Densities of Almost Surely Terminating Probabilistic Programs are Differentiable Almost Everywhere

Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner(✉)

Department of Computer Science, University of Oxford, Oxford, UK
{pui.mak,luke.ong,hugo.paquet,dominik.wagner}@cs.ox.ac.uk

Abstract. We study the differential properties of higher-order statistical probabilistic programs with recursion and conditioning. Our starting point is an open problem posed by Hongseok Yang: what class of statistical probabilistic programs have densities that are differentiable almost everywhere? To formalise the problem, we consider Statistical PCF (SPCF), an extension of call-by-value PCF with real numbers, and constructs for sampling and conditioning. We give SPCF a sampling-style operational semantics à la Borgström et al., and study the associated weight (commonly referred to as the density) function and value function on the set of possible execution traces.

Our main result is that almost surely terminating SPCF programs, generated from a set of primitive functions (e.g. the set of analytic functions) satisfying mild closure properties, have weight and value functions that are almost everywhere differentiable. We use a stochastic form of symbolic execution to reason about almost everywhere differentiability. A by-product of this work is that almost surely terminating *deterministic* (S)PCF programs with real parameters denote functions that are almost everywhere differentiable.

Our result is of practical interest, as almost everywhere differentiability of the density function is required to hold for the correctness of major gradient-based inference algorithms.

1 Introduction

Probabilistic programming refers to a set of tools and techniques for the systematic use of programming languages in Bayesian statistical modelling. Users of probabilistic programming — those wishing to make inferences or predictions — **(i)** encode their domain knowledge in program form; **(ii)** *condition* certain program variables based on observed data; and **(iii)** make a query. The resulting code is then passed to an *inference engine* which performs the necessary computation to answer the query, usually following a generic approximate Bayesian inference algorithm. (In some recent systems [5,14], users may also write their own inference code.) The Programming Language community has contributed to the field by developing formal methods for probabilistic programming languages (PPLs), seen as usual languages enriched with primitives for **(i)** sampling and

N. Yoshida (Ed.): ESOP 2021, LNCS 12648, pp. 432–461, 2021.
https://doi.org/10.1007/978-3-030-72019-3_16

(ii) conditioning. (The query **(iii)** can usually be encoded as the return value of the program.)

It is crucial to have access to reasoning principles in this context. The combination of these new primitives with the traditional constructs of programming languages leads to a variety of new computational phenomena, and a major concern is the *correctness of inference*: given a query, will the algorithm converge, in some appropriate sense, to a correct answer? In a *universal* PPL (i.e. one whose underlying language is Turing-complete), this is not obvious: the inference engine must account for a wide class of programs, going beyond the more well-behaved models found in many of the current statistical applications. Thus the design of inference algorithms, and the associated correctness proofs, are quite delicate. It is well-known, for instance, that in its original version the popular lightweight Metropolis-Hastings algorithm [53] contained a bug affecting the result of inference [20,25].

Fortunately, research in this area benefits from decades of work on the semantics of programs with random features, starting with pioneering work by Kozen [26] and Saheb-Djahromi [44]. Both operational and denotational models have recently been applied to the validation of inference algorithms: see e.g. [20,8] for the former and [45,10] for the latter. There are other approaches, e.g. using refined type systems [33].

Inference algorithms in probabilistic programming are often based on the concept of *program trace*, because the operational behaviour of a program is parametrised by the sequence of random numbers it draws along the way. Accordingly a probabilistic program has an associated *value function* which maps traces to output values. But the inference procedure relies on another function on traces, commonly called the *density*[1] of the program, which records a cumulative likelihood for the samples in a given trace. Approximating a normalised version of the density is the main challenge that inference algorithms aim to tackle. We will formalise these notions: in Sec. 3 we demonstrate how the value function and density of a program are defined in terms of its operational semantics.

Contributions. The main result of this paper is that both the density and value function are *differentiable almost everywhere* (that is, everywhere but on a set of measure zero), provided the program is *almost surely terminating* in a suitable sense. Our result holds for a universal language with recursion and higher-order functions. We emphasise that it follows immediately that *purely deterministic programs with real parameters* denote functions that are almost everywhere differentiable. This class of programs is important, because they can express machine learning models which rely on gradient descent [30].

This result is of practical interest, because many modern inference algorithms are "gradient-based": they exploit the derivative of the density function in order to optimise the approximation process. This includes the well-known methods of Hamiltonian Monte-Carlo [15,37] and stochastic variational inference [18,40,6,27]. But these techniques can only be applied when the derivative

[1] For some readers this terminology may be ambiguous; see Remark 1 for clarification.

exists "often enough", and thus, in the context of probabilistic programming, almost everywhere differentiability is often cited as a requirement for correctness [55,31]. The question of which probabilistic programs satisfy this property was selected by Hongseok Yang in his FSCD 2019 invited lecture [54] as one of three open problems in the field of semantics for probabilistic programs.

Points of non-differentiability exist largely because of *branching*, which typically arises in a program when the control flow reaches a conditional statement. Hence our work is a study of the connections between the traces of a probabilistic program and its branching structure. To achieve this we introduce *stochastic symbolic execution*, a form of operational semantics for probabilistic programs, designed to identify sets of traces corresponding to the same control-flow branch. Roughly, a reduction sequence in this semantics corresponds to a control flow branch, and the rules additionally provide for every branch a symbolic expression of the trace density, parametrised by the outcome of the random draws that the branch contains. We obtain our main result in conjunction with a careful analysis of the branching structure of almost surely terminating programs.

Outline. We devote Sec. 2 to a more detailed introduction to the problem of trace-based inference in probabilistic programming, and the issue of differentiability in this context. In Sec. 3, we present a trace-based operational semantics to Statistical PCF, a prototypical higher-order functional language previously studied in the literature. This is followed by a discussion of differentiability and almost sure termination of programs (Sec. 4). In Sec. 5 we define the "symbolic" operational semantics required for the proof of our main result, which we present in Sec. 6. We discuss related work and further directions in Sec. 7.

For the extended version of the paper refer to [34].

2 Probabilistic Programming and Trace-Based Inference

In this section we give a short introduction to probabilistic programs and the densities they denote, and we motivate the need for gradient-based inference methods. Our account relies on classical notions from measure theory, so we start with a short recap.

2.1 Measures and Densities

A ***measurable space*** is a pair (X, Σ_X) consisting of a set together with a ***σ-algebra*** of subsets, i.e. $\Sigma_X \subseteq \mathcal{P}(X)$ contains $\emptyset$ and is closed under complements and countable unions and intersections. Elements of Σ_X are called ***measurable sets***. A ***measure*** on (X, Σ_X) is a function $\mu : \Sigma_X \to [0, \infty]$ satisfying $\mu(\emptyset) = 0$, and $\mu(\bigcup_{i \in I} U_i) = \sum_{i \in I} \mu(U_i)$ for every countable family $\{U_i\}_{i \in I}$ of pairwise disjoint measurable subsets. A (possibly partial) function $X \rightharpoonup Y$ is ***measurable*** if for every $U \in \Sigma_Y$ we have $f^{-1}(U) \in \Sigma_X$.

The space $\mathbb{R}$ of real numbers is an important example. The (Borel) σ-algebra $\Sigma_{\mathbb{R}}$ is the smallest one containing all intervals $[a, b)$, and the ***Lebesgue measure***

Leb is the unique measure on $(\mathbb{R}, \Sigma_{\mathbb{R}})$ satisfying $\mathrm{Leb}([a,b)) = b - a$. For measurable spaces (X, Σ_X) and (Y, Σ_Y), the ***product σ-algebra*** $\Sigma_{X\times Y}$ is the smallest one containing all $U \times V$, where $U \in \Sigma_X$ and $V \in \Sigma_Y$. So in particular we get for each $n \in \mathbb{N}$ a space $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$, and additionally there is a unique measure Leb_n on $\mathbb{R}^n$ satisfying $\mathrm{Leb}_n(\prod_i U_i) = \prod_i \mathrm{Leb}(U_i)$.

When a function $f : X \to \mathbb{R}$ is measurable and non-negative and μ is a measure on X, for each $U \in \Sigma_X$ we can define the ***integral*** $\int_U (\mathrm{d}\mu) f \in [0, \infty]$. Common families of probability distributions on the reals (Uniform, Normal, etc.) are examples of measures on $(\mathbb{R}, \Sigma_{\mathbb{R}})$. Most often these are defined in terms of *probability density functions* with respect to the Lebesgue measure, meaning that for each μ_D there is a measurable function $\mathrm{pdf}_D : \mathbb{R} \to \mathbb{R}_{\geq 0}$ which determines it: $\mu_D(U) = \int_U (\mathrm{d}\,\mathrm{Leb})\ \mathrm{pdf}_D$. As we will see, density functions such as pdf_D have a central place in Bayesian inference.

Formally, if μ is a measure on a measurable space X, a ***density*** for μ with respect to another measure ν on X (most often ν is the Lebesgue measure) is a measurable function $f : X \to \mathbb{R}$ such that $\mu(U) = \int_U (\mathrm{d}\nu) f$ for every $U \in \Sigma_X$. In the context of the present work, an *inference algorithm* can be understood as a method for approximating a distribution of which we only know the density up to a normalising constant. In other words, if the algorithm is fed a (measurable) function $g : X \to \mathbb{R}$, it should produce samples approximating the probability measure $U \mapsto \frac{\int_U (\mathrm{d}\nu) g}{\int_X (\mathrm{d}\nu) g}$ on X.

We will make use of some basic notions from topology: given a topological space X and an set $A \subseteq X$, the ***interior*** of A is the largest open set $\mathring{A}$ contained in A. Dually the ***closure*** of A is the smallest closed set $\overline{A}$ containing A, and the ***boundary*** of A is defined as $\partial A \coloneqq \overline{A} \setminus \mathring{A}$. Note that for all $U \subseteq \mathbb{R}^n$, all of $\mathring{U}$, $\overline{U}$ and ∂U are measurable (in $\Sigma_{\mathbb{R}^n}$).

2.2 Probabilistic Programming: a (Running) Example

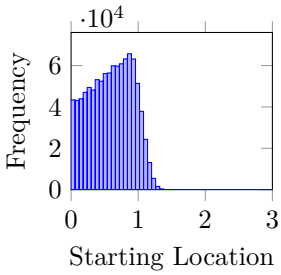
Our running example is based on a random walk in $\mathbb{R}_{\geq 0}$.

The story is as follows: a pedestrian has gone on a walk on a certain semi-infinite street (i.e. extending infinitely on one side), where she may periodically change directions. Upon reaching the end of the street she has forgotten her starting point, only remembering that she started no more than 3km away. Thanks to an odometer, she knows the total distance she has walked is 1.1km, although there is a small margin of error. Her starting point can be inferred using probabilistic programming, via the program in Fig. 1a.

The function walk in Fig. 1a is a recursive simulation of the random walk: note that in this model a new direction is sampled after at most 1km. Once the pedestrian has travelled past 0 the function returns the total distance travelled. The rest of the program first specifies a *prior distribution* for the starting point, representing the pedestrian's belief — uniform distribution on $[0, 3]$ — before observing the distance measured by the odometer. After drawing a value for start the program simulates a random walk, and the execution is weighted (via score) according to how close distance is to the observed value of 1.1. The return

```
(*returns total distance travelled*)
let rec walk start =
  if (start <= 0) then
    0
  else
    (*each leg < 1km*)
    let step = Uniform(0, 1) in
    if (flip ()) then
      (*go towards +infty*)
      step + walk (start+step)
    else
      (*go towards 0*)
      step + walk (start-step)
in
(*prior*)
let start = Uniform(0, 3) in
let distance = walk start in
(*likelihood*)
score ((pdfN distance 0.1) 1.1);
(*query*)
start
```

(a) Running example in pseudo-code.



(b) Resulting histogram.

Fig. 1: Inferring the starting point of a random walk on $\mathbb{R}_{\geq 0}$, in a PPL.

value is our query: it indicates that we are interested in the *posterior* distribution on the starting point.

The histogram in Fig. 1b is obtained by sampling repeatedly from the posterior of a Python model of our running example. It shows the mode of the pedestrian's starting point to be around the 0.8km mark.

To approximate the posterior, inference engines for probabilistic programs often proceed indirectly and operate on the space of *program traces*, rather than on the space of possible return values. By *trace*, we mean the sequence of samples drawn in the course of a particular run, one for each random primitive encountered. Because each random primitive (qua probability distribution) in the language comes with a density, given a particular trace we can compute a coefficient as the appropriate product. We can then multiply this coefficient by all scores encountered in the execution, and this yields a (*weight*) function, mapping traces to the non-negative reals, over which the chosen inference algorithm may operate. This indirect approach is more practical, and enough to answer the query, since every trace unambiguously induces a return value.

Remark 1. In much of the probabilistic programming literature (e.g. [31,55,54], including this paper), the above-mentioned weight function on traces is called the *density* of the probabilistic program. This may be confusing: as we have seen, a probabilistic program induces a posterior probability distribution on return

values, and it is natural to ask whether this distribution admits a probability density function (Radon-Nikodym derivative) w.r.t. some base measure. This problem is of current interest [2,3,21] but unrelated to the present work.

2.3 Gradient-Based Approximate Inference

Some of the most influential and practically important inference algorithms make use of the gradient of the density functions they operate on, when these are differentiable. Generally the use of gradient-based techniques allow for much greater efficiency in inference.

A popular example is the Markov Chain Monte Carlo algorithm known as Hamiltonian Monte Carlo (HMC) [15,37]. Given a density function $g : X \to \mathbb{R}$, HMC samples are obtained as the states of a Markov chain by (approximately) simulating Hamilton's equations via an integrator that uses the gradient $\nabla_x g(x)$. Another important example is (stochastic) variational inference [18,40,6,27], which transforms the posterior inference problem to an optimisation problem. This method takes two inputs: the posterior density function of interest $g : X \to \mathbb{R}$, and a function $h : \Theta \times X \to \mathbb{R}$; typically, the latter function is a member of an expressive and mathematically well-behaved family of densities that are parameterised in Θ. The idea is to use stochastic gradient descent to find the parameter $\theta \in \Theta$ that minimises the "distance" (typically the Kullback–Leibler divergence) between $h(\theta, -)$ and g, relying on a suitable estimate of the gradient of the objective function. When g is the density of a probabilistic program (the *model*), h can be specified as the density of a second program (the *guide*) whose traces have additional θ-parameters. The gradient of the objective function is then estimated in one approach (score function [41]) by computing the gradient $\nabla_\theta h(\theta, x)$, and in another (reparameterised gradient [24,42,49]) by computing the gradient $\nabla_x g(x)$.

In probabilistic programming, the above inference methods must be adapted to deal with the fact that in a universal PPL, the set of random primitives encountered can vary between executions, and traces can have arbitrary and unbounded dimension; moreover, the density function of a probabilistic program is generally not (everywhere) differentiable. Crucially these adapted algorithms are only valid when the input densities are *almost everywhere* differentiable [55,38,32]; this is the subject of this paper.

Our main result (Thm. 3) states that the weight function and value function of almost surely terminating SPCF programs are almost everywhere differentiable. This applies to our running example: the program in Fig. 1a (expressible in SPCF using primitive functions that satisfy Assumption 1 – see Ex. 1) is almost surely terminating.

3 Sampling Semantics for Statistical PCF

In this section, we present a simply-typed statistical probabilistic programming language with recursion and its operational semantics.

$$\sigma, \tau ::= \mathsf{R} \mid \sigma \Rightarrow \tau$$
$$M, N, L ::= y \mid \underline{r} \mid \underline{f}(M_1, \ldots, M_\ell) \mid \lambda y.\, M \mid M\, N \mid \mathsf{Y} M \mid \mathsf{if}\big(L \leq 0, M, N\big)$$
$$\mid \mathsf{sample} \mid \mathsf{score}(M)$$

$$\frac{}{\Gamma \vdash \mathsf{sample} : \mathsf{R}} \qquad \frac{\Gamma \vdash M : \mathsf{R}}{\Gamma \vdash \mathsf{score}(M) : \mathsf{R}} \qquad \frac{\Gamma \vdash M : (\sigma \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau)}{\Gamma \vdash \mathsf{Y} M : \sigma \Rightarrow \tau}$$

Fig. 2: Syntax of SPCF, where $r \in \mathbb{R}$, x, y are variables, and $f : \mathbb{R}^n \rightharpoonup \mathbb{R}$ ranges over a set $\mathcal{F}$ of partial, measurable ***primitive functions*** (see Sec. 4.2).

3.1 Statistical PCF

Statistical PCF (SPCF) is higher-order probabilistic programming with recursion in purified form. The terms and part of the (standard) typing system of SPCF are presented in Fig. 2 [2]. In the rest of the paper we write $\boldsymbol{x}$ to represent a sequence of variables $x_1, \ldots, x_n$, Λ for the set of SPCF terms, and Λ^0 for the set of closed SPCF terms. In the interest of readability, we sometimes use pseudo code (e.g. Fig. 1a) in the style of Core ML to express SPCF terms.

SPCF is a statistical probabilistic version of call-by-value PCF [46,47] with reals as the ground type. The probabilistic constructs of SPCF are relatively standard (see for example [48]): the sampling construct sample draws from $\mathcal{U}(0,1)$, the standard uniform distribution with end points 0 and 1; the scoring construct $\mathsf{score}(M)$ enables conditioning on observed data by multiplying the weight of the current execution with the (non-negative) real number denoted by M. Sampling from other real-valued distributions can be obtained from $\mathcal{U}(0,1)$ by applying the inverse of the distribution's cumulative distribution function.

Our SPCF is an (inconsequential) variant of CBV SPCF [51] and a (CBV) extension of PPCF [16] with scoring; it may be viewed as a simply-typed version of the untyped probabilistic languages of [8,13,52].

Example 1 (Running Example Ped*).* We express in SPCF the example in Fig. 1a.

$$\mathsf{Ped} \equiv \begin{pmatrix} \mathsf{let}\ x = \mathsf{sample} \cdot \underline{3}\ \mathsf{in} \\ \mathsf{let}\ d = \mathsf{walk}\, x\ \mathsf{in} \\ \mathsf{let}\ w = \mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(d))\ \mathsf{in}\ x \end{pmatrix} \quad \text{where}$$

$$\mathsf{walk} \equiv \mathsf{Y} \begin{pmatrix} \lambda f x.\ \mathsf{if}\ x \leq \underline{0}\ \mathsf{then}\ \underline{0} \\ \qquad \mathsf{else} \begin{pmatrix} \mathsf{let}\ s = \mathsf{sample}\ \mathsf{in} \\ \mathsf{if}\big((\mathsf{sample} \leq \underline{0.5}), \big(s + f(x+s)\big), \big(s + f(x-s)\big)\big) \end{pmatrix} \end{pmatrix}$$

The let construct, $\mathsf{let}\ x = N\ \mathsf{in}\ M$, is syntactic sugar for the term $(\lambda x.M)\, N$; and $\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}$, the density function of the normal distribution with mean 1.1 and variance 0.1, is a primitive function. To enhance readability we use infix notation and omit the underline for standard functions such as addition.

[2] In Fig. 2 and in other figures, we highlight the elements that are new or otherwise noteworthy.

3.2 Operational Semantics

The execution of a probabilistic program generates a *trace*: a sequence containing the values sampled during a run. Our operational semantics captures this dynamic perspective. This is closely related to the treatment in [8] which, following [26], views a probabilistic program as a deterministic program parametrized by the sequence of random draws made during the evaluation.

Traces. Recall that in our language, sample produces a random value in the open unit interval; accordingly a ***trace*** is a finite sequence of elements of $(0,1)$. We define a ***measure space*** $\mathbb{S}$ ***of traces*** to be the set $\bigcup_{n\in\mathbb{N}}(0,1)^n$, equipped with the standard disjoint union σ-algebra, and the sum of the respective (higher-dimensional) Lebesgue measures. Formally, writing $\mathbb{S}_n \coloneqq (0,1)^n$, we define:

$$\mathbb{S} \coloneqq \left(\bigcup_{n\in\mathbb{N}} \mathbb{S}_n, \left\{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \Sigma_{\mathbb{S}_n}\right\}, \mu_{\mathbb{S}}\right) \text{ and } \mu_{\mathbb{S}}\left(\bigcup_{n\in\mathbb{N}} U_n\right) \coloneqq \sum_{n\in\mathbb{N}} \mathrm{Leb}_n(U_n).$$

Henceforth we write traces as lists, such as $[0.5, 0.999, 0.12]$; the empty trace as $[]$; and the concatenation of traces $\boldsymbol{s}, \boldsymbol{s}' \in \mathbb{S}$ as $\boldsymbol{s} +\!\!+ \boldsymbol{s}'$.

More generally, to account for open terms, we define, for each $m \in \mathbb{N}$, the measure space

$$\mathbb{R}^m \times \mathbb{S} \coloneqq \left(\bigcup_{n\in\mathbb{N}} \mathbb{R}^m \times \mathbb{S}_n, \left\{\bigcup_{n\in\mathbb{N}} V_n \mid V_n \in \Sigma_{\mathbb{R}^m\times\mathbb{S}_n}\right\}, \mu_{\mathbb{R}^m\times\mathbb{S}}\right)$$

where $\mu_{\mathbb{R}^m\times\mathbb{S}}\left(\bigcup_{n\in\mathbb{N}} V_n\right) \coloneqq \sum_{n\in\mathbb{N}} \mathrm{Leb}_{m+n}(V_n)$. To avoid clutter, we will elide the subscript from $\mu_{\mathbb{R}^m\times\mathbb{S}}$ whenever it is clear from the context.

Small-Step Reduction. Next, we define the ***values*** (typically denoted V), ***redexes*** (typically R) and ***evaluation contexts*** (typically E):

$$\begin{aligned}
V &::= \underline{r} \mid \lambda y.M \\
R &::= (\lambda y.\, M)\, V \mid \underline{f}(\underline{r_1}, \ldots, \underline{r_\ell}) \mid \mathsf{Y}(\lambda y.\, M) \mid \mathsf{if}\big(\underline{r} \le 0, M, N\big) \mid \mathsf{sample} \mid \mathsf{score}(\underline{r}) \\
E &::= [\,] \mid E\, M \mid (\lambda y.M)\, E \mid \underline{f}(\underline{r_1}, \ldots, \underline{r_{i-1}}, E, M_{i+1}, \ldots, M_\ell) \mid \mathsf{Y}E \\
&\quad \mid \mathsf{if}\big(E \le 0, M, N\big) \mid \mathsf{score}(E)
\end{aligned}$$

We write Λ_v for the set of SPCF values, and Λ_v^0 for the set of closed SPCF values.

It is easy to see that every closed SPCF term M is either a value, or there exists a unique pair of context E and redex R such that $M \equiv E[R]$.

We now present the operational semantics of SPCF as a rewrite system of ***configurations***, which are triples of the form $\langle M, w, \boldsymbol{s}\rangle$ where M is a closed SPCF term, $w \in \mathbb{R}_{\ge 0}$ is a ***weight***, and $\boldsymbol{s} \in \mathbb{S}$ a trace. (We will sometimes refer to

Redex Contractions:

$$\begin{aligned}
\langle (\lambda y.\, M)\, V, w, \boldsymbol{s}\rangle &\to \langle M[V/y], w, \boldsymbol{s}\rangle \\
\big\langle \underline{f}(\underline{r_1}, \ldots, \underline{r_\ell}), w, \boldsymbol{s}\big\rangle &\to \Big\langle \underline{f(r_1, \ldots, r_\ell)}, w, \boldsymbol{s}\Big\rangle && (\text{if } (r_1, \ldots, r_\ell) \in \mathsf{dom}(f)) \\
\big\langle \underline{f}(\underline{r_1}, \ldots, \underline{r_\ell}), w, \boldsymbol{s}\big\rangle &\to \mathsf{fail} && (\text{if } (r_1, \ldots, r_\ell) \notin \mathsf{dom}(f)) \\
\langle \mathsf{Y}(\lambda y.M), w, \boldsymbol{s}\rangle &\to \langle \lambda z.M[\mathsf{Y}(\lambda y.M)/y]\, z, w, \boldsymbol{s}\rangle && (\text{for fresh variable } z) \\
\big\langle \mathsf{if}\big(\underline{r} \le 0, M, N\big), w, \boldsymbol{s}\big\rangle &\to \langle M, w, \boldsymbol{s}\rangle && (\text{if } r \le 0) \\
\big\langle \mathsf{if}\big(\underline{r} \le 0, M, N\big), w, \boldsymbol{s}\big\rangle &\to \langle N, w, \boldsymbol{s}\rangle && (\text{if } r > 0) \\
\langle \mathsf{sample}, w, \boldsymbol{s}\rangle &\to \langle \underline{r}, w, \boldsymbol{s} +\!\!+ [r]\rangle && (\text{for some } r \in (0,1)) \\
\langle \mathsf{score}(\underline{r}), w, \boldsymbol{s}\rangle &\to \langle \underline{r}, r \cdot w, \boldsymbol{s}\rangle && (\text{if } r \ge 0) \\
\langle \mathsf{score}(\underline{r}), w, \boldsymbol{s}\rangle &\to \mathsf{fail} && (\text{if } r < 0)
\end{aligned}$$

Evaluation Contexts:

$$\frac{\langle R, w, \boldsymbol{s}\rangle \to \langle R', w', \boldsymbol{s}'\rangle}{\langle E[R], w, \boldsymbol{s}\rangle \to \langle E[R'], w', \boldsymbol{s}'\rangle} \qquad \frac{\langle R, w, \boldsymbol{s}\rangle \to \mathsf{fail}}{\langle E[R], w, \boldsymbol{s}\rangle \to \mathsf{fail}}$$

Fig. 3: Operational small-step semantics of SPCF

these as the *concrete* configurations, in contrast with the *abstract* configurations of our symbolic operational semantics, see Sec. 5.2.)

The small-step reduction relation $\to$ is defined in Fig. 3. In the rule for sample, a random value $r \in (0,1)$ is generated and recorded in the trace, while the weight remains unchanged: in a uniform distribution on $(0,1)$ each value is drawn with likelihood 1. In the rule for $\mathsf{score}(\underline{r})$, the current weight is multiplied by non-negative $r \in \mathbb{R}$: typically this reflects the likelihood of the current execution given some observed data. Similarly to [8] we reduce terms which cannot be reduced in a reasonable way (i.e. scoring with negative constants or evaluating functions outside their domain) to fail.

Example 2. We present a possible reduction sequence for the program in Ex. 1:

$$\begin{aligned}
\langle \mathsf{Ped}, 1, []\rangle &\to^* \left\langle \begin{pmatrix} \mathsf{let}\ x = \underline{0.2} \cdot \underline{3}\ \mathsf{in} \\ \mathsf{let}\ d = \mathsf{walk}\, x\ \mathsf{in} \\ \mathsf{let}\ w = \mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(d))\ \mathsf{in}\ x \end{pmatrix}, 1, [0.2] \right\rangle \\
&\to^* \left\langle \begin{pmatrix} \mathsf{let}\ d = \mathsf{walk}\, \underline{0.6}\ \mathsf{in} \\ \mathsf{let}\ w = \mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(d))\ \mathsf{in}\ \underline{0.6} \end{pmatrix}, 1, [0.2] \right\rangle \\
&\to^* \left\langle \mathsf{let}\ w = \mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(\underline{0.9}))\ \mathsf{in}\ \underline{0.6}, 1, [0.2, 0.9, 0.7] \right\rangle \qquad (\star) \\
&\to^* \langle \mathsf{let}\ w = \mathsf{score}(\underline{0.54})\ \mathsf{in}\ \underline{0.6}, 1, [0.2, 0.9, 0.7]\rangle \\
&\to^* \langle \underline{0.6}, 0.54, [0.2, 0.9, 0.7]\rangle
\end{aligned}$$

In this execution, the initial sample yields $\underline{0.2}$, which is appended to the trace. At step $(\star)$, we assume given a reduction sequence $\langle \mathsf{walk}\, \underline{0.6}, 1, [0.2]\rangle \to^*$

$\langle \underline{0.9}, 1, [0.2, 0.9, 0.7] \rangle$; this means that in the call to walk, 0.9 was sampled as the the step size and 0.7 as the direction factor; this makes the new location -0.3, which is negative, so the return value is 0.9. In the final step, we perform *conditioning* using the likelihood of observing 0.9 given the data 1.1: the $\mathsf{score}()$ expression updates the current weight using the the density of 0.9 in the normal distribution with parameters $(1.1, 0.1)$.

Value and Weight Functions. Using the relation $\to$, we now aim to reason more globally about probabilistic programs in terms of the traces they produce. Let M be an SPCF term with free variables amongst $x_1, \ldots, x_m$ of type R. Its ***value function*** $\mathsf{value}_M : \mathbb{R}^m \times \mathbb{S} \to \Lambda_v^0 \cup \{\bot\}$ returns, given values for each free variable and a trace, the output value of the program, if the program terminates in a value. The ***weight function*** $\mathsf{weight}_M : \mathbb{R}^m \times \mathbb{S} \to \mathbb{R}_{\geq 0}$ returns the final weight of the corresponding execution. Formally:

$$\mathsf{value}_M(\boldsymbol{r}, \boldsymbol{s}) \coloneqq \begin{cases} V & \text{if } \langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, [] \rangle \to^* \langle V, w, \boldsymbol{s} \rangle \\ \bot & \text{otherwise} \end{cases}$$

$$\mathsf{weight}_M(\boldsymbol{r}, \boldsymbol{s}) \coloneqq \begin{cases} w & \text{if } \langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, [] \rangle \to^* \langle V, w, \boldsymbol{s} \rangle \\ 0 & \text{otherwise} \end{cases}$$

For closed SPCF terms M we just write $\mathsf{weight}_M(\boldsymbol{s})$ for $\mathsf{weight}_M([], \boldsymbol{s})$ (similarly for value_M), and it follows already from [8, Lemma 9] that the functions value_M and weight_M are measurable (see also Sec. 4.1).

Finally, every closed SPCF term M has an associated ***value measure***

$$[\![M]\!] : \Sigma_{\Lambda_v^0} \to \mathbb{R}_{\geq 0}$$

defined by $[\![M]\!](U) \coloneqq \int_{\mathsf{value}_M{}^{-1}(U)} \mathrm{d}\mu_{\mathbb{S}}\, \mathsf{weight}_M$. This corresponds to the denotational semantics of SPCF in the ω-quasi-Borel space model via computational adequacy [51].

Returning to Remark 1, what are the connections, if any, between the two types of density of a program? To distinguish them, let's refer to the weight function of the program, weight_M, as its *trace density*, and the Radon-Nikodyn derivative of the program's value-measure, $\frac{d[\![M]\!]}{d\nu}$ where ν is the reference measure of the measurable space $\Sigma_{\Lambda_v^0}$, as the *output density*. Observe that, for any measurable function $f : \Lambda_v^0 \to [0, \infty]$, $\int_{\Lambda_v^0} d\,[\![M]\!]\, f = \int_{\mathsf{value}_M^{-1}(\Lambda_v^0)} d\mu_{\mathbb{S}}\, \mathsf{weight}_M \cdot (f \circ \mathsf{value}_M) = \int_{\mathbb{S}} d\mu_{\mathbb{S}}\, \mathsf{weight}_M \cdot (f \circ \mathsf{value}_M)$ (because if $\boldsymbol{s} \notin \mathsf{value}_M^{-1}(\Sigma_{\Lambda_v^0})$ then $\mathsf{weight}_M(\boldsymbol{s}) = 0$). It follows that we can express any expectation w.r.t. the output density $\frac{d[\![M]\!]}{d\nu}$ as an expectation w.r.t. the trace density weight_M. If our aim is, instead, to generate samples from $\frac{d[\![M]\!]}{d\nu}$ then we can simply generate samples from weight_M, and deterministically convert each sample to the space $(\Lambda_v^0, \Sigma_{\Lambda_v^0})$ via the value function value_M. In other words, if our intended output is just a sequence of samples, then our inference engine does not need to concern itself with the consequences of change of variables.

4 Differentiability of the Weight and Value Functions

To reason about the differential properties of these functions we place ourselves in a setting in which differentiation makes sense. We start with some preliminaries.

4.1 Background on Differentiable Functions

Basic real analysis gives a standard notion of differentiability at a point $\boldsymbol{x} \in \mathbb{R}^n$ for functions between Euclidean spaces $\mathbb{R}^n \to \mathbb{R}^m$. In this context a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is ***smooth*** on an open $U \subseteq \mathbb{R}^n$ if it has derivatives of all orders at every point of U. The theory of *differential geometry* (see e.g. the textbooks [50,29,28]) abstracts away from Euclidean spaces to *smooth manifolds*. We recall the formal definitions.

A topological space $\mathcal{M}$ is ***locally Euclidean at a point*** $x \in \mathcal{M}$ if x has a neighbourhood U such that there is a homeomorphism ϕ from U onto an open subset of $\mathbb{R}^n$, for some n. The pair $(U, \phi : U \to \mathbb{R}^n)$ is called a ***chart*** (of dimension n). We say $\mathcal{M}$ is ***locally Euclidean*** if it is locally Euclidean at every point. A ***manifold*** $\mathcal{M}$ is a Hausdorff, second countable, locally Euclidean space.

Two charts, $(U, \phi : U \to \mathbb{R}^n)$ and $(V, \psi : V \to \mathbb{R}^m)$, are ***compatible*** if the function $\psi \circ \phi^{-1} : \phi(U \cap V) \to \psi(U \cap V)$ is smooth, with a smooth inverse. An ***atlas*** on $\mathcal{M}$ is a family $\{(U_\alpha, \phi_\alpha)\}$ of pairwise compatible charts that cover $\mathcal{M}$. A ***smooth manifold*** is a manifold equipped with an atlas.

It follows from the topological invariance of dimension that charts that cover a part of the same connected component have the same dimension. We emphasise that, although this might be considered slightly unusual, distinct connected components need not have the same dimension. This is important for our purposes: $\mathbb{S}$ is easily seen to be a smooth manifold since each connected component $\mathbb{S}_i$ is diffeomorphic to $\mathbb{R}^i$. It is also straightforward to endow the set Λ of SPCF terms with a (smooth) manifold structure. Following [8] we view Λ as $\bigcup_{m \in \mathbb{N}} \left(\mathsf{SK}_m \times \mathbb{R}^m\right)$, where SK_m is the set of SPCF terms with exactly m place-holders (a.k.a. *skeleton terms*) for numerals. Thus identified, we give Λ the countable disjoint union topology of the product topology of the discrete topology on SK_m and the standard topology on $\mathbb{R}^m$. Note that the connected components of Λ have the form $\{M\} \times \mathbb{R}^m$, with M ranging over SK_m, and m over $\mathbb{N}$. So in particular, the subspace $\Lambda_v \subseteq \Lambda$ of values inherits the manifold structure. We fix the Borel algebra of this topology to be the σ-algebra on Λ.

Given manifolds $(\mathcal{M}, \{U_\alpha, \phi_\alpha\})$ and $(\mathcal{M}', \{V_\beta, \psi_\beta\})$, a function $f : \mathcal{M} \to \mathcal{M}'$ is ***differentiable*** at a point $x \in \mathcal{M}$ if there are charts (U_α, ϕ_α) about x and (V_β, ψ_β) about $f(x)$ such that the composite $\psi_\beta \circ f \circ \phi_\alpha^{-1}$ restricted to the open subset $\phi_\alpha(f^{-1}(V_\beta) \cap U_\alpha)$ is differentiable at $\phi_\alpha(x)$.

The definitions above are useful because they allow for a uniform presentation. But it is helpful to unpack the definition of differentiability in a few instances, and we see that they boil down to the standard sense in real analysis. Take an SPCF term M with free variables amongst $x_1, \ldots, x_m$ (all of type $\mathbb{R}$), and $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}_n$.

- The function $\mathsf{weight}_M : \mathbb{R}^m \times \mathbb{S} \to \mathbb{R}_{\geq 0}$ is differentiable at $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}_n$ just if its restriction $\mathsf{weight}_M|_{\mathbb{R}^m \times \mathbb{S}_n} : \mathbb{R}^m \times \mathbb{S}_n \to \mathbb{R}_{\geq 0}$ is differentiable at $(\boldsymbol{r}, \boldsymbol{s})$.
- In case M is of type R, $\mathsf{value}_M : \mathbb{R}^m \times \mathbb{S} \to \Lambda_v^0 \cup \{\bot\}$ is in essence a partial function $\mathbb{R}^m \times \mathbb{S} \rightharpoonup \mathbb{R}$. Precisely value_M is differentiable at $(\boldsymbol{r}, \boldsymbol{s})$ just if for some open neighbourhood $U \subseteq \mathbb{R}^m \times \mathbb{S}_n$ of $(\boldsymbol{r}, \boldsymbol{s})$:
  1. $\mathsf{value}_M(\boldsymbol{r'}, \boldsymbol{s'}) = \bot$ for all $(\boldsymbol{r'}, \boldsymbol{s'}) \in U$; or
  2. $\mathsf{value}_M(\boldsymbol{r'}, \boldsymbol{s'}) \neq \bot$ for all $(\boldsymbol{r'}, \boldsymbol{s'}) \in U$, and $\mathsf{value}'_M : U \to \mathbb{R}$ is differentiable at $(\boldsymbol{r}, \boldsymbol{s})$, where we define $\mathsf{value}'_M(\boldsymbol{r'}, \boldsymbol{s'}) := r''$ whenever $\mathsf{value}_M(\boldsymbol{r'}, \boldsymbol{s'}) = \underline{r''}$.

4.2 Why Almost Everywhere Differentiability Can Fail

Conditional statements break differentiability. This is easy to see with an example: the weight function of the term

$$\mathsf{if}\big(\mathsf{sample} \leq \mathsf{sample}, \mathsf{score}(\underline{1}), \mathsf{score}(\underline{0})\big)$$

is exactly the characteristic function of $\{[s_1, s_2] \in \mathbb{S} \mid s_1 \leq s_2\}$, which is not differentiable on the diagonal $\{[s, s] \in \mathbb{S}_2 \mid s \in (0, 1)\}$.

This function is however differentiable *almost everywhere*: the diagonal is an uncountable set but has Leb_2 measure zero in the space $\mathbb{S}_2$. Unfortunately, this is not true in general. Without sufficient restrictions, conditional statements also break almost everywhere differentiability. This can happen for two reasons.

Problem 1: Pathological Primitive Functions. Recall that our definition of SPCF is parametrised by a set $\mathcal{F}$ of primitive functions. It is tempting in this context to take $\mathcal{F}$ to be the set of all differentiable functions, but this is too general, as we show now. Consider that for every $f : \mathbb{R} \to \mathbb{R}$ the term

$$\mathsf{if}\big(\underline{f}(\mathsf{sample}) \leq 0, \mathsf{score}(\underline{1}), \mathsf{score}(\underline{0})\big)$$

has weight function the characteristic function of $\{[s_1] \in \mathbb{S} \mid f(s_1) \leq 0\}$. This function is non-differentiable at every $s_1 \in \mathbb{S}_1 \cap \partial f^{-1}(-\infty, 0]$: in every neighbourhood of s_1 there are s_1' and s_1'' such that $f(s_1') \leq 0$ and $f(s_1'') > 0$. One can construct a differentiable f for which this is *not* a measure zero set. (For example, there exists a non-negative function f which is zero exactly on a *fat* Cantor set, i.e., a Cantor-like set with strictly positive measure. See [43, Ex. 5.21].)

Problem 2: Non-Terminating Runs. Our language has recursion, so we can construct a term which samples a random number, halts if this number is in $\mathbb{Q} \cap [0, 1]$, and diverges otherwise. In pseudo-code:

```
let rec enumQ p q r =
  if (r = p/q) then (score 1) else
    if (r < p/q) then
```

```
        enumQ p (q+1) r
      else
        enumQ (p+1) q r
in enumQ 0 1 sample
```

The induced weight function is the characteristic function of $\{[s_1] \in \mathbb{S} \mid s_1 \in \mathbb{Q}\}$; the set of points at which this function is non-differentiable is $\mathbb{S}_1$, which has measure 1.

We proceed to overcome Problem 1 by making appropriate assumptions on the set of primitives. We will then address Problem 2 by focusing on *almost surely terminating* programs.

4.3 Admissible Primitive Functions

One contribution of this work is to identify sufficient conditions for $\mathcal{F}$. We will show in Sec. 6 that our main result holds provided:

Assumption 1 (Admissible Primitive Functions)**.** *$\mathcal{F}$ is a set of partial, measurable functions $\mathbb{R}^\ell \rightharpoonup \mathbb{R}$ including all constant and projection functions which satisfies*

1. *if $f : \mathbb{R}^\ell \rightharpoonup \mathbb{R}$ and $g_i : \mathbb{R}^m \rightharpoonup \mathbb{R}$ are elements of $\mathcal{F}$ for $i = 1, \ldots, \ell$, then $f \circ \langle g_i \rangle_{i=1}^{\ell} : \mathbb{R}^m \rightharpoonup \mathbb{R}$ is in $\mathcal{F}$*
2. *if $(f : \mathbb{R}^\ell \rightharpoonup \mathbb{R}) \in \mathcal{F}$, then f is differentiable in the interior of $\mathsf{dom}(f)$*
3. *if $(f : \mathbb{R}^\ell \rightharpoonup \mathbb{R}) \in \mathcal{F}$, then $\mathrm{Leb}_\ell(\partial f^{-1}[0, \infty)) = 0$.*

Example 3. The following sets of primitive operations satisfy the above sufficient conditions. (See [34] for a proof.)

1. The set $\mathcal{F}_1$ of analytic functions with co-domain $\mathbb{R}$. Recall that a function $f : \mathbb{R}^\ell \to \mathbb{R}^n$ is *analytic* if it is infinitely differentiable and its multivariate Taylor expansion at every point $x_0 \in \mathbb{R}^\ell$ converges pointwise to f in a neighbourhood of x_0.
2. The set $\mathcal{F}_2$ of (partial) functions $f : \mathbb{R}^\ell \rightharpoonup \mathbb{R}$ such that $\mathsf{dom}(f)$ is open[3], and f is differentiable everywhere in $\mathsf{dom}(f)$, and $f^{-1}(I)$ is a finite union of (possibly unbounded) rectangles[4] for (possibly unbounded) intervals I.

Note that all primitive functions mentioned in our examples (and in particular the density of the normal distribution) are included in both $\mathcal{F}_1$ and $\mathcal{F}_2$.

It is worth noting that both $\mathcal{F}_1$ and $\mathcal{F}_2$ satisfy the following stronger (than Assumption 1.3) property: $\mathrm{Leb}_n(\partial f^{-1} I) = 0$ for every interval I, for every primitive function f.

[3] This requirement is crucial, and cannot be relaxed.
[4] i.e. a finite union of $I_1 \times \cdots \times I_\ell$ for (possibly unbounded) intervals I_i

4.4 Almost Sure Termination

To rule out the contrived counterexamples which diverge we restrict attention to *almost surely terminating* SPCF terms. Intuitively, a program M (closed term of ground type) is almost surely terminating if the probability that a run of M terminates is 1.

Take an SPCF term M with variables amongst $x_1, \ldots, x_m$ (all of type $\mathbb{R}$), and set

$$\mathbb{T}_{M,\mathsf{term}} := \left\{ (\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S} \mid \exists V, w \,.\, \langle M[\boldsymbol{r}/\boldsymbol{x}], 1, [] \rangle \to^* \langle V, w, \boldsymbol{s} \rangle \right\}. \tag{1}$$

Let us first consider the case of closed $M \in \Lambda^0$ i.e. $m = 0$ (notice that the measure $\mu_{\mathbb{R}^m \times \mathbb{S}}$ is not finite, for $m \geq 1$). As $\mathbb{T}_{M,\mathsf{term}}$ now coincides with $\mathsf{value}_M^{-1}(\Lambda_v^0)$, $\mathbb{T}_{M,\mathsf{term}}$ is a measurable subset of $\mathbb{S}$. Plainly if M is deterministic (i.e. sample-free), then $\mu_{\mathbb{S}}(\mathbb{T}_{M,\mathsf{term}}) = 1$ if M converges to a value, and 0 otherwise. Generally for an arbitrary (stochastic) term M we can regard $\mu_{\mathbb{S}}(\mathbb{T}_{M,\mathsf{term}})$ as the probability that a run of M converges to a value, because of Lem. 1.

Lemma 1. *If $M \in \Lambda^0$ then $\mu_{\mathbb{S}}(\mathbb{T}_{M,\mathsf{term}}) \leq 1$.*

More generally, if M has free variables amongst $x_1, \ldots, x_m$ (all of type R), then we say that M is almost surely terminating if for almost every (instantiation of the free variables by) $\boldsymbol{r} \in \mathbb{R}^m$, $M[\boldsymbol{r}/\boldsymbol{x}]$ terminates with probability 1.

We formalise the notion of almost sure termination as follows.

Definition 1. Let M be an SPCF term. We say that M ***terminates almost surely*** if

1. M is closed and $\mu(\mathbb{T}_{M,\mathsf{term}}) = \mu(\mathsf{value}_M^{-1}(\Lambda_v^0)) = 1$; or
2. M has free variables amongst $x_1, \ldots, x_m$ (all of which are of type R), and there exists $T \in \Sigma_{\mathbb{R}^m}$ such that $\mathrm{Leb}_m(\mathbb{R}^m \setminus T) = 0$ and for each $\boldsymbol{r} \in T$, $M[\underline{\boldsymbol{r}}/\boldsymbol{x}]$ terminates almost surely.

Suppose that M is a closed term and $M^\flat$ is obtained from M by recursively replacing subterms $\mathsf{score}(L)$ with the term $\mathsf{if}\big(L < 0, N_{\mathsf{fail}}, L\big)$, where N_{fail} is a term that reduces to fail such as $\underline{1}/\underline{0}$. It is easy to see that for all $\boldsymbol{s} \in \mathbb{S}$, $\left\langle M^\flat, 1, [] \right\rangle \to^* \langle V, 1, \boldsymbol{s} \rangle$ iff for some (unique) $w \in \mathbb{R}_{\geq 0}$, $\langle M, 1, [] \rangle \to^* \langle V, w, \boldsymbol{s} \rangle$. Therefore,

$$\begin{aligned}
[\![M^\flat]\!](\Lambda_v) &= \int_{\mathsf{value}_{M^\flat}^{-1}(\Lambda_v)} \mathrm{d}\mu_{\mathbb{S}}\, \mathsf{weight}_{M^\flat} \\
&= \mu_{\mathbb{S}}(\{\boldsymbol{s} \in \mathbb{S} \mid \exists V \,.\, \left\langle M^\flat, 1, [] \right\rangle \to^* \langle V, 1, \boldsymbol{s} \rangle\}) = \mu_{\mathbb{S}}(\mathbb{T}_{M,\mathsf{term}})
\end{aligned}$$

Consequently, the closed term M terminates almost surely iff $[\![M^\flat]\!]$ is a probability measure.

Remark 2. – Like many treatments of semantics of probabilistic programs in the literature, we make no distinction between non-terminating runs and aborted runs of a (closed) term M: both could result in the value semantics $[\![M^\flat]\!]$ being a sub-probabilty measure (cf. [4]).

– Even so, current probabilistic programming systems do not place any restrictions on the code that users can write: it is perfectly possible to construct invalid models because catching programs that do not define valid probability distributions can be hard, or even impossible. This is not surprising, because almost sure termination is hard to decide: it is Π^0_2-complete in the arithmetic hierarchy [22]. Nevertheless, because a.s. termination is an important correctness property of probabilistic programs (not least because of the main result of this paper, Thm. 3), the development of methods to prove a.s. termination is a hot research topic.

Accordingly the main theorem of this paper is stated as follows:

Theorem 3. *Let M be an SPCF term (possibly with free variables of type* R*) which terminates almost surely. Then its weight function* weight_M *and value function* value_M *are differentiable almost everywhere.*

5 Stochastic Symbolic Execution

We have seen that a source of discontinuity is the use of if-statements. Our main result therefore relies on an in-depth understanding of the branching behaviour of programs. The operational semantics given in Sec. 3 is unsatisfactory in this respect: any two execution paths are treated independently, whether they go through different branches of an if-statement or one is obtained from the other by using slightly perturbed random samples not affecting the control flow.

More concretely, note that although we have derived $\mathsf{weight}_{\mathsf{Ped}}[0.2, 0.9, 0.7] = 0.54$ and $\mathsf{value}_{\mathsf{Ped}}[0.2, 0.9, 0.7] = \underline{0.6}$ in Ex. 2, we cannot infer anything about $\mathsf{weight}_{\mathsf{Ped}}[0.21, 0.91, 0.71]$ and $\mathsf{value}_{\mathsf{Ped}}[0.21, 0.91, 0.71]$ unless we perform the corresponding reduction.

So we propose an alternative *symbolic* operational semantics (similar to the "compilation scheme" in [55]), in which no sampling is performed: whenever a sample command is encountered, we simply substitute a fresh variable α_i for it, and continue on with the execution. We can view this style of semantics as a stochastic form of symbolic execution [12,23], i.e., a means of analysing a program so as to determine what *inputs*, and *random draws* (from sample) cause each part of a program to execute.

Consider the term $M \equiv \mathsf{let}\ x = \mathsf{sample} \cdot \underline{3}\ \mathsf{in}\ (\mathsf{walk}\, x)$, defined using the function walk of Ex. 1. We have a reduction path

$$M \Rightarrow \mathsf{let}\ (x = \alpha_1 \cdot \underline{3})\ \mathsf{in}\ (\mathsf{walk}\, x) \Rightarrow \mathsf{walk}\,(\alpha_1 \cdot \underline{3})$$

but at this point we are stuck: the CBV strategy requires a value for α_1. We will "delay" the evaluation of the multiplication $\alpha_1 \cdot \underline{3}$; we signal this by drawing a box around the delayed operation: $\alpha_1 \boxdot \underline{3}$. We continue the execution, inspecting the definition of walk, and get:

$$M \Rightarrow^* \mathsf{walk}\,(\alpha_1 \boxdot \underline{3}) \Rightarrow^* N \equiv \mathsf{if}\big(\alpha_1 \boxdot \underline{3} \leq 0, \underline{0}, P\big)$$

where

$$P \equiv \begin{pmatrix} \mathsf{let}\ s = \mathsf{sample}\ \mathsf{in} \\ \mathsf{if}\big((\mathsf{sample} \leq \underline{0.5}), \big(s + \mathsf{walk}(\alpha_1 \boxdot \underline{3} + s)\big), \big(s + \mathsf{walk}(\alpha_1 \boxdot \underline{3} - s)\big)\big) \end{pmatrix}.$$

We are stuck again: the value of α_1 is needed in order to know which branch to follow. Our approach consists in considering the space $\mathbb{S}_1 = (0,1)$ of possible values for α_1, and splitting it into $\{s_1 \in (0,1) \mid s_1 \cdot 3 \leq 0\} = \emptyset$ and $\{s_1 \in (0,1) \mid s_1 \cdot 3 > 0\} = (0,1)$. Each of the two branches will then yield a weight function restricted to the appropriate subspace.

Formally, our symbolic operational semantics is a rewrite system of configurations of the form $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$, where $\mathcal{M}$ is a term with delayed (boxed) operations, and free "sampling" variables[5] $\alpha_1, \ldots, \alpha_n$; $U \subseteq \mathbb{S}_n$ is the subspace of sampling values compatible with the current branch; and $w : U \to \mathbb{R}_{\geq 0}$ is a function assigning to each $s \in U$ a weight $w(s)$. In particular, for our running example[6]

$$\langle\!\langle M, \mathbb{\lambda}[]. 1, \mathbb{S}_0 \rangle\!\rangle \Rightarrow^* \langle\!\langle N, \mathbb{\lambda}[s_1]. 1, (0,1) \rangle\!\rangle.$$

As explained above, this leads to two branches:

$$\langle\!\langle N, \mathbb{\lambda}[s_1]. 1, (0,1) \rangle\!\rangle \begin{array}{l} \nearrow^* \ \langle\!\langle \underline{0}, \mathbb{\lambda}[s_1]. 1, \emptyset \rangle\!\rangle \\ \searrow^* \ \langle\!\langle P, \mathbb{\lambda}[s_1]. 1, (0,1) \rangle\!\rangle \end{array}$$

The first branch has reached a value, and the reader can check that the second branch continues as

$$\begin{aligned} &\langle\!\langle P, \mathbb{\lambda}[s_1]. 1, (0,1) \rangle\!\rangle \Rightarrow^* \\ &\langle\!\langle \mathsf{if}\big(\alpha_3 \leq \underline{0.5}, \alpha_2 + \mathsf{walk}(\alpha_1 \boxdot \underline{3} + \alpha_2), \alpha_2 + \mathsf{walk}(\alpha_1 \boxdot \underline{3} - \alpha_2)\big), \mathbb{\lambda}[s_1, s_2, s_3]. 1, (0,1)^3 \rangle\!\rangle \end{aligned}$$

where α_2 and α_3 stand for the two sample statements in P. From here we proceed by splitting $(0,1)^3$ into $(0,1) \times (0,1) \times (0, 0.5]$ and $(0,1) \times (0,1) \times (0.5, 1)$ and after having branched again (on whether we have passed 0) the evaluation of walk can terminate in the configuration

$$\langle\!\langle \alpha_2 \boxed{+} 0, \mathbb{\lambda}[s_1, s_2, s_3]. 1, U \rangle\!\rangle$$

where $U \coloneqq \{[s_1, s_2, s_3] \in \mathbb{S}_3 \mid s_3 > 0.5 \wedge s_1 \cdot 3 - s_2 \leq 0\}$.

Recall that M appears in the context of our running example Ped. Using our calculations above we derive one of its branches:

$$\begin{aligned} &\langle\!\langle \mathsf{Ped}, \mathbb{\lambda}[]. 1, \{[]\} \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathsf{let\ w} = \mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(\alpha_2))\ \mathsf{in}\ \alpha_1 \boxdot \underline{3}, \mathbb{\lambda}[\mathsf{s_1}, \mathsf{s_2}, \mathsf{s_3}]. 1, \mathsf{U} \rangle\!\rangle \\ &\quad \Rightarrow \langle\!\langle \mathsf{let}\ w = \mathsf{score}(\boxed{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(\alpha_2))\ \mathsf{in}\ \alpha_1 \boxdot \underline{3}, \mathbb{\lambda}[s_1, s_2, s_3]. 1, U \rangle\!\rangle \\ &\quad \Rightarrow^* \langle\!\langle \mathsf{let}\ w = \boxed{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}}(\alpha_2)\ \mathsf{in}\ \alpha_1 \boxdot \underline{3}, \mathbb{\lambda}[s_1, s_2, s_3]. \mathrm{pdf}_{\mathcal{N}(1.1,0.1)}(s_2), U \rangle\!\rangle \\ &\quad \Rightarrow^* \langle\!\langle \alpha_1 \boxdot \underline{3}, \mathbb{\lambda}[s_1, s_2, s_3]. \mathrm{pdf}_{\mathcal{N}(1.1,0.1)}(s_2), U \rangle\!\rangle \end{aligned}$$

[5] Note that $\mathcal{M}$ may be open and contain other free "non-sampling" variables, usually denoted $x_1, \ldots, x_m$.

[6] We use the meta-lambda-abstraction $\mathbb{\lambda}x. f(x)$ to denote the set-theoretic function $x \mapsto f(x)$.

In particular the trace $[0.2, 0.9, 0.7]$ of Ex. 2 lies in the subspace U. We can immediately read off the corresponding value and weight functions for *all* $[s_1, s_2, s_3] \in U$ simply by evaluating the computation $\alpha_1 \cdot \underline{3}$, which we have delayed until now:

$$\mathsf{value}_{\mathsf{Ped}}[s_1, s_2, s_3] = \underline{s_1 \cdot 3} \qquad \mathsf{weight}_{\mathsf{Ped}}[s_1, s_2, s_3] = \mathrm{pdf}_{\mathcal{N}(1.1,0.1)}(s_2)$$

5.1 Symbolic Terms and Values

We have just described informally our symbolic execution approach, which involves delaying the evaluation of primitive operations. We make this formal by introducing an extended notion of terms, which we call ***symbolic terms*** and define in Fig. 4a along with a notion of ***symbolic values***. For this we assume fixed denumerable sequences of ***distinguished*** variables: $\alpha_1, \alpha_2, \ldots$, used to represent sampling, and $x_1, x_2, \ldots$ used for free variables of type R. Symbolic terms are typically denoted $\mathcal{M}$, $\mathcal{N}$, or $\mathcal{L}$. They contain terms of the form $\boxed{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell)$ for $f : \mathbb{R}^\ell \rightharpoonup \mathbb{R} \in \mathcal{F}$ a primitive function, representing delayed evaluations, and they also contain the sampling variables α_j. The type system is adapted in a straightforward way, see Fig. 4b.

We use $\Lambda_{(m,n)}$ to refer to the set of well-typed symbolic terms with free variables amongst $x_1, \ldots, x_m$ and $\alpha_1, \ldots, \alpha_n$ (and all are of type R). Note that every term in the sense of Fig. 2 is also a symbolic term.

Each symbolic term $\mathcal{M} \in \Lambda_{(m,n)}$ has a corresponding set of regular terms, accounting for all possible values for its sampling variables $\alpha_1, \ldots, \alpha_n$ and its (other) free variables $x_1, \ldots, x_m$. For $\boldsymbol{r} \in \mathbb{R}^m$ and $\boldsymbol{s} \in \mathbb{S}_n$, we call ***partially evaluated instantiation*** of $\mathcal{M}$ the term $\lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s})$ obtained from $\mathcal{M}[\underline{\boldsymbol{r}}/\boldsymbol{x}, \underline{\boldsymbol{s}}/\boldsymbol{\alpha}]$ by recursively "evaluating" subterms of the form $\boxed{f}(\underline{r_1}, \ldots, \underline{r_\ell})$ to $\underline{f(r_1, \ldots, r_\ell)}$, provided $(r_1, \ldots, r_\ell) \in \mathsf{dom}(f)$. In this operation, subterms of the form $\underline{f(\underline{r_1}, \ldots, \underline{r_\ell})}$ are left unchanged, and so are any other redexes. $\lfloor \mathcal{M} \rfloor$ can be viewed as a partial function $\lfloor \mathcal{M} \rfloor : \mathbb{R}^m \times \mathbb{S}_n \rightharpoonup \Lambda$ and a formal definition is presented in Fig. 5b. (To be completely rigorous, we define for *fixed* m and n, partial functions $\lfloor \mathcal{M} \rfloor_{m,n} : \mathbb{R}^m \times \mathbb{S}_n \rightharpoonup \Lambda$ for symbolic terms $\mathcal{M}$ whose distinguished variables are amongst $x_1, \ldots, x_m$ and $\alpha_1, \ldots, \alpha_n$. $\mathcal{M}$ may contain other variables $y, z, \ldots$ of any type. Since m and n are usually clear from the context, we omit them.) Observe that for $\mathcal{M} \in \Lambda_{(m,n)}$ and $(\boldsymbol{r}, \boldsymbol{s}) \in \mathsf{dom} \lfloor \mathcal{M} \rfloor$, $\lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s})$ is a closed term.

Example 4. Consider $\mathcal{M} \equiv (\lambda z.\, \alpha_1 \boxdot \underline{3})\, (\mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}(\alpha_2)}))$. Then, for $\boldsymbol{r} = []$ and $\boldsymbol{s} = [0.2, 0.9, 0.7]$, we have $\lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s}) = \underline{(\lambda z.\, \underline{0.6})\, (\mathsf{score}(\underline{\mathrm{pdf}_{\mathcal{N}(1.1,0.1)}(\underline{0.9})}))}$.

More generally, observe that if $\Gamma \vdash \mathcal{M} : \sigma$ and $(\boldsymbol{r}, \boldsymbol{s}) \in \mathsf{dom} \lfloor \mathcal{M} \rfloor$ then $\Gamma \vdash \lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s}) : \sigma$. In order to evaluate conditionals $\mathsf{if}\big(\mathcal{L} \leq 0, \mathcal{M}, \mathcal{N}\big)$ we need to reduce $\mathcal{L}$ to a real constant, i.e., we need to have $\lfloor \mathcal{L} \rfloor (\boldsymbol{r}, \boldsymbol{s}) = \underline{r}$ for some $r \in \mathbb{R}$. This is the case whenever $\mathcal{L}$ is a symbolic value of type R, since these are built only out of delayed operations, real constants and distinguished variables x_i or α_j. Indeed we can show the following:

Lemma 2. *Let* $(\boldsymbol{r}, \boldsymbol{s}) \in \mathsf{dom} \lfloor \mathcal{M} \rfloor$. *Then* $\mathcal{M}$ *is a symbolic value iff* $\lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s})$ *is a value.*

$$\mathcal{V} ::= \underline{r} \mid x_i \mid \alpha_j \mid \boxed{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell) \mid \lambda y.\, \mathcal{M}$$
$$\mathcal{M}, \mathcal{N}, \mathcal{L} ::= \mathcal{V} \mid y \mid \underline{f}(\mathcal{M}_1, \ldots, \mathcal{M}_\ell) \mid \mathcal{M}\,\mathcal{N} \mid \mathsf{Y}\mathcal{M} \mid \mathsf{if}\big(\mathcal{L} \leq 0, \mathcal{M}, \mathcal{N}\big) \mid \mathsf{sample} \mid \mathsf{score}(\mathcal{M})$$

(a) Symbolic values (typically $\mathcal{V}$) and symbolic terms (typically $\mathcal{M}$, $\mathcal{N}$ or $\mathcal{L}$)

$$\frac{\Gamma \vdash \mathcal{V}_1 : \mathsf{R} \cdots \Gamma \vdash \mathcal{V}_\ell : \mathsf{R}}{\Gamma \vdash \boxed{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell) : \mathsf{R}} \qquad \frac{}{\Gamma \vdash x_i : \mathsf{R}} \qquad \frac{}{\Gamma \vdash \alpha_j : \mathsf{R}}$$

$$\frac{}{\Gamma, y : \sigma \vdash y : \sigma} \qquad \frac{}{\Gamma \vdash \underline{r} : \mathsf{R}}\ r \in \mathbb{R} \qquad \frac{\Gamma \vdash \mathcal{M}_1 : \mathsf{R} \cdots \Gamma \vdash \mathcal{M}_\ell : \mathsf{R}}{\Gamma \vdash \underline{f}(\mathcal{M}_1, \ldots, \mathcal{M}_\ell) : \mathsf{R}}$$

$$\frac{\Gamma, y : \sigma \vdash \mathcal{M} : \tau}{\Gamma \vdash \lambda y.\, \mathcal{M} : \sigma \to \tau} \qquad \frac{\Gamma \vdash \mathcal{M} : \sigma \to \tau \Gamma \vdash \mathcal{N} : \sigma}{\Gamma \vdash \mathcal{M}\,\mathcal{N} : \tau} \qquad \frac{\Gamma \vdash \mathcal{M} : (\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \tau}{\Gamma \vdash \mathsf{Y}\mathcal{M} : \sigma \Rightarrow \tau}$$

$$\frac{\Gamma \vdash \mathcal{L} : \mathsf{R} \quad \Gamma \vdash \mathcal{M} : \sigma \Gamma \vdash \mathcal{N} : \sigma}{\Gamma \vdash \mathsf{if}\big(\mathcal{L} \leq 0, \mathcal{M}, \mathcal{N}\big) : \sigma} \qquad \frac{}{\Gamma \vdash \mathsf{sample} : \mathsf{R}} \qquad \frac{\Gamma \vdash \mathcal{M} : \mathsf{R}}{\Gamma \vdash \mathsf{score}(\mathcal{M}) : \mathsf{R}}$$

(b) Type system for symbolic terms

$$\mathcal{R} ::= (\lambda y.\, \mathcal{M})\, \mathcal{V} \mid \underline{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell) \mid \mathsf{Y}(\lambda y.\, \mathcal{M}) \mid \mathsf{if}\big(\mathcal{V} \leq 0, \mathcal{M}, \mathcal{N}\big) \mid \mathsf{sample} \mid \mathsf{score}(\mathcal{V})$$
$$\mathcal{E} ::= [\,] \mid \mathcal{E}\,\mathcal{M} \mid (\lambda y.\, \mathcal{M})\, \mathcal{E} \mid \underline{f}(\mathcal{V}_1, \ldots, \mathcal{V}_{i-1}, \mathcal{E}, \mathcal{M}_{i+1}, \ldots, \mathcal{M}_\ell) \mid \mathsf{Y}\mathcal{E} \mid$$
$$\mathsf{if}\big(\mathcal{E} \leq 0, \mathcal{M}, \mathcal{N}\big) \mid \mathsf{score}(\mathcal{E})$$

(c) Symbolic values (typically $\mathcal{V}$), redexes ($\mathcal{R}$) and reduction contexts ($\mathcal{E}$).

Fig. 4: Symbolic terms and values, type system, reduction contexts, and redexes. As usual $f \in \mathcal{F}$ and $r \in \mathbb{R}$.

For symbolic values $\mathcal{V} : \mathsf{R}$ and $(\boldsymbol{r}, \boldsymbol{s}) \in \mathsf{dom}\,\lfloor \mathcal{V} \rfloor$ we employ the notation $\|\mathcal{V}\|\,(\boldsymbol{r}, \boldsymbol{s}) := r'$ provided that $\lfloor \mathcal{V} \rfloor\,(\boldsymbol{r}, \boldsymbol{s}) = \underline{r'}$.

A simple induction on symbolic terms and values yields the following property, which is crucial for the proof of our main result (Thm. 3):

Lemma 3. *Suppose the set $\mathcal{F}$ of primitives satisfies Item 1 of Assumption 1.*

1. *For each symbolic value $\mathcal{V}$ of type R, by identifying $\mathsf{dom}\,\|\mathcal{V}\|$ with a subset of $\mathbb{R}^{m+n}$, we have $\|\mathcal{V}\| \in \mathcal{F}$.*
2. *If $\mathcal{F}$ also satisfies item 2 of Assumption 1 then for each symbolic term $\mathcal{M}$, $\lfloor \mathcal{M} \rfloor : \mathbb{R}^m \times \mathbb{S}_n \rightharpoonup \Lambda$ is differentiable in the interior of its domain.*

5.2 Symbolic Operational Semantics

We aim to develop a symbolic operational semantics that provides a sound and complete abstraction of the (concrete) operational trace semantics. The symbolic

$$
\begin{aligned}
\mathsf{dom}\left\lfloor \boxed{f}(\mathcal{V}_1,\ldots,\mathcal{V}_\ell)\right\rfloor &\coloneqq \{(\boldsymbol{r},\boldsymbol{s}) \in \mathsf{dom}\left\lfloor \mathcal{V}_1\right\rfloor \cap \cdots \cap \mathsf{dom}\left\lfloor \mathcal{V}_\ell\right\rfloor \mid (r'_1,\ldots,r'_\ell) \in \mathsf{dom}(f),\\
&\qquad \text{where } \underline{r'_1} = \left\lfloor \mathcal{V}_1\right\rfloor(\boldsymbol{r},\boldsymbol{s}),\cdots,\underline{r'_\ell} = \left\lfloor \mathcal{V}_\ell\right\rfloor(\boldsymbol{r},\boldsymbol{s})\}\\
\mathsf{dom}\left\lfloor \mathsf{sample}\right\rfloor &\coloneqq \mathsf{dom}\left\lfloor x_i\right\rfloor \coloneqq \mathsf{dom}\left\lfloor \alpha_j\right\rfloor \coloneqq \mathsf{dom}\left\lfloor y\right\rfloor \coloneqq \mathsf{dom}\left\lfloor \underline{r'}\right\rfloor \coloneqq \mathbb{R}^m \times \mathbb{S}_n\\
\mathsf{dom}\underline{f}(\mathcal{M}_1,\ldots,\mathcal{M}_\ell) &\coloneqq \mathsf{dom}\left\lfloor \mathcal{M}_1\right\rfloor \cap \cdots \cap \mathsf{dom}\left\lfloor \mathcal{M}_\ell\right\rfloor\\
\mathsf{dom}\left\lfloor \lambda y.\,\mathcal{M}\right\rfloor &\coloneqq \mathsf{dom}\left\lfloor \mathsf{Y}\mathcal{M}\right\rfloor \coloneqq \mathsf{dom}\left\lfloor \mathsf{score}(\mathcal{M})\right\rfloor \coloneqq \mathsf{dom}\left\lfloor \mathcal{M}\right\rfloor\\
\mathsf{dom}\left\lfloor \mathcal{M}\,\mathcal{N}\right\rfloor &\coloneqq \mathsf{dom}\left\lfloor \mathcal{M}\right\rfloor \cap \mathsf{dom}\left\lfloor \mathcal{N}\right\rfloor\\
\mathsf{dom}\left\lfloor \mathsf{if}\big(\mathcal{L}\le 0,\mathcal{M},\mathcal{N}\big)\right\rfloor &\coloneqq \mathsf{dom}\left\lfloor \mathcal{L}\right\rfloor \cap \mathsf{dom}\left\lfloor \mathcal{M}\right\rfloor \cap \mathsf{dom}\left\lfloor \mathcal{N}\right\rfloor
\end{aligned}
$$

(a) Domain of $\lfloor\cdot\rfloor$

$$
\begin{aligned}
\left\lfloor \boxed{f}(\mathcal{V}_1,\ldots,\mathcal{V}_\ell)\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \underline{f(r'_1,\ldots,r'_\ell)}\text{, where for } 1\le i\le \ell,\ \left\lfloor \mathcal{V}_i\right\rfloor(\boldsymbol{r},\boldsymbol{s}) = \underline{r'_i}\\
\left\lfloor x_i\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \underline{r_i}\\
\left\lfloor \alpha_j\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \underline{s_j}\\
\left\lfloor y\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq y\\
\left\lfloor \underline{r'}\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \underline{r'}\\
\left\lfloor \underline{f}(\mathcal{M}_1,\ldots,\mathcal{M}_\ell)\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \underline{f}(\left\lfloor \mathcal{M}_1\right\rfloor(\boldsymbol{r},\boldsymbol{s}),\ldots,\left\lfloor \mathcal{M}_\ell\right\rfloor(\boldsymbol{r},\boldsymbol{s}))\\
\left\lfloor \lambda y.\,\mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \lambda y.\left\lfloor \mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s})\\
\left\lfloor \mathcal{M}\,\mathcal{N}\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq (\left\lfloor \mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}))\,(\left\lfloor \mathcal{N}\right\rfloor(\boldsymbol{r},\boldsymbol{s}))\\
\left\lfloor \mathsf{Y}\mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \mathsf{Y}(\left\lfloor \mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}))\\
\left\lfloor \mathsf{if}\big(\mathcal{L}\le 0,\mathcal{M},\mathcal{N}\big)\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \mathsf{if}\big(\left\lfloor \mathcal{L}\right\rfloor(\boldsymbol{r},\boldsymbol{s})\le 0,\left\lfloor \mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}),\left\lfloor \mathcal{N}\right\rfloor(\boldsymbol{r},\boldsymbol{s})\big)\\
\left\lfloor \mathsf{sample}\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \mathsf{sample}\\
\left\lfloor \mathsf{score}(\mathcal{M})\right\rfloor(\boldsymbol{r},\boldsymbol{s}) &\coloneqq \mathsf{score}(\left\lfloor \mathcal{M}\right\rfloor(\boldsymbol{r},\boldsymbol{s}))
\end{aligned}
$$

(b) Definition of $\lfloor\cdot\rfloor$ on $\mathsf{dom}\,\lfloor\cdot\rfloor$

Fig. 5: Formal definition of the instantiation and partial evaluation function $\lfloor\cdot\rfloor$

semantics is presented as a rewrite system of ***symbolic configurations***, which are defined to be triples of the form $\langle\!\langle \mathcal{M}, w, U\rangle\!\rangle$, where for some m and n, $\mathcal{M} \in \Lambda_{(m,n)}$, $U \subseteq \mathsf{dom}\left\lfloor \mathcal{M}\right\rfloor \subseteq \mathbb{R}^m \times \mathbb{S}_n$ is measurable, and $w : \mathbb{R}^m \times \mathbb{S} \rightharpoonup \mathbb{R}_{\ge 0}$ with $\mathsf{dom}(w) = U$. Thus we aim to prove the following result (writing $\mathit{1}$ for the constant function $\lambda(\boldsymbol{r},\boldsymbol{s}).\,1$):

Theorem 1. *Let M be a term with free variables amongst $x_1,\ldots,x_m$.*

1. *(Soundness). If $\langle\!\langle M, \mathit{1}, \mathbb{R}^m\rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}, w, U\rangle\!\rangle$ then for all $(\boldsymbol{r},\boldsymbol{s}) \in U$ it holds $\mathsf{weight}_M(\boldsymbol{r},\boldsymbol{s}) = w(\boldsymbol{r},\boldsymbol{s})$ and $\mathsf{value}_M(\boldsymbol{r},\boldsymbol{s}) = \left\lfloor \mathcal{V}\right\rfloor(\boldsymbol{r},\boldsymbol{s})$.*
2. *(Completeness). If $\boldsymbol{r} \in \mathbb{R}^m$ and $\langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, []\rangle \to^* \langle V, w, \boldsymbol{s}\rangle$ then there exists $\langle\!\langle M, \mathit{1}, \mathbb{R}^m\rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}, w, U\rangle\!\rangle$ such that $(\boldsymbol{r},\boldsymbol{s}) \in U$.*

As formalised by Thm. 1, the key intuition behind symbolic configurations $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$ (that are reachable from a given $\langle\!\langle M, 1, \mathbb{R}^m \rangle\!\rangle$) is that, whenever $\mathcal{M}$ is a symbolic value:

- $\mathcal{M}$ gives a correct *local* view of value_M (restricted to U), and
- w gives a correct *local* view of weight_M (restricted to U);

moreover, the respective third components U (of the symbolic configurations $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$) cover $\mathbb{T}_{M,\mathsf{term}}$.

To establish Thm. 1, we introduce ***symbolic reduction contexts*** and ***symbolic redexes***. These are presented in Fig. 4c and extend the usual notions (replacing real constants with arbitrary symbolic values of type R).

Using Lem. 2 we obtain:

Lemma 4. *If $\mathcal{R}$ is a symbolic redex and $(\boldsymbol{r}, \boldsymbol{s}) \in \mathsf{dom}\,\lfloor \mathcal{R} \rfloor$ then $\lfloor \mathcal{R} \rfloor\,(\boldsymbol{r}, \boldsymbol{s})$ is a redex.*

The following can be proven by a straightforward induction:

Lemma 5 (Subject Construction). *Let $\mathcal{M}$ be a symbolic term.*

1. *If $\mathcal{M}$ is a symbolic value then for all symbolic contexts $\mathcal{E}$ and symbolic redexes $\mathcal{R}$, $\mathcal{M} \not\equiv \mathcal{E}[\mathcal{R}]$.*
2. *If $\mathcal{M} \equiv \mathcal{E}_1[\mathcal{R}_1] \equiv \mathcal{E}_2[\mathcal{R}_2]$ then $\mathcal{E}_1 \equiv \mathcal{E}_2$ and $\mathcal{R}_1 \equiv \mathcal{R}_2$.*
3. *If $\mathcal{M}$ is not a symbolic value and $\mathsf{dom}\,\lfloor \mathcal{M} \rfloor \neq \emptyset$ then there exist $\mathcal{E}$ and $\mathcal{R}$ such that $\mathcal{M} \equiv \mathcal{E}[\mathcal{R}]$.*

The partial instantiation function also extends to symbolic contexts $\mathcal{E}$ in the evident way – we give the full definition in [34].

Now, we introduce the following rules for ***symbolic redex contractions***:

$$\langle\!\langle (\lambda y.\, \mathcal{M})\, \mathcal{V}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{M}[\mathcal{V}/y], w, U \rangle\!\rangle$$

$$\langle\!\langle \underline{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell), w, U \rangle\!\rangle \Rightarrow \langle\!\langle \boxed{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell), w, \mathsf{dom}\left\|\boxed{f}(\mathcal{V}_1, \ldots, \mathcal{V}_\ell)\right\| \cap U \rangle\!\rangle$$

$$\langle\!\langle \mathsf{Y}(\lambda y.\, \mathcal{M}), w, U \rangle\!\rangle \Rightarrow \langle\!\langle \lambda z.\, \mathcal{M}\,[\mathsf{Y}(\lambda y.\, \mathcal{M})/y]\, z, w, U \rangle\!\rangle$$

$$\langle\!\langle \mathsf{if}\big(\mathcal{V} \leq 0, \mathcal{M}, \mathcal{N}\big), w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{M}, w, \|\mathcal{V}\|^{-1}(-\infty, 0] \cap U \rangle\!\rangle$$

$$\langle\!\langle \mathsf{if}\big(\mathcal{V} \leq 0, \mathcal{M}, \mathcal{N}\big), w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{N}, w, \|\mathcal{V}\|^{-1}(0, \infty) \cap U \rangle\!\rangle$$

$$\langle\!\langle \mathsf{sample}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \alpha_{n+1}, w', U' \rangle\!\rangle \qquad (U \subseteq \mathbb{R}^m \times \mathbb{S}_n)$$

$$\langle\!\langle \mathsf{score}(\mathcal{V}), w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{V}, \|\mathcal{V}\| \cdot w, \|\mathcal{V}\|^{-1}[0, \infty) \cap U \rangle\!\rangle$$

In the rule for sample, $U' \coloneqq \{(\boldsymbol{r}, \boldsymbol{s} +\!\!+ [s']) \mid (\boldsymbol{r}, \boldsymbol{s}) \in U \wedge s' \in (0,1)\}$ and $w'(\boldsymbol{r}, \boldsymbol{s} +\!\!+ [s']) \coloneqq w(\boldsymbol{r}, \boldsymbol{s})$; in the rule for $\mathsf{score}(\mathcal{V})$, $(\|\mathcal{V}\| \cdot w)(\boldsymbol{r}, \boldsymbol{s}) \coloneqq \|\mathcal{V}\|\,(\boldsymbol{r}, \boldsymbol{s}) \cdot w(\boldsymbol{r}, \boldsymbol{s})$.

The rules are designed to closely mirror their concrete counterparts. Crucially, the rule for sample introduces a "fresh" sampling variable, and the two rules for conditionals split the last component $U \subseteq \mathbb{R}^m \times \mathbb{S}_n$ according to whether $\|\mathcal{V}\|\,(\boldsymbol{r}, \boldsymbol{s}) \leq 0$ or $\|\mathcal{V}\|\,(\boldsymbol{r}, \boldsymbol{s}) > 0$. The "delay" contraction (second rule) is introduced for a technical reason: ultimately, to enable item 1 (Soundness). Otherwise

it is, for example, unclear whether $\lambda y.\, \alpha_1 + \underline{1}$ should correspond to $\lambda y.\, \underline{0.5} + \underline{1}$ or $\lambda y.\, \underline{1.5}$ for $s_1 = 0.5$.

Finally we lift this to arbitrary symbolic terms using the obvious rule for symbolic evaluation contexts:

$$\frac{\langle\!\langle \mathcal{R}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{R}', w', U' \rangle\!\rangle}{\langle\!\langle \mathcal{E}[\mathcal{R}], w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{E}[\mathcal{R}'], w', U' \rangle\!\rangle}$$

Note that we do not need rules corresponding to reductions to fail because the third component of the symbolic configurations "filters out" the pairs $(\boldsymbol{r}, \boldsymbol{s})$ corresponding to undefined behaviour. In particular, the following holds:

Lemma 6. *Suppose $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$ is a symbolic configuration and $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{N}, w', U' \rangle\!\rangle$. Then $\langle\!\langle \mathcal{N}, w', U' \rangle\!\rangle$ is a symbolic configuration.*

A key advantage of the symbolic execution is that the induced computation tree is finitely branching, since branching only arises from conditionals, splitting the trace space into disjoint subsets. This contrasts with the concrete situation (from Sec. 3), in which sampling creates uncountably many branches.

Lemma 7 (Basic Properties). *Let $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$ be a symbolic configuration. Then*

1. *There are at most countably distinct such U' that $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{N}, w', U' \rangle\!\rangle$.*
2. *If $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}_i, w_i, U_i \rangle\!\rangle$ for $i \in \{1, 2\}$ then $U_1 = U_2$ or $U_1 \cap U_2 = \emptyset$.*
3. *If $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{E}_i[\mathsf{sample}], w_i, U_i \rangle\!\rangle$ for $i \in \{1, 2\}$ then $U_1 = U_2$ or $U_1 \cap U_2 = \emptyset$.*

Crucially, there is a correspondence between the concrete and symbolic semantics in that they can "simulate" each other:

Proposition 1 (Correspondence). *Suppose $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$ is a symbolic configuration, and $(\boldsymbol{r}, \boldsymbol{s}) \in U$. Let $M \equiv \lfloor \mathcal{M} \rfloor (\boldsymbol{r}, \boldsymbol{s})$ and $w := w(\boldsymbol{r}, \boldsymbol{s})$. Then*

1. *If $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{N}, w', U' \rangle\!\rangle$ and $(\boldsymbol{r}, \boldsymbol{s} +\!\!+ \boldsymbol{s'}) \in U'$ then*

$$\langle M, w, \boldsymbol{s} \rangle \to \langle \lfloor \mathcal{N} \rfloor (\boldsymbol{r}, \boldsymbol{s} +\!\!+ \boldsymbol{s'}), w(\boldsymbol{r}, \boldsymbol{s'}), \boldsymbol{s} +\!\!+ \boldsymbol{s'} \rangle\,.$$

2. *If $\langle M, w, \boldsymbol{s} \rangle \to \langle N, w', \boldsymbol{s'} \rangle$ then there exists $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{N}, w', U' \rangle\!\rangle$ such that $\lfloor \mathcal{N} \rfloor (\boldsymbol{r}, \boldsymbol{s'}) \equiv N$, $w'(\boldsymbol{r}, \boldsymbol{s'}) = w'$ and $(\boldsymbol{r}, \boldsymbol{s'}) \in U'$.*

As a consequence of Lem. 2, we obtain a proof of Thm. 1.

6 Densities of Almost Surely Terminating Programs are Differentiable Almost Everywhere

So far we have seen that the symbolic execution semantics provides a sound and complete way to reason about the weight and value functions. In this section we impose further restrictions on the primitive operations and the terms to obtain results about the differentiability of these functions.

Henceforth we assume Assumption 1 and we fix a term M with free variables amongst $x_1, \ldots, x_m$.

From Lem. 3 we immediately obtain the following:

Lemma 8. *Let $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle$ be a symbolic configuration such that w is differentiable on $\mathring{U}$ and $\mu(\partial U) = 0$. If $\langle\!\langle \mathcal{M}, w, U \rangle\!\rangle \Rightarrow \langle\!\langle \mathcal{M}', w', U' \rangle\!\rangle$ then w' is differentiable on $\mathring{U}'$ and $\mu(\partial U') = 0$.*

6.1 Differentiability on Terminating Traces

As an immediate consequence of the preceding, Lem. 3 and the Soundness (item 1 of Thm. 1), whenever $\langle\!\langle M, 1, \mathbb{R}^m \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}, w, U \rangle\!\rangle$ then weight_M and value_M are differentiable everywhere in $\mathring{U}$.

Recall the set $\mathbb{T}_{M,\mathsf{term}}$ of $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}$ from Eq. (1) for which M terminates. We abbreviate $\mathbb{T}_{M,\mathsf{term}}$ to $\mathbb{T}_{\mathsf{term}}$ and define

$$\mathbb{T}_{\mathsf{term}} \coloneqq \mathbb{T}_{M,\mathsf{term}} = \{(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S} \mid \exists V, w\,.\, \langle M[\boldsymbol{r}/\boldsymbol{x}], 1, []\rangle \to^* \langle V, w, \boldsymbol{s}\rangle\}$$
$$\mathbb{T}^{\mathsf{int}}_{\mathsf{term}} \coloneqq \bigcup \{\mathring{U} \mid \exists \mathcal{V}, w\,.\, \langle\!\langle M, 1, \mathbb{R}^m \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}, w, U \rangle\!\rangle\}$$

By Completeness (item 2 of Thm. 1), $\mathbb{T}_{\mathsf{term}} = \bigcup \{U \mid \exists \mathcal{V}, w\,.\, \langle\!\langle M, 1, \mathbb{R}^m \rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{V}, w, U \rangle\!\rangle\}$. Therefore, being countable unions of measurable sets (Lemmas 6 and 7), $\mathbb{T}_{\mathsf{term}}$ and $\mathbb{T}^{\mathsf{int}}_{\mathsf{term}}$ are measurable.

By what we have said above, weight_M and value_M are differentiable everywhere on $\mathbb{T}^{\mathsf{int}}_{\mathsf{term}}$. Observe that in general, $\mathbb{T}^{\mathsf{int}}_{\mathsf{term}} \subsetneq \mathring{\mathbb{T}}_{\mathsf{term}}$. However,

$$\mu\left(\mathbb{T}_{\mathsf{term}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{term}}\right) = \mu\left(\bigcup_{\substack{U : \langle\!\langle M,1,\mathbb{R}^m\rangle\!\rangle \Rightarrow^* \\ \langle\!\langle \mathcal{V},w,U\rangle\!\rangle}} \left(U \setminus \mathring{U}\right)\right) \leq \sum_{\substack{U : \langle\!\langle M,1,\mathbb{R}^m\rangle\!\rangle \Rightarrow^* \\ \langle\!\langle \mathcal{V},w,U\rangle\!\rangle}} \mu(\partial U) = 0 \quad (2)$$

The first equation holds because the U-indexed union is of pairwise disjoint sets. The inequality is due to $(U \setminus \mathring{U}) \subseteq \partial U$. The last equation above holds because each $\mu(\partial U) = 0$ (Assumption 1 and Lem. 8).

Thus we conclude:

Theorem 2. *Let M be an SPCF term. Then its weight function weight_M and value function value_M are differentiable for almost all terminating traces.*

6.2 Differentiability for Almost Surely Terminating Terms

Next, we would like to extend this insight for almost surely terminating terms to suitable subsets of $\mathbb{R}^m \times \mathbb{S}$, the union of which constitutes almost the entirety of $\mathbb{R}^m \times \mathbb{S}$. Therefore, it is worth examining consequences of almost sure termination (see Def. 1).

We say that $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}$ is ***maximal*** (for M) if $\langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, []\rangle \to^* \langle N, w, \boldsymbol{s}\rangle$ and for all $\boldsymbol{s'} \in \mathbb{S} \setminus \{[]\}$ and N', $\langle N, w, \boldsymbol{s}\rangle \not\to^* \langle N', w', \boldsymbol{s} +\!\!+ \boldsymbol{s'}\rangle$. Intuitively, $\boldsymbol{s}$ contains a maximal number of samples to reduce $M[\boldsymbol{r}/\boldsymbol{x}]$. Let $\mathbb{T}_{\mathsf{max}}$ be the set of maximal $(\boldsymbol{r}, \boldsymbol{s})$.

Note that $\mathbb{T}_{\mathsf{term}} \subseteq \mathbb{T}_{\mathsf{max}}$ and there are terms for which the inclusion is strict (e.g. for the diverging term $M \equiv \mathsf{Y}(\lambda f.\, f)$, $[] \in \mathbb{T}_{\mathsf{max}}$ but $[] \notin \mathbb{T}_{\mathsf{term}}$). Besides,

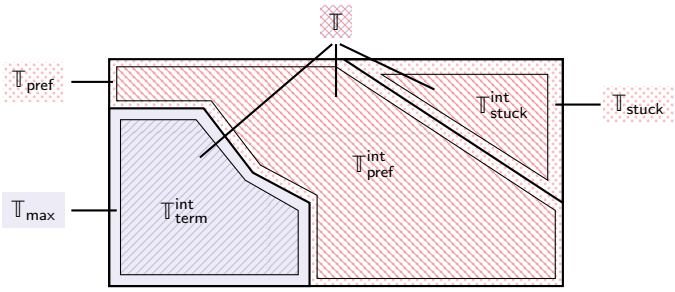


Fig. 6: Illustration of how $\mathbb{R}^m \times \mathbb{S}$ – visualised as the entire rectangle – is partitioned to prove Thm. 3. The value function returns $\bot$ in the red dotted area and a closed value elsewhere (i.e. in the blue shaded area).

$\mathbb{T}_{\mathsf{max}}$ is measurable because, thanks to Prop. 1, for every $n \in \mathbb{N}$,

$$\{(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}_n \mid \langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, []\rangle \to^* \langle N, w, \boldsymbol{s}\rangle\} = \bigcup_{\substack{U : \langle\!\langle M, 1, \mathbb{R}^m\rangle\!\rangle \Rightarrow^* \\ \langle\!\langle \mathcal{N}, w, U\rangle\!\rangle}} U \cap (\mathbb{R}^m \times \mathbb{S}_n)$$

and the RHS is a countable union of measurable sets (Lemmas 6 and 7).

The following is a consequnce of the definition of almost sure termination and a corollary of Fubini's theorem (see [34] for details):

Lemma 9. *If M terminates almost surely then $\mu(\mathbb{T}_{\mathsf{max}} \setminus \mathbb{T}_{\mathsf{term}}) = 0$.*

Now, observe that for all $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{R}^m \times \mathbb{S}$, exactly one of the following holds:

1. $(\boldsymbol{r}, \boldsymbol{s})$ is maximal
2. for a proper prefix $\boldsymbol{s'}$ of $\boldsymbol{s}$, $(\boldsymbol{r}, \boldsymbol{s'})$ is maximal
3. $(\boldsymbol{r}, \boldsymbol{s})$ is *stuck*, because $\boldsymbol{s}$ does not contain enough randomness.

Formally, we say $(\boldsymbol{r}, \boldsymbol{s})$ is ***stuck*** if $\langle M[\underline{\boldsymbol{r}}/\boldsymbol{x}], 1, []\rangle \to^* \langle E[\mathsf{sample}], w, \boldsymbol{s}\rangle$, and we let $\mathbb{T}_{\mathsf{stuck}}$ be the set of all $(\boldsymbol{r}, \boldsymbol{s})$ which get stuck. Thus,

$$\mathbb{R}^m \times \mathbb{S} = \mathbb{T}_{\mathsf{max}} \cup \mathbb{T}_{\mathsf{pref}} \cup \mathbb{T}_{\mathsf{stuck}}$$

where $\mathbb{T}_{\mathsf{pref}} := \{(\boldsymbol{r}, \boldsymbol{s} +\!\!+ \boldsymbol{s'}) \mid (\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{T}_{\mathsf{max}} \wedge \boldsymbol{s'} \neq []\}$, and the union is disjoint.

Defining $\mathbb{T}^{\mathsf{int}}_{\mathsf{stuck}} := \bigcup\{\mathring{U} \mid \langle\!\langle M, 1, \mathbb{R}^m\rangle\!\rangle \Rightarrow^* \langle\!\langle \mathcal{E}[\mathsf{sample}], w, U\rangle\!\rangle\}$ we can argue analogously to Eq. (2) that $\mu(\mathbb{T}_{\mathsf{stuck}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{stuck}}) = 0$.

Moreover, for $\mathbb{T}^{\mathsf{int}}_{\mathsf{pref}} := \{(\boldsymbol{r}, \boldsymbol{s} +\!\!+ \boldsymbol{s'}) \mid (\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{T}^{\mathsf{int}}_{\mathsf{term}} \text{ and } [] \neq \boldsymbol{s'} \in \mathbb{S}\}$ it holds

$$\mathbb{T}_{\mathsf{pref}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{pref}} = \bigcup_{n \in \mathbb{N}} \left\{(\boldsymbol{r}, \boldsymbol{s} +\!\!+ \boldsymbol{s'}) \mid (\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{T}_{\mathsf{max}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{term}} \wedge \boldsymbol{s'} \in \mathbb{S}_n\right\}$$

and hence, $\mu(\mathbb{T}_{\mathsf{pref}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{pref}}) \leq \sum_{n \in \mathbb{N}} \mu(\mathbb{T}_{\mathsf{max}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{term}}) \leq 0$.

Finally, we define

$$\mathbb{T} := \mathbb{T}^{\mathsf{int}}_{\mathsf{term}} \cup \mathbb{T}^{\mathsf{int}}_{\mathsf{pref}} \cup \mathbb{T}^{\mathsf{int}}_{\mathsf{stuck}}$$

Clearly, this is an open set and the situation is illustrated in Fig. 6. By what we have seen,

$$\mu\left((\mathbb{R}^m \times \mathbb{S}) \setminus \mathbb{T}\right) = \mu(\mathbb{T}_{\mathsf{term}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{term}}) + \mu(\mathbb{T}^{\mathsf{int}}_{\mathsf{pref}} \setminus \mathbb{T}_{\mathsf{pref}}) + \mu(\mathbb{T}_{\mathsf{stuck}} \setminus \mathbb{T}^{\mathsf{int}}_{\mathsf{stuck}}) = 0$$

Moreover, to conclude the proof of our main result Thm. 3 it suffices to note:

1. weight_M and value_M are differentiable everywhere on $\mathbb{T}^{\mathsf{int}}_{\mathsf{term}}$ (as for Thm. 2), and
2. $\mathsf{weight}_M(\boldsymbol{r}, \boldsymbol{s}) = 0$ and $\mathsf{value}_M(\boldsymbol{r}, \boldsymbol{s}) = \bot$ for $(\boldsymbol{r}, \boldsymbol{s}) \in \mathbb{T}^{\mathsf{int}}_{\mathsf{pref}} \cup \mathbb{T}^{\mathsf{int}}_{\mathsf{stuck}}$.

Theorem 3. *Let M be an SPCF term (possibly with free variables of type R) which terminates almost surely. Then its weight function weight_M and value function value_M are differentiable almost everywhere.*

We remark that almost sure termination was not used in our development until the proof of Lem. 9. For Thm. 3 we could have instead directly assumed the conclusion of Lem. 9; that is, almost all maximal traces are terminating. This is a strictly weaker condition than almost sure termination. The exposition we give is more appropriate: almost sure termination is a standard notion, and the development of methods to prove almost sure termination is a subject of active research.

We also note that the technique used in this paper to establish almost everywhere differentiability could be used to target another "almost everywhere" property instead: one can simply remove the requirement that elements of $\mathcal{F}$ are differentiable, and replace it with the desired property. A basic example of this is *smoothness*.

7 Conclusion

We have solved an open problem in the theory of probabilistic programming. This is mathematically interesting, and motivated the development of stochastic symbolic execution, a more informative form of operational semantics in this context. The result is also of major practical interest, since almost everywhere differentiability is necessary for correct gradient-based inference.

Related Work. This problem was partially addressed in the work of Zhou et al. [55] who prove a restricted form of our theorem for recursion-free first-order programs with analytic primitives. Our stochastic symbolic execution is related to their *compilation scheme*, which we extend to a more general language.

The idea of considering the possible control paths through a probabilistic programs is fairly natural and not new to this paper; it has been used towards the design of specialised inference algorithms for probabilistic programming, see [11,56]. To our knowledge, this is the first semantic formalisation of the concept, and the first time it is used to reason about whole-program density.

The notions of *weight function* and *value function* in this paper are inspired by the more standard trace-based operational semantics of Borgström et al. [8] (see also [52,31]).

Mazza and Pagani [35] study the correctness of automatic differentiation (AD) of purely *deterministic* programs. This problem is orthogonal to the work reported here, but it is interesting to combine their result with ours. Specifically, we show a.e. differentiability whilst [35] proves a.s. correctness of AD on the *differentiable* domain. Combining both results one concludes that for a deterministic program, AD returns a correct gradient a.s. on the *entire* domain. Going deeper into the comparison, Mazza and Pagani propose a notion of admissible primitive function strikingly similar to ours: given continuity, their condition 2 and our condition 3 are equivalent. On the other hand we require admissible functions to be differentiable, when they are merely continuous in [35]. Finally, we conjecture that "stable points", a central notion in [35], have a clear counterpart within our framework: for a symbolic evaluation path arriving at $\langle\!\langle \mathcal{V}, w, U \rangle\!\rangle$, for $\mathcal{V}$ a symbolic value, the points of $\mathring{U}$ are precisely the stable points.

Our work is also connected to recent developments in differentiable programming. Lee et al. [30] study the family of *piecewise functions under analytic partition*, or just "PAP" functions. PAP functions are a well-behaved family of almost everywhere differentiable functions, which can be used to reason about automatic differentiation in recursion-free first-order programs. An interesting question is whether this can be extended to a more general language, and whether densities of almost surely terminating SPCF programs are PAP functions. (See also [19,9] for work on differentiable programs *without* conditionals.)

A similar class of functions is also introduced by Bolte and Pauwels [7] in very recent work; this is used to prove a convergence result for stochastic gradient descent in deep learning. Whether this class of functions can be used to reason about probabilistic program densities remains to be explored.

Finally we note that *open logical relations* [1] are a convenient proof technique for establishing properties of programs which hold at first order, such as almost everywhere differentiability. This approach remains to be investigated in this context, as the connection with probabilistic densities is not immediate.

Further Directions. This investigation would benefit from a denotational treatment; this is not currently possible as existing models of probabilistic programming do not account for differentiability.

In another direction, it is likely that we can generalise the main result by extending SPCF with recursive types, as in [51], and, more speculatively, first-class differential operators as in [17]. It would also be useful to add to SPCF a family of *discrete* distributions, and more generally continuous-discrete mixtures, which have practical applications [36].

Our work will have interesting implications in the correctness of various gradient-based inference algorithms, such as the recent discontinuous HMC [39] and reparameterisation gradient for non-differentiable models [32]. But given the lack of guarantees of correctness properties available until now, these algorithms have not yet been developed in full generality, leaving many perspectives open.

Acknowledgements. We thank Wonyeol Lee for spotting an error in an example.

We gratefully acknowledge support from EPSRC and the Royal Society.

References

1. Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. On the versatility of open logical relations. In *European Symposium on Programming*, pages 56–83. Springer, 2020.
2. Sooraj Bhat, Ashish Agarwal, Richard W. Vuduc, and Alexander G. Gray. A type theory for probability density functions. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 545–556. ACM, 2012.
3. Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio V. Russo. Deriving probability density functions from probabilistic functional programs. *Logical Methods in Computer Science*, 13(2), 2017.
4. Benjamin Bichsel, Timon Gehr, and Martin T. Vechev. Fine-grained semantics for probabilistic programs. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 145–185. Springer, 2018.
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. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.
6. David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.
7. Jérôme Bolte and Edouard Pauwels. A mathematical model for automatic differentiation in machine learning. *CoRR*, abs/2006.02080, 2020.
8. Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 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*, pages 33–46, 2016.
9. Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. *Proc. ACM Program. Lang.*, 4(POPL):64:1–64:27, 2020.
10. Simon Castellan and Hugo Paquet. Probabilistic programming inference via intensional semantics. In *European Symposium on Programming*, pages 322–349. Springer, 2019.
11. Arun Chaganty, Aditya Nori, and Sriram Rajamani. Efficiently sampling probabilistic programs via program analysis. In *Artificial Intelligence and Statistics*, pages 153–160, 2013.
12. Lori A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Trans. Software Eng.*, 2(3):215–222, 1976.
13. Ryan Culpepper and Andrew Cobb. Contextual equivalence for probabilistic programs with continuous random variables and scoring. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 368–392. Springer, 2017.

14. Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In Kathryn S. McKinley and Kathleen Fisher, editors, *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 221–236. ACM, 2019.
15. S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics letters B*, 1987.
16. Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *PACMPL*, 2(POPL):59:1–59:28, 2018.
17. Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.
18. Matthew D. Hoffman, David M. Blei, Chong Wang, and John W. Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, 2013.
19. Mathieu Huot, Sam Staton, and Matthijs Vákár. Correctness of automatic differentiation via diffeologies and categorical gluing. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2020.
20. Chung-Kil Hur, Aditya V Nori, Sriram K Rajamani, and Selva Samuel. A provably correct sampler for probabilistic programs. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
21. Wazim Mohammed Ismail and Chung-chieh Shan. Deriving a probability density calculator (functional pearl). In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 47–59. ACM, 2016.
22. Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Inf.*, 56(3):255–285, 2019.
23. James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.
24. Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
25. Oleg Kiselyov. Problems of the Lightweight Implementation of Probabilistic Programming. In *PPS Workshop*, 2016.
26. Dexter Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 101–114, 1979.
27. Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic variational inference in stan. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 568–576, 2015.
28. Jeffrey M. Lee. *Manifolds and Differential Geometry*, volume 107 of *Graduate Studies in Mathematics*. AMS, 2009.

29. John M. Lee. *An introduction to smooth manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer, second edition, 2013.
30. Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. On correctness of automatic differentiation for non-differentiable functions. *CoRR*, abs/2006.06903, 2020.
31. Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. Towards verified stochastic variational inference for probabilistic programs. *PACMPL*, 4(POPL):16:1–16:33, 2020.
32. Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization gradient for non-differentiable models. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 5558–5568, 2018.
33. Alexander K Lew, Marco F Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K Mansinghka. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, 2019.
34. Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. Densities of almost-surely terminating probabilistic programs are differentiable almost everywhere. *CoRR*, abs/2004.03924, 2020.
35. Damiano Mazza and Michele Pagani. Automatic differentiation in pcf. *Proc. ACM Program. Lang.*, 5(POPL), January 2021.
36. Praveen Narayanan and Chung-chieh Shan. Symbolic disintegration with a variety of base measures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 42(2):1–60, 2020.
37. Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, page 113, 2011.
38. Akihiko Nishimura, David B Dunson, and Jianfeng Lu. Discontinuous hamiltonian monte carlo for discrete parameters and discontinuous likelihoods. *Biometrika*, 107(2):365–380, Mar 2020.
39. Akihiko Nishimura, David B Dunson, and Jianfeng Lu. Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods. *Biometrika*, 03 2020. asz083.
40. Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 814–822, 2014.
41. Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 814–822. JMLR.org, 2014.
42. Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1278–1286. JMLR.org, 2014.
43. Walter Rudin. *Principles of Mathematical Analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill Education, 3rd edition edition, 1976.

44. Nasser Saheb-Djahromi. Probabilistic lcf. In *International Symposium on Mathematical Foundations of Computer Science*, pages 442–451. Springer, 1978.
45. Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K Moss, Chris Heunen, and Zoubin Ghahramani. Denotational validation of higher-order bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(POPL):60, 2017.
46. Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theor. Comput. Sci.*, 121(1&2):411–440, 1993.
47. Kurt Sieber. Relating full abstraction results for different programming languages. In *Foundations of Software Technology and Theoretical Computer Science, Tenth Conference, Bangalore, India, December 17-19, 1990, Proceedings*, pages 373–387, 1990.
48. Sam Staton. Commutative semantics for probabilistic programming. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 855–879. Springer, 2017.
49. Michalis K. Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1971–1979. JMLR.org, 2014.
50. Loring W. Tu. *An introduction to manifolds*. Universitext. Springer-Verlag, 2011.
51. Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.
52. Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *PACMPL*, 2(ICFP):87:1–87:30, 2018.
53. David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 770–778. JMLR.org, 2011.
54. Hongseok Yang. Some semantic issues in probabilistic programming languages (invited talk). In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 4:1–4:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
55. Yuan Zhou, Bradley J. Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. LF-PPL: A low-level first order probabilistic programming language for non-differentiable models. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 148–157. PMLR, 2019.
56. Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. Divide, conquer, and combine: a new inference strategy for probabilistic programs with stochastic support. *CoRR*, abs/1910.13324, 2019.