (i.e., finite expected time to termination) is Σ^0_2-complete, assuming the program is AST. For general PAST, we can only infer a (possibly non-tight) upper bound of Δ^0_3. This does *not* match the Σ^0_2 bounds known for discrete distributions as a proof of this bound hinges on a countable set of executions [29]. See Sec. 3.

In addition we give an alternative presentation of our semantics as an intersection type system in Sec. 4. Our system extends [9] and [21] to languages with continuous distributions; moreover, both the probability of termination *and* the expected time to termination can be obtained as the least upper bound of all derivations. This gives a type-based, compositional method for lower bound computation.

AST Verification. While our computation of lower bounds gives a Π^0_2 decision procedure for AST, it is not really effective for AST verification. Many of the recent advances in the development of AST verification methods [1, 13, 14, 16, 17, 23, 27, 28, 44, 50] are concerned with loop-based programs. We can view such loops as tail-recursive programs that, in particular, are *affine recursive*, i.e., in each evaluation (or run) of the body of the recursion, recursive calls are made from at most one call site [36, §4.1]. By contrast, many probabilistic programming languages allow for richer recursive structures [25, 42, 56]. We propose a new verification method for probabilistic programs that are defined by *non-affine recursion*, i.e., in the evaluation of the body of the recursion, multiple recursive calls can be made from *distinct* call sites. (Note that whether a program is affine recursive cannot be checked by just counting textual occurrences of variables.)

Example 1.1 (Running Example)**.** Consider an unreliable 3d printing company. Unfortunately, for every printing, the outcome is acceptable with only probability p; if it is unacceptable, reprinting must take place on the following day, and thus, the process is repeated. We can model this scenario, starting with a single job, as the following program

$$\left(\mu^{\varphi}_{x}.\texttt{if sample} \leq p \texttt{ then } x \texttt{ else } \varphi(x+\underline{1})\right)\underline{1} \qquad (1)$$

[2]The class Π^0_n in the arithmetic hierarchy contains a language $\mathcal{L}$ iff there exists a decidable relation $R(x, y_1, \cdots, y_n)$ such that $x \in \mathcal{L} \Leftrightarrow \forall y_1.\exists y_2.\forall y_3 \cdots .R(x, y_1, \cdots, y_n)$. Σ^0_n is defined analogously starting with an existential instead of universal quantifier. Σ^0_1 is thus the class of recursive enumerable languages. Almost-sure termination means that for *all* (rational) termination probability δ strictly smaller than 1, there *exists* some finite set of terminating execution T whose weight is at least δ, making it a problem contained in Π^0_2.

where $\mu^{\varphi}_{x}.(\cdot)$ is a fixpoint constructor (that binds the variable φ to the fixpoint), and `sample` evaluates to a random draw from the uniform distribution on $[0,1]$. The value returned by the program is the number of days needed to complete the job. Luckily, as the program is AST for all success probabilities $p \in (0,1]$, the company can assure its customers that it will finish the job eventually. However, in a bid to drum up business, a new quality policy is introduced. The manager advises their customers: "Each day our print attempt fails, we will print an additional copy for you." We model the situation as follows:

$$\left(\mu^{\varphi}_{x}.\texttt{if sample} \le p \texttt{ then } x \texttt{ else } \varphi(\varphi(x+\underline{1}))\right)\underline{1} \tag{2}$$

Soon after implementing the new policy, it was noticed that some of the print jobs could never be completed. Phrased differently: Program (2) is no longer AST for every $p \in (0,1]$.

This example illustrates that non-affine recursion, as exhibited in program (2), can complicate the analysis of termination. While the affine program (1) is clearly AST for every $p > 0$, program (2) is not. It turns out that (2) is AST if and only if $p \ge \frac{1}{2}$; and in case $p = \frac{1}{2}$, while the process is AST, the expected time to termination is infinite. It is unsurprising that termination depends on the number of recursive calls, as termination itself is a quantitative property.

Termination analysis of non-affine recursive probabilistic programs does not seem to have received much attention. Methods such as those presented in [36] explicitly restrict to affine programs and are unsound otherwise. Our method for the analysis of non-affine recursive programs can be viewed as orthogonal to [36]: while they restrict to affine programs and investigate the recursive function argument for size information, we accept the function argument without examination, and admit non-affine programs. We call our methods *counting-based*, as we over-approximate the recursive behaviour by counting recursive calls from distinct call sites, thus reducing AST analysis to the analysis of a random walk for which we show linear decidability. See Sec. 5. Our method is the basis of an AST proof system that can verify programs (including the simple example above) well beyond the reach of existing methods (Sec. 6). As a simple corollary, we obtain a functional *generalisation* of the *zero-one law* for termination of while-programs [43, §2.6][3]

Contributions. Our main contributions are as follows:

- We propose a new sound and complete interval-based semantics that enables lower bound computation. We obtain a first proof that the (CbN) AST (resp. PAST) decision problem is, under mild assumptions on primitive functions, Π^0_2-complete (resp. Σ^0_2-complete) even in the presence of continuous distributions.
- We give a local representation of our semantics as an intersection type system where both the probability of termination and expected time to termination are characterised as the least upper bound over all derivations.
- We provide a new proof method for AST verification of non-affine recursive programs. We show how our proof system can be automated.

Our theoretical results give rise to practical algorithms. We provide prototype implementations for both lower bound computation and AST verification based on our novel semantics and proof system respectively (Sec. 7). Missing proofs and further discussions can be found in [4].

2 Statistical PCF (SPCF)

We begin by introducing some basics of probability theory and presenting our language of study.

2.1 Basic Probability Theory

A *σ-algebra* on a set Ω, typically written Σ_Ω, is a collection of subsets of Ω such that $\Omega \in \Sigma_\Omega$, and Σ_Ω is closed under complementation and countable unions (and hence countable intersections). A *measurable space* is a pair (Ω, Σ_Ω) where Ω is a set (of outcomes) and Σ_Ω is a σ-algebra on Ω. A function $f : \Omega_1 \to \Omega_2$ between measurable spaces, $(\Omega_1, \Sigma_{\Omega_1})$ and $(\Omega_2, \Sigma_{\Omega_2})$, is called *measurable* if for every $A \in \Sigma_{\Omega_2}$, $f^{-1}(A) \in \Sigma_{\Omega_1}$. A *measure* on (Ω, Σ_Ω) is a function $\mu : \Sigma_\Omega \to \overline{\mathbb{R}_+}$ that satisfies $\mu(\emptyset) = 0$ and is σ-additive: if $\{A_i\}_{i\in\mathbb{N}}$ is a countable family of pairwise disjoint sets from Σ_Ω then $\mu(\cup_i A_i) = \sum_i \mu(A_i)$. If $\mu(\Omega) \le 1$ we call μ a subprobability measure and if $\mu(\Omega) = 1$ we call it a probability measure (or distribution). For the n-dimensional Euclidean space $\mathbb{R}^n$ we write $\Sigma_{\mathbb{R}^n}$ for the Borel σ-algebra over $\mathbb{R}^n$, which is the smallest σ-algebra that contains all open and closed n-dimensional boxes. In the special case of $n = 1$, this is the set generated by all open (and closed) intervals. The n-dimensional Lebesgue measure, denoted λ_n, is the unique measure on $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$ that satisfies $\lambda_n([a_1,b_1] \times \cdots \times [a_n,b_n]) = \prod_{i=1}^n (b_i - a_i)$.

Discrete Sample Space. In case Ω is countable, we often work with the powerset 2^Ω as the trivial σ-algebra. Every probability measure is then uniquely determined by a *probability mass function (pmf)*, a function $p : \Omega \to \mathbb{R}_{[0,1]}$ with $\sum_{x\in\Omega} p(x) = 1$. Every pmf p gives rise to a probability measure by defining $\mu(A) := \sum_{x\in A} p(x)$; conversely, for every probability measure μ on the powerset we can recover a generating pmf by defining $p(x) := \mu(\{x\})$. A subprobability mass function is defined analogously.

2.2 SPCF

Statistical PCF (SPCF) is an extension of PCF [51] with support to sample[4] from the uniform distribution on $[0,1]$ and condition executions (see [26]). Terms in SPCF are implicitly

[3] The *zero-one law* states that a while-loop is almost-surely terminating if there is a positive lower bound on the probability of exiting it.

[4] Sampling from other real-valued distributions can be obtained from `sample` by applying the inverse of the distribution's cumulative distribution function; see e.g. [54, §2.3.1].

$$\frac{}{\Gamma \vdash \texttt{sample} : \mathbf{R}} \qquad \frac{\Gamma \vdash M : \mathbf{R}}{\Gamma \vdash \texttt{score}(M) : \mathbf{R}}$$

$$\frac{\Gamma, \varphi : \alpha \to \beta, x : \alpha \vdash M : \beta}{\Gamma \vdash \mu_x^{\varphi}.M : \alpha \to \beta} \qquad \frac{\{\Gamma \vdash M_i : \mathbf{R}\}_{i=1}^{|f|}}{\Gamma \vdash f(M_1, \cdots, M_{|f|}) : \mathbf{R}}$$

Figure 1. Selection of SPCF typing rules

parametrised over a set $\mathbb{F}$ of *measurable* functions $f : \mathbb{R}^n \to \mathbb{R}$ that model primitive operations. Each function $f \in \mathbb{F}$ has an arity $|f| \geq 0$. The sets of terms and values are defined by the following grammar where x and φ are distinct variables (from a fixed denumerable set of symbols), $r \in \mathbb{R}$ and $f \in \mathbb{F}$:

$$V ::= x \mid \underline{r} \mid \lambda x.M \mid \mu_x^{\varphi}.M$$
$$M, N, P ::= V \mid MN \mid \texttt{if}(M, N, P) \mid f(M_1, \cdots, M_{|f|}) \mid \texttt{sample} \mid \texttt{score}(M)$$

As usual, we identify terms modulo α-conversion. The fixpoint constructor, $\mu_x^{\varphi}.(\cdot)$, binds the recursively defined function φ and its argument x. We abbreviate[5]

$$M \oplus_P N ::= \texttt{if}\ (\texttt{sample} - P, M, N)$$

in the style of [43] and write $M \oplus N$ for $M \oplus_{\underline{.5}} N$. We type terms using a standard simple type system with types defined by $\alpha, \beta ::= \mathbf{R} \mid \alpha \to \beta$. A selection of typing rules is given in Fig. 1 (see [4] for a full system) . We denote the set of typable SPCF terms by Λ and its subset of closed terms by Λ_0.

In this paper we consider both call-by-name (CbN) and call-by-value (CbV) evaluation strategies. We use CbN for the first part of this paper, as the results (especially those about intersection types) are cleaner this way [9, 21, 33]. (Our CbN SPCF can express CbV computation at base types, giving it a suitable algorithmic expressiveness; c.f. [20].) We switch to CbV SPCF when presenting our AST proof system, thereby enabling a more straightforward comparison to related approaches such as [36].

2.3 Operational Semantics

We give a sampling-style operational semantics for SPCF. The idea (going back to Kozen [34]) is to evaluate a term M together with a sequence of (fixed) probabilistic outcomes for each `sample` statement [6, 41]. We then generate a probabilistic interpretation of programs by endowing the set of traces with a measure.

CbN SPCF. We define the set of traces $\mathbb{S}$ as all finite sequences of real numbers from $\mathbb{R}_{[0,1]} := \{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$), i.e., $\mathbb{S} := \mathbb{R}^*_{[0,1]} = \bigcup_{n \in \mathbb{N}} \mathbb{R}^n_{[0,1]}$. We let $\boldsymbol{s}$ range over elements in $\mathbb{S}$, denote the empty trace with ϵ; for $r \in \mathbb{R}_{[0,1]}$ write r for the one element trace; and $\boldsymbol{s}_1, \boldsymbol{s}_2$ for concatenation. The set of CbN redexes and evaluation contexts is defined by:

$$R ::= (\lambda x.M)N \mid (\mu_x^{\varphi}.M)N \mid \texttt{if}(\underline{r}, N, P) \mid f(\underline{r_1}, \cdots, \underline{r_{|f|}}) \mid \texttt{sample} \mid \texttt{score}(\underline{r})$$
$$E ::= [\cdot] \mid EM \mid \texttt{if}(E, N, P) \mid \texttt{score}(E) \mid f(\underline{r_1}, \cdots, \underline{r_{k-1}}, E, M_{k+1}, \cdots, M_{|f|})$$

Given a context E and a term M the (capture-permitting) substitution $E[M]$ is defined in the obvious way. An easy induction establishes that every $M \in \Lambda_0$ is either a value or there are *unique* E and R, s.t., $M = E[R]$ (see e.g. [6]). The small-step reduction relation has the from $\langle M, \boldsymbol{s}\rangle \to \langle M', \boldsymbol{s}'\rangle$ where M, M' are terms and $\boldsymbol{s}, \boldsymbol{s}'$ are traces. It is defined inductively by the rules given in Fig. 2 where $M[N_i/x_i]_i$ denotes standard capture-avoiding substitution [3]. Note that our reduction does not enjoy *progress*, as e.g. redex $\texttt{score}(\underline{r})$ cannot reduce if $r < 0$.

The score constructs is used to stochastic condition of executions (see e.g. [26]) by weighting each execution [6]. As this work is a study of termination properties, we elide the weight parameter used for stochastic conditioning as the weight of a execution is irrelevant for the termination behaviour[6].

A Measure on Traces. To interpret probabilistic programs using traces, we first need to endow the set of traces with a measure. We cannot assign probability mass to individual traces directly, as there are uncountably many traces. Instead we define a suitable measurable space of program traces following [6]. Let $\Sigma_{\mathbb{R}^n_{[0,1]}}$ be the Borel σ-algebra on $\mathbb{R}^n_{[0,1]}$ (We set $\Sigma_{\mathbb{R}^0_{[0,1]}} := \{\emptyset, \{\epsilon\}\}$). We can then define a σ-algebra on traces ($\Sigma_{\mathbb{S}}$) and a measure ($\mu_{\mathbb{S}}$) by:

$$\Sigma_{\mathbb{S}} := \{\biguplus_{n \in \mathbb{N}} B_n \mid B_n \in \Sigma_{\mathbb{R}^n_{[0,1]}}\}$$
$$\mu_{\mathbb{S}}\big(\biguplus_{n \in \mathbb{N}} B_n\big) := \sum_{n \in \mathbb{N}} \lambda_n(B_n)$$

As shown in [6, Lem. 7 & 8], $(\mathbb{S}, \Sigma_{\mathbb{S}})$ is a measurable space and $\mu_{\mathbb{S}}$ a (σ-finite) measure on $(\mathbb{S}, \Sigma_{\mathbb{S}})$.

2.4 Probabilistic Termination

With $\to^n$ we denote the n-fold self-composition, and with $\to^*$ the reflexive-transitive closure, of $\to$. We define

$$\mathbb{T}_{M,\text{term}} := \{\boldsymbol{s} \in \mathbb{S} \mid \exists V : \langle M, \boldsymbol{s}\rangle \to^* \langle V, \epsilon\rangle\}$$

as the set of traces on which a term M terminates, which is measurable (similar to [6, Lem. 9]). As shown in [41, Lem. 7], $\mu_{\mathbb{S}}(\mathbb{T}_{M,\text{term}}) \leq 1$; we are therefore justified in calling the interpretation $\mu_{\mathbb{S}}(\mathbb{T}_{M,\text{term}})$ a "probability".

[5] Our conditional statement, $\texttt{if}(P, M, N)$, branches on whether $P \leq 0$.

[6] The weight function can be seen as a function mapping terminating traces to weights (i.e., $\mathbb{R}$). The denotation of a program is then the Lebesgue integral of this weight functions over the set of terminating traces ([6, §3.4]). To get e.g., the almost-everywhere differentiability of the weight function (needed for correct inference), it is sufficient to show that the measure of the set of termination traces is 1 (irrespective of the weight on these traces) [41, §4.3]. The `score`-construct has, nevertheless, a subtle effect on termination as we require the conditioned value to be positive.

$$\frac{}{\langle (\lambda x.M)N, s\rangle \to \langle M[N/x], s\rangle} \qquad \frac{}{\langle (\mu^{\varphi}_{x}.M)N, s\rangle \to \langle M[N/x, (\mu^{\varphi}_{x}.M)/\varphi], s\rangle} \qquad \frac{}{\langle \mathtt{sample}, r\, s\rangle \to \langle \underline{r}, s\rangle}$$

$$\frac{r \le 0}{\langle \mathtt{if}(\underline{r}, N, P), s\rangle \to \langle N, s\rangle} \qquad \frac{r > 0}{\langle \mathtt{if}(\underline{r}, N, P), s\rangle \to \langle P, s\rangle} \qquad \frac{r \ge 0}{\langle \mathtt{score}(\underline{r}), s\rangle \to \langle \underline{r}, s\rangle}$$

$$\frac{}{\langle f(\underline{r_1}, \cdots, \underline{r_{|f|}}), s\rangle \to \langle \underline{f(r_1, \cdots, r_{|f|})}, s\rangle} \qquad \frac{\langle R, s\rangle \to \langle M, s'\rangle}{\langle E[R], s\rangle \to \langle E[M], s'\rangle}$$

Figure 2. Call-by-name small-step reduction for SPCF

Definition 2.1. The *probability of termination* of $M \in \Lambda_0$ is defined by $\mathbb{P}_{\mathrm{term}}(M) := \mu_{\mathbb{S}}(\mathbb{T}_{M,\mathrm{term}})$. M is called *almost-surely terminating* (AST) if $\mathbb{P}_{\mathrm{term}}(M) = 1$.

Positive Almost-Sure Termination. An even stronger property than AST is finiteness of the expected time to termination. For any trace $s \in \mathbb{T}_{M,\mathrm{term}}$ we define $\#^{s}_{\downarrow}(M) \in \mathbb{N}$ as the unique number n such that $\langle M, s\rangle \to^{n} \langle V, \epsilon\rangle$ for some value V. For any $n \in \mathbb{N}$ we define

$$\mathbb{T}^{\le n}_{M,\mathrm{term}} := \left\{ s \in \mathbb{T}_{M,\mathrm{term}} \mid \#^{s}_{\downarrow}(M) \le n \right\}$$

as the set of traces on which termination occurs within n steps, which is measurable. We define $\mathbb{T}^{n}_{M,\mathrm{term}}$ analogously.

Definition 2.2. For $M \in \Lambda_0$ we define the *expected time to termination*, $\mathbb{E}_{\mathrm{term}}(M) \in \overline{\mathbb{R}_+}$, by

$$\mathbb{E}_{\mathrm{term}}(M) := \sum_{n=0}^{\infty} \left(1 - \mu_{\mathbb{S}}\left(\mathbb{T}^{\le n}_{M,\mathrm{term}}\right)\right)$$

M is *positive almost-surely terminating* if $\mathbb{E}_{\mathrm{term}}(M) < \infty$.

It is easy to see that any program that is PAST is also AST. Following [29], $\mathbb{E}_{\mathrm{term}}(M)$ can be phrased as $\sum_{n=0}^{\infty} \mathbb{P}(\text{“}M \text{ runs for more than } n \text{ steps”}) = \sum_{n=0}^{\infty} \left(1 - \mathbb{P}(\text{“}M \text{ terminates within } n \text{ steps”})\right)$, with the latter expressed in Def. 2.2. We can show that, provided M is AST, the expected time to termination is the expected value of the random variable that gives the number of reduction steps:

Lemma 2.3. *If M is AST, $\mathbb{E}_{term}(M) = \sum_{n=0}^{\infty} \mu_{\mathbb{S}}\left(\mathbb{T}^{n}_{M,term}\right) \cdot n$*

CbV SPCF. Our CbV SPCF is essentially the system of [41], except that we use a simpler CbV fixpoint reduction rule.

3 Interval-Based Semantics

It is impractical to use the standard trace-based (or sampling-style) semantics to reason about termination properties of SPCF programs, because the trace measure $\mu_{\mathbb{S}}$ is continuous. Suppose we are interested in the decidability of the *lower bound question*: does a term terminate with probability *strictly* greater than p? For discrete distributions, this problem is r.e. (in Σ^0_1) as we can enumerate terminating paths until the sum of the weight of those paths exceeds p [29, 33]. In the presence of continuous distributions, this is no longer possible. A well-known property of the Lebesgue measure on $\mathbb{R}^n_{[0,1]}$ (inherited by the trace measure $\mu_{\mathbb{S}}$) is that every countable set of elements is a null set. So even if we can identify a countably infinite set of traces $A \subseteq \mathbb{T}_{M,\mathrm{term}}$, we cannot obtain any non-trivial lower bound on $\mathbb{P}_{\mathrm{term}}(M)$. Thus the semantics itself cannot be used to settle such complexity questions as whether the lower bound problem for SPCF is in Σ^0_1, or whether the AST problem is in Π^0_2, or whether the PAST problem in Σ^0_2. In this section, we introduce a novel operational semantics for SPCF by executing terms parameterised by a trace of *intervals*. We demonstrate that this semantics, which is complete w.r.t. the trace-based semantics, is well-suited to the derivation of lower bounds. The completeness hinges on the observation that, under mild restrictions on primitive functions, interval-based reasoning can effectively abstract actual traces. This is the basis of our positive answer to the questions above.

Syntax of Interval Terms. We adjust the syntax of terms slightly and treat intervals as constant symbols of type $\mathbf{R}$. We define interval values and interval terms as follows where $a \le b \in \mathbb{R}$.

$$\mathcal{V} ::= x \mid \underline{[a,b]} \mid \lambda x.\mathcal{M} \mid \mu^{\varphi}_{x}.\mathcal{M}$$
$$\mathcal{M}, \mathcal{N}, \mathcal{P} ::= \mathcal{V} \mid \mathcal{M}\mathcal{N} \mid \mathtt{if}(\mathcal{M}, \mathcal{N}, \mathcal{P}) \mid f(\mathcal{M}_1, \cdots, \mathcal{M}_{|f|}) \mid \mathtt{sample} \mid \mathtt{score}(\mathcal{M})$$

Our simple type system extends naturally. We denote the set of (closed, bounded) intervals by $\mathfrak{I}$, and write $\mathfrak{I}_{0,1} := \{[a,b] \mid a, b \in \mathbb{R}, 0 \le a \le b \le 1\}$ as the set of intervals with endpoints between 0 and 1. With $\mathfrak{I}^{\mathbb{Q}}$ and $\mathfrak{I}^{\mathbb{Q}}_{0,1}$ we denote the sets $\mathfrak{I}$ and $\mathfrak{I}_{0,1}$ respectively, restricted to *rational endpoints*.

Definition 3.1. We call $f : \mathbb{R}^n \to \mathbb{R}$ *interval preserving* (resp. $\mathbb{Q}$-*interval preserving*) if there is a function $\hat{f} : \mathbb{R}^{2n} \to \mathfrak{I}$ (resp. $\hat{f} : \mathbb{Q}^{2n} \to \mathfrak{I}^{\mathbb{Q}}$) such that for every sequence of intervals $[a_1, b_1], \cdots, [a_n, b_n] \in \mathfrak{I}$ (resp. $\in \mathfrak{I}^{\mathbb{Q}}$) we have $f([a_1, b_1] \times \cdots \times [a_n, b_n]) = \hat{f}(a_1, b_1, \cdots, a_n, b_n)$, i.e., the image of every n-dimensional box (resp. with rational endpoints) is an interval (resp. with rational endpoints).

We restrict the primitive functions to those that are interval preserving to ensure that interval-based reasoning is compatible with primitive operations. As the following shows, most interesting functions (including e.g. $+, \cdot, -, \exp, |\cdot|, \cdots$) are interval preserving.

Lemma 3.2. *If $f : \mathbb{R}^n \to \mathbb{R}$ is continuous then f is interval preserving.*

$$\frac{b \le 0}{\langle \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}), \wp\rangle \rightsquigarrow \langle \mathcal{N}, \wp\rangle} \qquad \frac{a > 0}{\langle \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}), \wp\rangle \rightsquigarrow \langle \mathcal{P}, \wp\rangle}$$

$$\frac{}{\langle \mathtt{sample}, [a,b] :: \wp\rangle \rightsquigarrow \langle \underline{[a,b]}, \wp\rangle} \qquad \frac{0 \le a}{\langle \mathtt{score}(\underline{[a,b]}), \wp\rangle \rightsquigarrow \langle \underline{[a,b]}, \wp\rangle}$$

$$\langle f(\underline{[a_1,b_1]}, \cdots, \underline{[a_{|f|},b_{|f|}]}), \wp\rangle \rightsquigarrow \langle \underline{\hat{f}(a_1,b_1,\cdots,a_{|f|},b_{|f|})}, \wp\rangle$$

Figure 3. Selection of interval-based reduction rules

3.1 Interval-Based Semantics

We define the set of interval traces by $\mathbb{S}_{\mathfrak{I}} := \bigcup_{n\in\mathbb{N}} \mathfrak{I}_{0,1}^n$, i.e., finite sequences of intervals with endpoints between 0 and 1 (inclusive). We let $\wp$ range over elements in $\mathbb{S}_{\mathfrak{I}}$. To avoid confusion, we shall refer to elements of $\mathbb{S}_{\mathfrak{I}}$ as *interval traces*, and elements of $\mathbb{S}$ as *standard traces*.

Redexes and evaluation contexts of interval terms are defined as expected. As we only replace real-valued numerals with interval-valued, our standard small-step semantics (Fig. 2) mostly extends to interval terms. The specific reduction rules concerning the control flow and primitive functions are given in Fig. 3. As a useful intuition, it is helpful to view an interval numeral $\underline{[a,b]}$ as an unknown value within that interval. As in the standard semantics, we are interested in the interval traces that lead to a normal form.

$$\mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\mathrm{term}} := \{\wp \in \mathbb{S}_{\mathfrak{I}} \mid \exists \mathcal{V} : \langle \mathcal{M}, \wp\rangle \rightsquigarrow^* \langle \mathcal{V}, \epsilon\rangle\}$$

For any $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\mathrm{term}}$, we define $\#^{\wp}_{\downarrow}(\mathcal{M})$ as the number of reduction steps to termination.

Embedding Into Intervals. While we want to analyse the termination probability of standard terms, our interval-based semantics builds on interval terms. We define a natural embedding $(\cdot)^{2\mathfrak{I}}$ that maps every standard term M to the interval term $M^{2\mathfrak{I}}$ obtained by replacing every numeral $\underline{r}$ by the interval numeral $\underline{[r,r]}$. Our soundness and completeness results are now based on the operational behavior of $M^{2\mathfrak{I}}$ (in the interval semantics), and they allow us to draw conclusions about the behavior of M (in the standard semantics).

3.2 Soundness

We now show that the interval-based semantics gives lower bounds on the probability of termination in the standard semantics. We define the *weight of an interval trace* $\wp$, denoted by $\omega(\wp)$, in the obvious way:

$$\omega([a_1,b_1],\cdots,[a_n,b_n]) := \textstyle\prod_{i=1}^n (b_i - a_i)$$

To combine the weight of multiple terminating interval traces we need to ensure that the interval traces are disjoint, i.e., we do not account twice for the same standard trace.

Definition 3.3. Two interval traces $\wp = [a_1,b_1],\cdots,[a_n,b_n]$ and $\wp' = [a'_1,b'_1],\cdots,[a'_m,b'_m]$ are *compatible* if $n \neq m$ or there exists i such that $b_i \le a'_i$ or $b'_i \le a_i$.

For example, the four interval traces, $[0,1][0,\frac{1}{3}]$, $[0,1][\frac{1}{3},\frac{1}{2}]$, $[0,1][\frac{3}{4},1]$ and $[0,1]$, are pairwise compatible. For a *countable* set of interval traces A we define $\omega(A) := \sum_{\wp\in A} \omega(\wp)$; if $A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\mathrm{term}}$ we also define the expected value of A, denoted $\mathbb{E}(\mathcal{M}, A)$, by

$$\mathbb{E}(\mathcal{M}, A) := \textstyle\sum_{\wp\in A} \omega(\wp) \cdot \#^{\wp}_{\downarrow}(\mathcal{M})$$

We can now state soundness as follows:

Theorem 3.4. *For every countable set of* pairwise compatible *traces* $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},term}$ *the following holds:*

- $\omega(A) \le \mathbb{P}_{term}(M)$
- $\mathbb{E}(M^{2\mathfrak{I}}, A) \le \mathbb{E}_{term}(M)$

This (perhaps unsurprising) soundness result is the basis of an effective tool to verify lower bounds on $\mathbb{P}_{\mathrm{term}}(M)$ and $\mathbb{E}_{\mathrm{term}}(M)$. The real force of the interval-based semantics lies in its completeness.

3.3 Completeness

We show that, under mild assumptions on the primitive functions, a countable number of traces for $M^{2\mathfrak{I}}$ already gives the exact probability of termination $\mathbb{P}_{\mathrm{term}}(M)$. Consequently, by an incremental search of terminating interval-traces, we can compute arbitrarily tight lower bounds on $\mathbb{P}_{\mathrm{term}}(M)$.

Example 3.5. Consider the term

$$M = \left(\mu^{\varphi}_x.\mathtt{if}\ \mathtt{sample} + \mathtt{sample} - \underline{1}\ \mathtt{else}\ x\ \mathtt{else}\ \varphi\, x\right) \underline{0}.$$

For the moment we focus on the set of traces on which this term terminates *without* making a single recursive call which is $T = \{r_1\, r_2 \in \mathbb{R}^2_{[0,1]} \mid r_1 + r_2 \le 1\}$. This set cannot be described by a countable union of interval traces, i.e., there are no interval traces $\{\wp_i\}_{i\in\mathbb{N}}$ such that $\boldsymbol{s} \in T \Leftrightarrow \exists i \in \mathbb{N} : \boldsymbol{s} \triangleleft \wp_i$, where $\boldsymbol{s} \triangleleft \wp_i$ means that $\boldsymbol{s}$ refines $\wp_i$ (see [4]). Nevertheless, as M is AST, our completeness result states that we can find a countable family of (pairwise compatible) interval traces, $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\mathrm{term}}$, whose cumulative weight (i.e. $\omega(\mathcal{A})$) equals $\mathbb{P}_{\mathrm{term}}(M) = 1$.

To achieve completeness we need the concept of *interval separable* primitive functions. For measurable $A, B \subseteq \mathbb{R}^n$ we write $A \Subset B$ if $A \subseteq B$ and $\lambda_n(B \setminus A) = 0$, i.e., A is contained in, and, up to a null set, equal to, B. Interval separability now states that the preimage of every interval can be written, up to a null set, as a countable union of boxes. Precisely:

Definition 3.6. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *interval separable* if for every interval $[a,b] \in \mathfrak{I}$, there exists a family of *boxes* $\{B_i\}_{i\in\mathbb{N}}$ with $B_i \subseteq \mathbb{R}^n$ such that $\cup_i B_i \Subset f^{-1}([a,b])$.

Most interesting functions such as $+$, $\cdot$, exp, etc. are interval separable.

Lemma 3.7. *If* $f : \mathbb{R}^n \to \mathbb{R}$ *is continuous, and for all* $y \in \mathbb{R}$, $f^{-1}(\{y\})$ *is a Lebesgue null set, then* f *is interval separable.*

Theorem 3.8. *If every $f \in \mathbb{F}$ is interval separable, then for every $M \in \Lambda_0$ there exists a countable set of pairwise-compatible interval traces $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},term}$ such that $\omega(A) = \mathbb{P}_{term}(M)$; and if M is AST then $\mathbb{E}(M^{2\mathfrak{I}}, A) = \mathbb{E}_{term}(M)$.*

Proof Sketch. We first partition $\mathbb{T}_{M,\text{term}}$ according to the branching behaviour, i.e., sequences in $\{0,1\}^*$ indicating if the left or the right branch of conditionals was taken. We then fix a branching behaviour (notice that $\{0,1\}^*$ is countable) and employ *stochastic symbolic execution* (in the sense of [41]) by executing a term on a trace of variables, while collecting symbolic constraints along the way. As primitive functions are interval separable, we show that the corresponding constraints can be exhausted via interval traces. □

Incompleteness. While the collection of primitive functions with respect to which our semantics is complete is very broad (c.f. Lem. 3.7), interval-based reasoning is incomplete in the presence of arbitrary continuous functions.

Example 3.9. Let $C \subseteq \mathbb{R}$ be any Smith-Volterra-Cantor set, i.e., C has positive Lebesgue measure but is nowhere dense, i.e., there are no $a < b$ with $[a,b] \in C$. Now construct function $f_C : \mathbb{R} \to \mathbb{R}$ by $f_C(x) := d(x, C)$, the distance of x to C. As C is a closed set, the function is well-defined and obviously continuous; and the roots of f_C coincide with C. Then $M := \texttt{if } f_c(\texttt{sample}) \texttt{ then } \underline{0} \texttt{ else } \underline{1}$ is clearly AST. However, in the interval-based semantics, we can never derive a termination probability of more than $1 - \lambda_1(C) < 1$ as there is no non-trivial interval trace taking the left branch.

3.4 AST and PAST in the Arithmetic Hierarchy

If we only consider functions that are $\mathbb{Q}$-interval preserving we can restrict the previous reasoning to intervals and boxes with rational endpoints. This has direct recursion-theoretic consequences.

Theorem 3.10. *Assume that every $f \in \mathbb{F}$ is $\mathbb{Q}$-interval preserving and interval separable, and $\hat{f}$ is computable and we consider CbN evaluation. For any term M (containing only rational numerals), deciding AST is in Π^0_2. If M is AST, deciding PAST is in Σ^0_2. In general, deciding PAST is in Δ^0_3.*

Proof. Thanks to Thm. 3.4 and Thm. 3.8 we can express "M is AST" by the following $\forall\exists$-formula:

$$\forall \epsilon > 0 \in \mathbb{Q}.\ \exists A.\ A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\text{term}} \wedge \omega(A) \geq 1 - \epsilon$$

where A ranges over (encodings of) *finite, pairwise compatible* sets of interval traces with *rational* endpoints. If M is AST we can express PAST (i.e. $\mathbb{E}_{\text{term}}(M) < \infty$) as this $\exists\forall$-formula:

$$\exists c \in \mathbb{Q}.\ \forall A.\ A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\text{term}} \Rightarrow \mathbb{E}(M^{2\mathfrak{I}}, A) \leq c$$

In general, M-is-PAST $\Leftrightarrow$ M-is-AST $\wedge\ \mathbb{E}_{\text{term}}(M) < \infty$, so the general PAST decision problem is in Δ^0_3. □

If addition is definable, then—thanks to the hardness results in [29]—deciding AST in the presence of continuous distributions (and suitable primitive functions) is Π^0_2-complete; and deciding PAST (assuming AST) is Σ^0_2-complete.[7] We remark that the Δ^0_3 upper bound for the general PAST problem does not match the corresponding bound for discrete distributions [29]. The approach in [29] uses the fact that there are finitely many traces of a given length; this property obviously does not hold in the presence of continuous distributions.

4 Intersection Type System

Intersection types have long been studied in termination analysis as they can give a complete characterisation of termination: A λ-term is typable in a (suitable) intersection type system iff it is strongly normalising. A first study of the quantitative notion of AST, and whether the intriguing completeness of intersection types can be extended to a probabilistic language, was conducted in [9]. Owing to the intrinsic Π^0_2-hardness of AST [29], we cannot hope for a semi-decidable type system in which a term is typable iff it is AST. Instead [9] presented two approaches to termination analysis, where the probability of termination is either a sum over all (countably many) typing derivation (called the oracle system) or the least upper bound (lub) thereof. We show that completeness of intersection types w.r.t. termination can also be established for a language with continuous samples (where a program admits uncountably many distinct runs), thereby giving a *local representation* of our interval-based semantics. In our system the lub over countably many derivations gives the probability of termination *and* the expected number of computation steps. Thus we obtain a complete, compositional and recursion-theoretically optimal method for computing lower bounds on both the probability of termination and the expected time to termination.

Intersection Type System for SPCF. Our system conceptually lies between the two approaches of [9] (alluded to above): we reason about the lub, and at the same type explicitly enumerate terminating (interval) traces as in the oracle system of [9]. The system in [9] relies on the countable nature of the execution tree and can exhibit subject reduction by taking the weighted (finite) sum over the reduction relation. This approach does not work for SPCF because of the uncountable nature of the latter. Instead, our proofs hinge on the soundness and completeness of the interval-based semantics (Sec. 3).

Set Types. We define *set types* by the following grammar:

$$\alpha := [a,b] \mid \sigma \to \mathcal{A} \qquad \sigma := \{\mathcal{A}_1, \cdots, \mathcal{A}_n\}$$
$$\mathcal{A} := \{(\alpha_1, \wp_1, \tau_1), \cdots, (\alpha_m, \wp_m, \tau_m)\}$$

where each $\wp_i$ is an interval trace, and τ_i a natural number. We refer to elements σ as *intersections* and $\mathcal{A}$ as *set types*.

[7] The reduction from the complement of the of the universal halting problem used to establish Σ^0_2-hardness of deciding PAST [29, Thm. 8] always yields programs that are AST. It is therefore Σ^0_2-hard to decide PAST even if the program in question is already assumed AST.

$$\frac{\mathcal{A} \in \sigma}{\Gamma, x : \sigma \vdash x : \mathcal{A}}\ \textbf{(var)} \qquad \frac{}{\Gamma \vdash \underline{[a,b]} : \left\{([a,b], \epsilon, 0)\right\}}\ \textbf{(num)}$$

$$\frac{\{[a_i, b_i]\}_{i \in [n]} \text{ are almost disjoint}}{\Gamma \vdash \texttt{sample} : \left\{([a_i, b_i], [a_i, b_i], 1) \mid i \in [n]\right\}}\ \textbf{(sample)}$$

$$\frac{\Gamma, x : \sigma, \varphi : \gamma \vdash \mathcal{M} : \mathcal{A} \qquad \left\{\Gamma \vdash \mu_x^{\varphi}.\mathcal{M} : \mathcal{B} \mid \forall \mathcal{B} \in \gamma\right\}}{\Gamma \vdash \mu_x^{\varphi}.\mathcal{M} : \left\{(\sigma \to \mathcal{A}, \epsilon, 0)\right\}}\ \textbf{(fix)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \{\Gamma \vdash \mathcal{N} : C \mid (\sigma \to \mathcal{B}, \wp, \tau) \in \mathcal{A}, C \in \sigma\}}{\Gamma \vdash \mathcal{M}\mathcal{N} : \bigcup_{(\sigma \to \mathcal{B}, \wp, \tau) \in \mathcal{A}} \mathcal{B}^{(\uparrow \wp, \tau+1)}}\ \textbf{(app)}$$

$$\frac{}{\Gamma \vdash \mathcal{M} : \{\}}\ (\{\!\}) \qquad \frac{\Gamma, x : \sigma \vdash \mathcal{M} : \mathcal{A}}{\Gamma \vdash \lambda x.\mathcal{M} : \left\{(\sigma \to \mathcal{A}, \epsilon, 0)\right\}}\ \textbf{(abs)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A}}{\Gamma \vdash \texttt{score}(\mathcal{M}) : \left\{([a,b], \wp, \tau+1) \mid ([a,b], \wp, \tau) \in \mathcal{A}, a \geq 0\right\}}\ \textbf{(score)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \{\Gamma \vdash \mathcal{N} : \mathcal{B}_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}, b \leq 0\} \qquad \{\Gamma \vdash \mathcal{P} : C_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}, a > 0\}}{\Gamma \vdash \texttt{if}(\mathcal{M}, \mathcal{N}, \mathcal{P}) : \bigcup_{([a,b],\wp,\tau) \in \mathcal{A} \mid b \leq 0} \mathcal{B}_{([a,b],\wp,\tau)}^{(\uparrow \wp, \tau+1)} \cup \bigcup_{([a,b],\wp,\tau) \in \mathcal{A} \mid a > 0} C_{([a,b],\wp,\tau)}^{(\uparrow \wp, \tau+1)}}\ \textbf{(if)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \{\Gamma \vdash \mathcal{N} : \mathcal{B}_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}\}}{\Gamma \vdash f(\mathcal{M}, \mathcal{N}) : \bigcup_{([a,b],\wp,\tau) \in \mathcal{A}} \bigcup_{([c,d],\wp',\tau') \in \mathcal{B}_{([a,b],\wp,\tau)}} \left\{(\hat{f}(a,b,c,d), \wp\wp', \tau + \tau' + 1)\right\}}\ (f_2)$$

Figure 4. Intersection type system for SPCF

To effectively type conditionals we need to integrate first-order data, in our case intervals, in the types themselves. This is similar to the type system in [21]. For a set type $\mathcal{A} = \left\{(\alpha_i, \wp_i, \tau_i)\right\}_i$ we write $\mathcal{A}^{(\uparrow \wp, \tau)}$ for the set type $\left\{(\alpha_i, \wp\,\wp_i, \tau_i + \tau)\right\}_i$, i.e., the set obtained by prepending $\wp$ to every trace and adding τ to every count. We call two interval *almost disjoint* if their intersection contains at most one element.

Type System. Typing judgments are of the form $\Gamma \vdash \mathcal{M} : \mathcal{A}$. Valid judgments are defined by induction over the rules in Fig. 4. Intuitively, if $\vdash \mathcal{M} : \left\{(\alpha_i, \wp_i, \tau_i)\right\}_i$ then $\wp_i$ are all terminating traces for M on which exactly τ_i steps are made until a value is reached. We advise the reader to compare this system with the monadic system given in [9, §6.1]. Note, in particular, that the type of an application is determined by the left argument, matching the CbN β-reduction where arguments are passed unevaluated. While the **(if)** -rule looks complicated at first sight, the subscript $([a,b], \wp, \tau)$ for each set type is merely used as an index, i.e., if $\Gamma \vdash \mathcal{M} : \mathcal{A}$ we can combine a different type derivation for every element in $\mathcal{A}$. Although we restrict primitive functions to have arity 2, the rules can easily be extended to handle higher arities. We omitted the general rule as it gets chaotic. For set type $\mathcal{A} = \left\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\right\}$ we define $\omega(\mathcal{A}) := \sum_{i \in [n]} \omega(\wp_i)$ and $\mathbb{E}(\mathcal{A}) := \sum_{i \in [n]} \omega(\wp_i) \cdot \tau_i$. We can then show correctness.

Theorem 4.1. *For every term $M \in \Lambda_0$,*

1. $\bigvee_{\vdash M^{2\mathfrak{Z}} : \mathcal{A}} \omega(\mathcal{A}) = \mathbb{P}_{term}(M)$, *and*
2. *If M is AST,* $\bigvee_{\vdash M^{2\mathfrak{Z}} : \mathcal{A}} \mathbb{E}(\mathcal{A}) = \mathbb{E}_{term}(M)$

This gives a recursion-theoretically optimal characterisation of AST that is purely based on the type system (c.f. [9, §5.3]). Thus we can computationally analyse termination, not just by evaluation (c.f. Sec. 3), but also via a local typing system. By incrementally searching for typing derivations, we can compute arbitrarily tight bounds. Compared to [9], a novel feature of our work lies in the fact that we can explicitly reason about execution time, thus enabling a type-based characterisation of PAST (for terms that are AST). While our system as a whole may look a little intimidating, each rule is actually simple by itself, requiring no complex operations. The idea of annotating types by a step count is also applicable to the setting of [9], giving a strict generalisation of their system. Our system can also be easily generalised to the untyped λ-calculus considered in [6]. Lastly, while our correctness proof hinges on the completeness of the interval-based semantics, we can present the system without referring to interval traces directly, and instead consider probability mass functions on types (as done in e.g. [9]).

5 Counting-based Recursion Analysis

We now turn our attention to devising a method that, unlike the preceding approach, can prove AST efficiently. We focus on programs that can make multiple recursive calls from distinct call sites (during evaluation of the recursive body); we call such recursion *non-affine*. As evident from Ex. 1.1, non-affine recursion complicates AST analysis considerably. Intuitive results such as the *zero-one law of termination*[3] [43] are only valid for affine recursion.

Our framework builds on the idea of counting. We show that analysis of the resulting distributions on natural numbers suffices for proving AST of the program. As a corollary we obtain a functional generalisation of the *zero-one law*, which specialises to the original law in case the recursion is affine. Our approach to proving non-affine recursion can be viewed as orthogonal to [36] (which is restricted to affine recursion): rather than using the size-related information of the recursive function argument, we count the number

of recursive calls from distinct call sites in the evaluation of the body of the recursion. Moreover, compared to [36], our approach supports continuous distributions, and it is not restricted to binary probabilistic choice. Compared to techniques based on ranking functions – a dominant approach to AST verification [1, 13, 16, 17, 23, 31, 43, 44], our method is fully automatic and easy to implement, as we show in Sec. 6.

Example 5.1. Let's revisit the 3d printing company (Ex. 1.1). The new situation is that the staff gets tired over time and prints an incorrect number of copies. In case the print is faulty, there is a probability $sig(x)$ of the operator becoming tired and making mistakes, where sig is the sigmoid function. (Thus with increasing time (x), the probability of making mistakes approaches 1.) The operator's mistake takes the form of printing 3 instead of the intended 2 copies with probability .5. We model this scenario by the following term:

$$\mu_x^{\varphi}.x \oplus_p \left(\left(\varphi^3(x+1) \oplus \varphi^2(x+1)\right) \oplus_{sig(x)} \varphi^2(x+1) \right).$$

The question now becomes: for which p is this term AST?

We keep track of the number of calls by extracting a *counting distribution*, a (sub) pmf on $\mathbb{N}$, that models the distribution on new calls made. To account for the fact that a recursive function that is called n times (inclusive of the original call) contributes $n-1$ to the total number of *pending calls*[8], we shift the counting pattern by -1, obtaining a (sub) pmf on $\mathbb{Z}$. The counting distribution is analysed via a random walk whose current value can be seen as the number of pending calls. In this section, we first introduce the necessary tools to analyse a random walk (Sec. 5.1), then present the extraction of the counting distribution from programs (Sec. 5.2), and the soundness theorem that relates termination behavior of the shifted random walk with that of the non-affine recursive program in question (Sec. 5.3).

5.1 Random Walk on $\mathbb{N}$

Assume a countable state space X. A *stochastic matrix on* X is a function $\mathfrak{P} : X \times X \to \mathbb{R}_{[0,1]}$ such that $\sum_{y \in X} \mathfrak{P}(x,y) = 1$ for every $x \in X$ (see e.g. [2, §10.1] or [45]). $\mathfrak{P}(x,y)$ gives the probability of transitioning from x to y. Given stochastic matrices $\mathfrak{P}$ and $\mathfrak{P}'$, we write $\mathfrak{P}\,\mathfrak{P}'$ for their product, which is a stochastic matrix; and $\mathfrak{P}^n$ for the n-fold product of $\mathfrak{P}$.

We consider Markov chains whose step behaviour is definable in terms of relative change, independently of the current state. The relative change in each step is given by a *step distribution* which is a (sub)probability mass function $s : \mathbb{Z} \to \mathbb{R}_{[0,1]}$. We call s *finite* if it has finite support. We interpret the "missing probability", $1 - \sum_{i \in \mathbb{Z}} s(i)$, as failure.

Definition 5.2. Given a step distribution $s : \mathbb{Z} \to \mathbb{R}_{[0,1]}$ we define a stochastic matrix $\mathfrak{P}_s$ on $\mathbb{N}_\perp := \mathbb{N} \cup \{\perp\}$ by:

	$\perp$	0	$m > 0$
$\perp$	1	0	0
0	0	1	0
$n > 0$	$1 - \sum_{i \in \mathbb{Z}} s(i)$	$\sum_{i \le -n} s(i)$	$s(m-n)$

Note that s gives the relative change in each step, the walk is truncated (trapped) at 0, and the probability mass deficit in s (if any) is balanced by the probability of transitioning (from a good state) to the failure state $\perp$. We call s AST if the associated walk reaches state 0 a.s.

Definition 5.3. A step distribution s is called *AST* if for every $m \in \mathbb{N}$, $\lim_{n \to \infty} \mathfrak{P}_s^n(m,0) = 1$.

Note that the limit in the definition above always exists (and lies between 0 and 1) as the sequence $(\mathfrak{P}_s^n(m,0))_n$ is monotone *increasing* and bounded. A step distribution can be shown AST by reduction to a one-counter Markov decision process (MDP) (following [36]), for which a.s. termination can be decided in polynomial time [7]. We present a new proof that avoids the detour to MDPs, giving a tighter (in fact optimal) complexity upper bound than that in [36]. The crux lies in a simple, and decidable—if s is finite and rational valued—characterisation of AST which directly gives *linear-time decidability*.

Theorem 5.4. *A finite step distribution s is AST if and only if all of the following hold*

a) $\sum_{i \in \mathbb{Z}} s(i) = 1$ *b)* $s \neq \delta_0$ *c)* $\sum_{i \in \mathbb{Z}} i \cdot s(i) \le 0$

Uniform AST. We also model the case where in each time step, a different distribution can be chosen from an available set of step distributions, similar to a MDP [2].

Definition 5.5. A family of step distributions $\{s_i\}_{i \in \mathcal{I}}$ is *uniform AST* if for every $m \in \mathbb{N}$

$$\lim_{n \to \infty} \left(\inf_{i_1, \cdots, i_n} \mathfrak{P}_{s_{i_1}} \cdots \mathfrak{P}_{s_{i_n}}(m,0) \right) = 1$$

Informally it reads that as step count n tends to ∞, no matter which step distribution from $\{s_i\}_{i \in \mathcal{I}}$ is chosen at each step, the walk eventually reaches 0 almost surely. Obviously, uniform AST implies AST for each of the s_i but, in general, not conversely. However we can show:

Lemma 5.6. *If $\{s_i\}_{i \in \mathcal{I}}$ is a* finite *family of step distributions and each s_i is AST then $\{s_i\}_{i \in \mathcal{I}}$ is uniform AST.*

5.2 Counting-Based Extraction of Random Walks

Let's fix a *1st-order* program $\mu_x^{\varphi}.M$ with *no* nested recursion. To extract the counting pattern of $\mu_x^{\varphi}.M$, we instrument a counting-based reduction relation $\xrightarrow{\star}$, and use it to analyse a related term $\mathbf{body}_{\mu_x^{\varphi}.M}(r) := M[\underline{r}/x, \boxed{\mu}/\varphi]$, i.e., the body of the program $\mu_x^{\varphi}.M$, with x instantiated to a fixed actual argument $\underline{r}$, and a special symbol $\boxed{\mu}$ in place of all recursive calls. The counting-based reduction relation $\xrightarrow{\star}$ is presented in Fig. 5 and acts on configuration of the form $\langle N, \mathbf{s}, n \rangle$ where

[8] equivalently, the maximum number of stack frames on the function's call stack

$$\frac{}{\langle (\lambda x.M)V, s, n\rangle \xrightarrow{\star} \langle M[V/x], s, n\rangle} \qquad \frac{}{\langle \boxed{\mu} V, s, n\rangle \xrightarrow{\star} \langle \star, s, n+1\rangle} \qquad \frac{}{\langle \mathtt{sample}, r :: s, n\rangle \xrightarrow{\star} \langle \underline{r}, s, n\rangle}$$

$$\frac{r \le 0}{\langle \mathtt{if}(\underline{r}, N, P), s, n\rangle \xrightarrow{\star} \langle N, s, n\rangle} \qquad \frac{r > 0}{\langle \mathtt{if}(\underline{r}, N, P), s, n\rangle \xrightarrow{\star} \langle P, s, n\rangle} \qquad \frac{}{\langle f(\underline{r_1}, \cdots, \underline{r_{|f|}}), s, n\rangle \xrightarrow{\star} \langle \underline{f(r_1, \cdots, r_{|f|})}, s, n\rangle}$$

$$\frac{}{\langle f(V_1, \cdots, \star, \cdots, V_{|f|}), s, n\rangle \xrightarrow{\star} \langle \star, s, n\rangle} \qquad \frac{r \ge 0}{\langle \mathtt{score}(\underline{r}), s, n\rangle \xrightarrow{\star} \langle \underline{r}, s, n\rangle} \qquad \frac{\langle R, s, n\rangle \xrightarrow{\star} \langle M, s', n'\rangle}{\langle E[R], s, n\rangle \xrightarrow{\star} \langle E[M], s', n'\rangle}$$

Figure 5. Small-step reduction rules for $\xrightarrow{\star}$

$n \in \mathbb{N}$ counts recursive calls. The main idea is to replace outcomes of recursive calls by a distinguished value $\star$ of type **R** which stands for an unknown numeral. Note that the unknown numeral $\star$ can end up in the guard of a conditional if recursive outcomes affect the control flow of the program. This is, however, unavoidable if we want to count recursive calls (without reference to the program denotation) as the number of function call sites can depend on the (probabilistic) outcome of a prior call. We define

$$\mathbb{T}^{\star}_{N;n} := \{s \in \mathbb{S} \mid \exists V : \langle N, s, 0\rangle \xrightarrow{\star}{}^{*} \langle V, \epsilon, n\rangle\}$$

the set of traces on which recursive calls from exactly n *distinct* call sites are made. As the reduction relation is deterministic we get that $\{\mathbb{T}^{\star}_{N;n}\}_{n\in\mathbb{N}}$ are pairwise disjoint. Using similar arguments in [6], it is easy to see that $\mathbb{T}^{\star}_{N;n}$ is a measurable set of traces.

Definition 5.7. Given a term $\mu^{\varphi}_x.M$ we define the r-indexed family $\{[\![\mu^{\varphi}_x.M \mid r]\!] : \mathbb{N} \to \mathbb{R}_{[0,1]}\}_{r\in\mathbb{R}}$, called the *counting pattern of* $\mu^{\varphi}_x.M$, whereby

$$[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(n) := \mu_{\mathbb{S}}\big(\mathbb{T}^{\star}_{\mathbf{body}_{\mu^{\varphi}_x.M}(r);n}\big)$$

In words, $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(n)$ gives the probability of *a run of* $\mu^{\varphi}_x.M$, on the actual argument r, making recursive calls from n distinct call sites. It is straightforward to see that for every r we have $\sum_n [\![\mu^{\varphi}_x.M \mid r\,]\!]\,(n) \le 1$, by the same argument as in [41, Lem. 7].

Example 5.8. Consider the term $\mu^{\varphi}_x.M$ from Ex. 5.1. We get $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(0) = p$, $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(1) = 0$, $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(2) = (1-p)\cdot\frac{1}{2}\cdot(2 - sig(r))$, $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(3) = (1-p)\cdot\frac{1}{2}\cdot sig(r)$ and $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(n) = 0$ for all other n.

5.3 Termination via Counting Patterns

Our main result of this section is that we can use the counting pattern of a program to soundly reason about its termination property. For any *counting distribution*, i.e., (sub) pmf $s : \mathbb{N} \to \mathbb{R}_{[0,1]}$, we define the shifted step distribution $\overline{s} : \mathbb{Z} \to \mathbb{R}_{[0,1]}$ by $\overline{s}(z) = s(z+1)$ for $z \ge -1$ and $\overline{s}(z) = 0$ otherwise [9] .

[9]The shifting of the distribution accounts for the fact that resolving a recursive call by making n recursive calls changes the number of pending calls by $n-1$. In the extreme case, making no recursive call, decreases the number of pending calls by 1.

Theorem 5.9. *If* $\{\overline{[\![\mu^{\varphi}_x.M \mid r]\!]}\}_{r\in\mathbb{R}}$ *(qua family of step distributions) is uniform AST then* $\mu^{\varphi}_x.M$ *is AST on every actual argument.*

Proof Sketch. We decompose the set of terminating traces on a fixed argument according to the arguments of recursive calls arranged in a tree. We can lower bound the probability of each partition in terms of $\{\overline{[\![\mu^{\varphi}_x.M \mid r]\!]}\}_{r\in\mathbb{R}}$ and show that uniform AST implies that the cumulative weight over every decomposed part equals 1, i.e., the program is AST. □

A Partial Order for Counting Distributions. We can equip the set of counting distributions (i.e. (sub)pmfs $s, t : \mathbb{N} \to \mathbb{R}_{[0,1]}$) with a partial order that is compatible with the termination behavior. We define

$$s \sqsubseteq t \Leftrightarrow \forall n \in \mathbb{N}.\ \textstyle\sum_{m\le n} s(m) \le \sum_{m\le n} t(m)$$

i.e., $s \sqsubseteq t$ if the cumulative weight of s is no greater than that of t at any point. It is easy to see that $\sqsubseteq$ is a partial order. Furthermore, we can show compatibility w.r.t. AST (using Thm. 5.4):

Lemma 5.10. *If* s, $\{t_i\}_{i\in\mathcal{I}}$ *are counting distributions and for all* $i \in \mathcal{I}$, $s \sqsubseteq t_i$ *and* $\overline{s}$ *is AST then* $\{\overline{t_i}\}_{i\in\mathcal{I}}$ *is uniform AST.*

Example 5.11. The counting pattern presented in Ex. 5.8 for the term from Ex. 5.1 satisfies the preconditions of Lem. 5.10 for $s := p\delta_0 + (1-p)\frac{1}{2}\delta_2 + (1-p)\frac{1}{2}\delta_3$ (where δ_i denotes the Dirac-distribution). For $p \ge \frac{3}{5}$, we can deduce that the counting pattern is uniform AST (via Lem. 5.10 and Thm. 5.4) and thus the example is AST on every input (via Thm. 5.9).

5.4 ϵ-Recursion Avoiding Fixpoint Terms

An interesting quantity that arises from analysing Thm. 5.9 is $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(0)$, i.e., the probability of a run of $\mu^{\varphi}_x.M$ (on argument $\underline{r}$) making no further recursive calls. Let's consider programs where this probability has a positive lower bound.

Definition 5.12. A recursive program $\mu^{\varphi}_x.M$ is *ϵ-recursion avoiding (ϵ-RA)* if for all $r \in \mathbb{R}$, $[\![\mu^{\varphi}_x.M \mid r\,]\!]\,(0) \ge \epsilon$.

Lets assume $\sum_n [\![\mu^{\varphi}_x.M \mid r\,]\!]\,(n) = 1$, i.e., the $\xrightarrow{\star}$-reduction is never stuck. In [4] we show how this can be statically ensured via a type system. Note that a program may be ϵ-RA for a positive ϵ, and yet not AST (as evident from Ex. 1.1). To ensure AST, the positive probability ϵ must be "large enough",

in relation to the number of recursive calls. We define the *recursive rank* of $\mu_x^\varphi.M$ to be the minimal m such that for all $n > m$, and r, $[\![\mu_x^\varphi.M \mid r]\!](n) = 0$ (or, equivalently, the maximal number of call sites from which recursive calls are made in a run of $(\mu_x^\varphi.M)\ \underline{r}$, for any r). (In [4], we show that the recursive rank can be upper bounded via a decidable non-idempotent intersection type system.) Now, using Thm. 5.4 combined with Thm. 5.9, we can get an easy corollary:

Corollary 5.13. *If $\mu_x^\varphi.M$ has recursive rank m and is ϵ-RA for some $\epsilon > 0$ that satisfies $m(1-\epsilon) \leq 1$ then $\mu_x^\varphi.M$ is AST on every argument.*

Example 5.14. The program **(2)** in Ex. 1.1 has recursive rank 2, and is p-RA. So Cor. 5.13 is applicable whenever $2(1-p) \leq 1 \Leftrightarrow p \geq \frac{1}{2}$. Note that Cor. 5.13 is weaker than Thm. 5.9; for example, Cor. 5.13 on Ex. 5.1 is only applicable for $p \geq \frac{2}{3}$ whereas Thm. 5.9 is applicable for $p \geq \frac{3}{5}$ (Ex. 5.11).

Example 5.15. As a further example, consider yet another variation to our 3d-printing program from Ex. 5.1:

$$\mu_x^\varphi.\texttt{let}\ e = \texttt{sample}\ \texttt{in}\ \texttt{if}\ e \leq p\ \texttt{then}\ x\ \texttt{else}$$
$$\Big(\big(\varphi^3(x+1) \oplus_e \varphi^2(x+1)\big) \oplus_{sig(x)} \varphi^2(x+1)\Big)$$

We sample the error value e (the higher e is, the more damaged the print) and accept the print whenever $e \leq p$. If the print is unacceptable, we replace the binary choice in Ex. 5.1 with one that depends on the sampled value of e. In the extended version [4] we show how Thm. 5.9 and Lem. 5.10 can be used to prove this program AST whenever $p \geq \sqrt{7} - 2 \approx 0.646$. As this example illustrates well, termination analysis of terms that use continuous random samples as first-class values can become very intricate. Such examples are not expressible in PHORS [33] or with binary probabilistic choice [30, 36, 50]. Our framework can analyse such examples efficiently, even automatically.

Special Case: Affine Recursion. Every affine-recursive program [36, §4.1] has recursive rank at most 1, so by Cor. 5.13, ϵ-RA for *any* $\epsilon > 0$ implies AST. This can be seen as the functional equivalent of the *zero-one-law for termination* (c.f. [43, Sec. 2.6]). However, the real novelty of our result lies in the fact that sophisticated methods are necessary to deal with the case of non-affine recursion. Our proof rules (Thm. 5.9 and Cor. 5.13) give a powerful tool to verify AST for non-affine programs where the standard zero-one law fails. Similarly to the language studied e.g. in [40], we can use this to design languages that are AST-by-construction. In particular, in any probabilistic programming system we can (safely) add a special fixpoint operator that comes with the guarantee of ϵ-RA for a sufficiently large ϵ, whose size can be determined statically via the recursive rank. This corresponds to a generalization of the stochastic while-loop in [40]. As demonstrated in [40], probabilistic programming languages with this seemingly severe restriction can still describe complex models with arbitrary precision and convergence guarantee, supporting correct inference of (AST) programs.

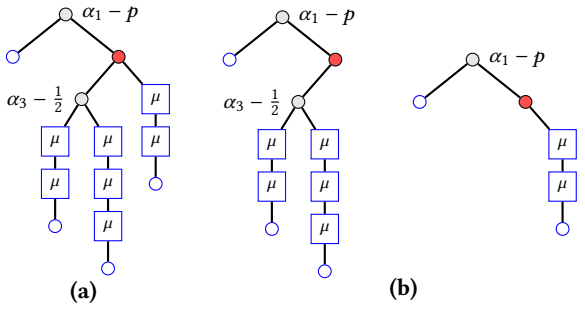

Figure 6. Symbolic execution trees for Ex. 5.1 **(a)** and all possible strategies **(b)**

6 A Proof System For Non-affine Recursion

The framework of Sec. 5 (Thm. 5.9) relies on the *counting pattern*, $\{[\![\mu_x^\varphi.M \mid r]\!]\}_{r\in\mathbb{R}}$, of the program $\mu_x^\varphi.M$. This family can contain uncountably many different counting distributions, making it impractical for analysis. As we saw in Ex. 5.8, for the counting pattern $\{[\![\mu_x^\varphi.M \mid r]\!]\}_{r\in\mathbb{R}}$ of Ex. 5.1, we have $[\![\mu_x^\varphi.M \mid r]\!] \neq [\![\mu_x^\varphi.M \mid r']\!]$ for every $r \neq r'$. So how can we automate analysis so that Thm. 5.9 can be applied without explicitly computing the counting pattern? In this section we use a simple *game-playing* perspective to solve the problem. We show that we can replace probabilistic branching that depends on the actual argument, by nondeterministic branching and thus obtain a sound method to apply Thm. 5.9. As a rule of thumb, our system can verify all programs that exhibit an AST counting pattern which is independent of the (exact values of the) actual arguments (of the recursive function in question). In this section we give an overview of our approach, and direct readers to [4] for a full account, including more complex examples.

6.1 Stochastic Symbolic Execution

The first idea we use for our system is *stochastic symbolic execution*. Instead of executing a program on a fixed trace (as done in $\xrightarrow{\star}$, Fig. 5) we evaluate on a trace of *sample variables* $(\alpha_0, \alpha_1, \cdots)$ whose values can be instantiated later. We organise execution in the form a binary tree where each branching represents a conditional which is annotated with the value on which control flow branches. We also record `score`-statements as well as recursive calls. We now replace the actual argument with an unknown value $\circledast$ (corresponding to the analysis of $\mathbf{body}_{\mu_x^\varphi.M}(\circledast)$ in Sec. 5.2). In Fig. 6a the execution tree that corresponds to the running example, Ex. 5.1, is depicted, we have coloured each branching that relies on the concrete argument $\circledast$ in red.

6.2 Strategies on Trees

As some of the branching (coloured red) depends on the actual argument, we cannot analyse it probabilistically. This can be overcome by an intuitive 2-player game reading of the execution tree: for every such node the Environment (player) can resolve its branching by an explicit strategy that indicates a left/right choice for each coloured node. For example, all possible strategies for the tree in Fig. 6a are depicted in Fig. 6b. As a strategy no longer relies on branching at nodes that contain $\circledast$, we can recover a probabilistic interpretation of paths, which is just the Lebesgue-measure of the possible assignment to the sample variables $\alpha_0, \alpha_1, \cdots$, such that a path is indeed followed. Given a strategy $\mathfrak{S}$, we denote with $\mathbb{P}(\mathfrak{S}, n)$ the probability of taking a path such that *at most n* recursive calls are made (i.e., a fixpoint node is traversed at most n times). Depending on the set of primitive functions, this value can be computed effectively. We now define the counting distribution $\mathbb{P}_{approx}$ by (where $n > 0$)

$$\mathbb{P}_{approx}(0) \coloneqq \min_{\mathfrak{S} \in Strat(\mathfrak{T})} \mathbb{P}(\mathfrak{S}, 0)$$

$$\mathbb{P}_{approx}(n) \coloneqq \left(\min_{\mathfrak{S} \in Strat(\mathfrak{T})} \mathbb{P}(\mathfrak{S}, n)\right) - \left(\min_{\mathfrak{S} \in Strat(\mathfrak{T})} \mathbb{P}(\mathfrak{S}, n-1)\right)$$

We can understand $\mathbb{P}_{approx}(n)$ as the least probability that n calls are made even if the Environment chooses in the worst (meaning maximal no. of recursive calls) possible way.

Example 6.1. Consider all strategies for the running example listed in Fig. 6b. We can compute $\mathbb{P}_{\text{approx}}(0) = p$; $\mathbb{P}_{\text{approx}}(2) = \mathbb{P}_{\text{approx}}(3) = (1-p) \cdot \frac{1}{2}$; and $\mathbb{P}_{\text{approx}}(n) = 0$ for all other n.

We can show that replacing probabilistic branching with nondeterministic one gives a lower bound (w.r.t. to the order $\sqsubseteq$) on the counting pattern.

Theorem 6.2. *For every* $r \in \mathbb{R}$, $\mathbb{P}_{\text{approx}} \sqsubseteq [\![\mu_x^{\varphi}.M \mid r]\!]$

Thus, if $\overline{\mathbb{P}_{\text{approx}}}$ is AST (which is checkable via Thm. 5.4) we get that $\{\overline{[\![\mu_x^{\varphi}.M \mid r]\!]}\}_{r \in \mathbb{R}}$ is uniform AST; and via Thm. 5.9 $\mu_x^{\varphi}.M$ is AST on every actual argument. Thm. 6.2 and the values computed in Ex. 6.1 allow us to deduce (*automatically*) that Ex. 5.1 is AST on every argument if $p \geq \frac{3}{5}$. Similarly, our tool can verify AST of Ex. 5.15 if $p \geq \sqrt{7} - 2$. For a formal description of our algorithm, correctness proof and further examples see [4].

7 Implementation

We provide prototype implementations for computing lower bound of the termination probability, and for AST verification, building on Sec. 3 and Sec. 6 respectively. This is the first prototype to compute lower bounds in the presence of continous distributions and one of the first that can automatically proof AST for non-affine recursive programs. In this section we present the results of our experiments. In the full version [4] we give a more detailed description of the algorithm and example terms. The experimental results were obtained on a Intel(R) Core i5-6200U process with 8GB of memory.

Table 1. Experimental results for lower bound computations. We give the actual probability of termination (if known), the lower bound computed (LB), the depth (d) at which we stopped the exploration and the time (t) in milliseconds. geo_p describes a geometric distribution with parameter p, $1dRW_{p,s}$ a 1-dimensional p-biased random walk starting at s [44], gr a term analysed in [50] terminating with a probability given as the inverse golden ratio, $3print_p$ the natural extension of Ex. 1.1 **(2)** to 3 recursive calls, $bin_{p,s}$ a 1-dimensional random walk in one direction [44] and *pedestrian* a stochastic program modelling a pedestrian inspired by [41]. See [4] for a detailed description of the example terms.

Term M	$\mathbb{P}_{\text{term}}(M)$	LB	d	t
$geo_{\frac{1}{2}}$	1	0.9999990463	100	78
$geo_{\frac{1}{5}}$	1	0.9995620416	200	192
$1dRW_{\frac{1}{2},1}$	1	0.8036193847	200	28223
$1dRW_{\frac{7}{10},1}$	1	0.9720964250	150	10224
gr	$\frac{\sqrt{5}-1}{2}$	0.6112594604	80	4389
Ex. 1.1, $p = \frac{1}{2}$	1	0.8318119049	90	15749
Ex. 1.1, $p = \frac{1}{4}$	$?(< 1)$	0.3328795089	90	15749
$3print_{\frac{3}{4}}$	1	0.9606655982	80	4622
$bin_{\frac{1}{2},2}$	1	0.9998493194	100	2265
pedestrian	1	0.6002376673	40	4493

Table 2. Experimental results for AST verification. For each term (all of which our tool can verify to be AST) we give the counting distribution $\mathbb{P}_{approx}$ computed by our tool (which is analysed via Thm. 5.4) and the total time t in milliseconds.

Term M	$\mathbb{P}_{approx}$	t
Ex. 1.1, **(1)**, $p = \frac{1}{2}$	$\frac{1}{2}\delta_0 + \frac{1}{2}\delta_1$	239
Ex. 1.1, **(2)**, $p = \frac{1}{2}$	$\frac{1}{2}\delta_0 + \frac{1}{2}\delta_2$	237
$3print_{\frac{2}{3}}$	$\frac{2}{3}\delta_0 + \frac{1}{3}\delta_3$	297
Ex. 5.1, $p = 0.6$	$0.6\delta_0 + 0.2\delta_2 + 0.2\delta_3$	396
Ex. 5.15, $p = 0.65$	$0.65\delta_0 + 0.06125\delta_2 + 0.28875\delta_3$	373

7.1 Lower Bounds of Termination Probability

Our prototype for lower bound computation exploits the completeness of the interval-traces semantics. Our tool combines the symbolic exploration of terms with a simple sweep algorithm to search for terminating interval traces. In the stochastic symbolic execution, we execute terms while substituting sample-variable for random outcomes. Each trace leading to a symbolic value is then analysed by iteratively searching for terminating interval traces by splitting the

unit box $[0,1]^m$ (where m is the number of sample-variables along a path). As lower bound computation is an intrinsically non-terminating process the user must specify a target depth (or timeout) at which the computation is stopped. Even in its current, unoptimized, form our tool is able to compute meaningful lower bounds in a reasonable time. We evaluate our tool on various examples taken from [33, 44, 50] and [41] (possibly modified to match the CbN evaluation). The computed lower bounds can be found in table 1. Our tool computes rational lower-bounds to avoid rounding errors. For presentation we only gave the first 10 digits of the decimal representation.

7.2 AST Verification

The challenge in implementing the ideas from Sec. 6 lies in the computation of branching probabilities, i.e., given a fixed strategy for the environment what is the probability of traversing at most n fixpoint nodes. We adopt a geometric interpretation of probability and make use of various results and implementation techniques for volume computation. For simplicity we restrict primitive operations to addition, and multiplication by a constant (and thus subtraction), as the probability of branching can be seen as the volume of a convex polytope [19] (a subset of $\mathbb{R}^d$ of the from $\{\vec{x} \mid A\vec{x} \leq b\}$). We make use of the analytic formula for this volume in [38] and its subsequent implementation in [10], and appeal to Thm. 5.4. Our implementation then performs the basic-tree operations outlined in Sec. 6 (see [4] for a fuller account) and uses [10] as a volume-computation oracle. Our prototype implementation can verify many examples, including those from Sec. 1.1, Sec. 5, and Sec. 6 (see table 1), thereby illustrating that our approach is well-suited to implementation. All of those terms can be verified to be AST in less than a second.

8 Related Work and Conclusion

Our interval-traces approach can be seen as a probabilistic interpretation of interval analysis, a standard approach to infer bounds on program variables [11, 47]. The attractive feature of intervals in our work is its completeness w.r.t. the Lebesgue measure for a broad range of primitive operations.

The only comparable lower bound computation we are aware of is presented in [33]. Kobayashi et al. show that the termination probability of CbN order-n probabilistic recursion schemes (n-PHORS) can be obtained as the least fixpoint of suitable order-$(n-1)$ fixpoint equations, which can be solved using standard Kleene fixpoint iteration. By contrast, our approach works on programs directly, and can handle continuous distributions. It is worth noting that (order-n) PHORS is readily encodable as (order-n) CbN SPCF, but the former is strictly less expressive (because the underlying recursion schemes are not Turing complete). Some interesting SPCF terms such as Ex. 5.15 cannot be expressed as PHORS. Since recursive Markov chains [22] (equivalently, probabilistic pushdown automata [8]) are essentially equivalent to 1-PHORS [33], it follows that order-1 SPCF (which contains the term in Ex. 5.15) is strictly more expressive than recursive Markov chains.

Our intersection type system is inspired by, and builds upon, the ideas of [9, 21]. However, unlike [9], we cannot prove correctness directly (because of continuous samples), rather we need to appeal to the completeness of our interval-based semantics. The step annotation of types enables us to reason about expected termination time. We conjecture these ideas to also be applicable to the system of [9]. Independent of us, [35] designed an intersection type system that is also able to reason about the expected time to termination. Their approach is, however restricted to a language with discrete samples and it is not obvious whether the approach extends to continuous samples.

Our AST verification method is closely related to [50], in that they also study recursive programs and allow for non-affine behaviour. Our work differs nonetheless in several key aspects: While they study an imperative language with discrete distributions, we work with a purely functional language with continuous distributions. Though their proposed rules can produce lower bounds on the probability of termination, they seem cumbersome to use. Their rule informally reads: if, for all n, we assume that each recursive call terminates with probability l_n after n fixpoint unfoldings, and we can prove that it terminates with probability at least l_{n+1} after $n+1$ unfoldings, then the program terminates with probability at least $\sup_n l_n$ (c.f. [50, Thm. 4.2]). In order to apply this rule, the user must manually find an explicit (and often non-trivial) sequence $(l_n)_{n\in\mathbb{N}}$. By contrast, our system provides a sound reduction to a random walk which can be analysed efficiently in linear time.

As already mentioned, our method can be seen as orthogonal to that in [36]. It is not at all obvious if their techniques can be extended to our setting with sampling from the uniform distribution. An interesting future direction is to develop a unified framework that analyses both the size-related information of the recursive function argument and the number of recursion call sites.

Conclusion. Recent advances in probabilistic programming systems and allied areas (such as [41]) provide strong impetus for the study of AST of programs with continuous distribution. We have presented a first comprehensive study of the lower bound problem, and ascertained the recursion-theoretic complexity of several termination problems. We have introduced a novel proof system for AST verification of non-affine programs which is easily implementable. While some of the existing AST proof methods support continuous distributions [15, 17, 23] the majority do not. It would be interesting to investigate if they [29, 30, 33, 36, 43, 50] can be so extended.

References

[1] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 34:1–34:32. https://doi.org/10.1145/3158122

[2] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

[3] Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics*. Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

[4] Raven Beutner and Luke Ong. 2021. On Probabilistic Termination of Functional Programs with Continuous Distributions. *CoRR* abs/2104.04990 (2021). arXiv:2104.04990 http://arxiv.org/abs/2104.04990

[5] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6. http://jmlr.org/papers/v20/18-403.html

[6] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*. ACM, 33–46. https://doi.org/10.1145/2951913.2951942

[7] Tomás Brázdil, Václav Brozek, Kousha Etessami, Antonín Kucera, and Dominik Wojtczak. 2010. One-Counter Markov Decision Processes. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. SIAM, 863–874. https://doi.org/10.1137/1.9781611973075.70

[8] Tomás Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kucera. 2013. Analyzing probabilistic pushdown automata. *Formal Methods Syst. Des.* 43, 2 (2013), 124–163. https://doi.org/10.1007/s10703-012-0166-0

[9] Flavien Breuvart and Ugo Dal Lago. 2018. On Intersection Types and Probabilistic Lambda Calculi. In *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*. ACM, 8:1–8:13. https://doi.org/10.1145/3236950.3236968

[10] Benno Büeler, Andreas Enge, and Komei Fukuda. 2000. *Exact Volume Computation for Polytopes: A Practical Study*. Birkhäuser Basel, Basel, 131–154. https://doi.org/10.1007/978-3-0348-8438-9_6

[11] Michael G. Burke. 1990. An Interval-Based Approach to Exhaustive and Incremental Interprocedural Data-Flow Analysis. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 341–395. https://doi.org/10.1145/78969.78963

[12] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software, Articles* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

[13] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

[14] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9779)*. Springer, 3–22. https://doi.org/10.1007/978-3-319-41528-4_1

[15] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. https://doi.org/10.1145/3174800

[16] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. ACM, 145–160. http://dl.acm.org/citation.cfm?id=3009873

[17] Jianhui Chen and Fei He. 2020. Proving almost-sure termination by omega-regular decomposition. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*. ACM, 869–882. https://doi.org/10.1145/3385412.3386002

[18] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*. ACM, 221–236. https://doi.org/10.1145/3314221.3314642

[19] Martin E. Dyer and Alan M. Frieze. 1988. On the Complexity of Computing the Volume of a Polyhedron. *SIAM J. Comput.* 17, 5 (1988), 967–974. https://doi.org/10.1137/0217060

[20] Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2018. Full Abstraction for Probabilistic PCF. *J. ACM* 65, 4 (2018), 23:1–23:44. https://doi.org/10.1145/3164540

[21] Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. ACM, 309–320. https://doi.org/10.1145/2535838.2535865

[22] Kousha Etessami and Mihalis Yannakakis. 2009. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM* 56, 1 (2009), 1:1–1:66. https://doi.org/10.1145/1462153.1462154

[23] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. ACM, 489–501. https://doi.org/10.1145/2676726.2677001

[24] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 84)*. PMLR, 1682–1690. http://proceedings.mlr.press/v84/ge18b.html

[25] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. AUAI Press, 220–229. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1346&proceeding_id=24

[26] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*. ACM, 167–181. https://doi.org/10.1145/2593882.2593900

[27] Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11275)*. Springer, 181–201. https://doi.org/10.1007/978-3-030-02768-1_11

[28] Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2019. Modular verification for almost-sure termination of probabilistic programs. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 129:1–129:29. https://doi.org/10.1145/3360555

[29] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost-Sure Termination. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015,*

Milan, Italy, August 24-28, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9234). Springer, 307–318. https://doi.org/10.1007/978-3-662-48057-1_24

[30] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (2018), 30:1–30:68. https://doi.org/10.1145/3208102

[31] Andrew Kenyon-Roberts and Luke Ong. 2021. Supermartingales, Ranking Functions and Probabilistic Lambda Calculus. In *36nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE Computer Society.

[32] S. C. Kleene. 1955. Hierarchies of number-theoretic predicates. *Bull. Amer. Math. Soc.* 61 (05 1955), 193–213. https://doi.org/10.1090/S0002-9904-1955-09896-3

[33] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. 2019. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 1–14. https://doi.org/10.1109/LICS.2019.8785679

[34] Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* 22, 3 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

[35] Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2021. Intersection types and (positive) almost-sure termination. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–32. https://doi.org/10.1145/3434313

[36] Ugo Dal Lago and Charles Grellois. 2019. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Trans. Program. Lang. Syst.* 41, 2 (2019), 10:1–10:65. https://doi.org/10.1145/3293605

[37] Ugo Dal Lago and Margherita Zorzi. 2012. Probabilistic operational semantics for the lambda calculus. *RAIRO Theor. Informatics Appl.* 46, 3 (2012), 413–450. https://doi.org/10.1051/ita/2012012

[38] Jean B Lasserre. 1983. An analytical expression and an algorithm for the volume of a convex polyhedron in $\mathbb{R}^n$. *Journal of optimization theory and applications* 39, 3 (1983), 363–377.

[39] Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. 2018. Reparameterization Gradient for Non-differentiable Models. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 5558–5568. https://proceedings.neurips.cc/paper/2018/hash/b096577e264d1ebd6b41041f392eec23-Abstract.html

[40] Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2020. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.* 4, POPL (2020), 19:1–19:32. https://doi.org/10.1145/3371087

[41] Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. 2021. Densities of Almost Surely Terminating Probabilistic Programs are Differentiable Almost Everywhere. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*. Springer, 432–461. https://doi.org/10.1007/978-3-030-72019-3_16

[42] Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR* abs/1404.0099 (2014). arXiv:1404.0099 http://arxiv.org/abs/1404.0099

[43] Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

[44] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A new proof rule for almost-sure termination. *Proc. ACM Program. Lang.* 2, POPL (2018), 33:1–33:28. https://doi.org/10.1145/3158121

[45] Sean Meyn and Richard L. Tweedie. 2009. *Markov Chains and Stochastic Stability* (2nd ed.). Cambridge University Press.

[46] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511813603

[47] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. 2009. *Introduction to Interval Analysis*. SIAM. https://doi.org/10.1137/1.9780898717716

[48] R. Motwani and P. Raghavan. 1995. *Randomized algorithms*. Cambridge University Press.

[49] Akihiko Nishimura, David B Dunson, and Jianfeng Lu. 2020. Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods. *Biometrika* 107, 2 (2020), 365–380.

[50] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. ACM, 672–681. https://doi.org/10.1145/2933575.2935317

[51] Gordon D. Plotkin. 1977. LCF Considered as a Programming Language. *Theor. Comput. Sci.* 5, 3 (1977), 223–255. https://doi.org/10.1016/0304-3975(77)90044-5

[52] M. O. Rabin. 1976. Probabilistic algorithms. In *Algorithms and complexity: new directions and results*. Academic Press, New York, 21–39.

[53] Tom Rainforth. 2017. *Automating Inference, Learning, and Design Using Probabilistic Programming*. Ph.D. Dissertation. University of Oxford.

[54] Reuven Y. Rubinstein and Dirk P. Kroese. 2017. *Simulation and the Monte Carlo Method* (3rd ed.). Wiley.

[55] Eugene S. Santos. 1969. Probabilistic Turing Machines and Computability. *Proc. Amer. Math. Soc.* 22, 3 (1969), 704–710. http://www.jstor.org/stable/2037463

[56] David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9286)*. Springer, 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

[57] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja R. Rudolph, Dawen Liang, and David M. Blei. 2016. Edward: A library for probabilistic modeling, inference, and criticism. *CoRR* abs/1610.09787 (2016). arXiv:1610.09787 http://arxiv.org/abs/1610.09787

[58] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 http://arxiv.org/abs/1809.10756

[59] Yuan Zhou, Bradley J. Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. 2019. LF-PPL: A Low-Level First Order Probabilistic Programming Language for Non-Differentiable Models. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan (Proceedings of Machine Learning Research, Vol. 89)*. PMLR, 148–157. http://proceedings.mlr.press/v89/zhou19b.html