Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming

Raven Beutner*
CISPA Helmholtz Center for Information Security
Germany

C.-H. Luke Ong
University of Oxford
United Kingdom

Fabian Zaiser
University of Oxford
United Kingdom

Abstract

We propose a new method to approximate the posterior distribution of probabilistic programs by means of computing *guaranteed* bounds. The starting point of our work is an interval-based trace semantics for a recursive, higher-order probabilistic programming language with continuous distributions. Taking the form of (super-/subadditive) measures, these lower/upper bounds are non-stochastic and provably correct: using the semantics, we prove that the actual posterior of a given program is sandwiched between the lower and upper bounds (soundness); moreover, the bounds converge to the posterior (completeness). As a practical and sound approximation, we introduce a weight-aware interval type system, which automatically infers interval bounds on not just the return value but also the weight of program executions, simultaneously. We have built a tool implementation, called GuBPI, which automatically computes these posterior lower/upper bounds. Our evaluation on examples from the literature shows that the bounds are useful, and can even be used to recognise wrong outputs from stochastic posterior inference procedures.

CCS Concepts: • **Mathematics of computing** → **Probabilistic inference problems**; • **Theory of computation** → *Program analysis*; • **Software and its engineering** → **Formal methods**.

Keywords: probabilistic programming, Bayesian inference, verification, abstract interpretation, operational semantics, interval arithmetic, type system, symbolic execution

ACM Reference Format:
Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. 2022. Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22), June 13–17, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 32 pages. https://doi.org/10.1145/3519939.3523721

*Member of the Saarbrücken Graduate School of Computer Science.

PLDI '22, June 13–17, 2022, San Diego, CA, USA

1 Introduction

Probabilistic programming is a rapidly developing discipline at the interface of programming and Bayesian statistics [32, 33, 62]. The idea is to express probabilistic models (incorporating the prior distributions) and the observed data as programs, and to use a general-purpose Bayesian inference engine, which acts directly on these programs, to find the posterior distribution given the observations.

Some of the most influential probabilistic programming languages (PPLs) used in practice are *universal* (i.e. the underlying language is Turing-complete); e.g. Church [31], Anglican [61], Gen [18], Pyro [5], and Turing [25]. Using stochastic branching, recursion, and higher-order features, universal PPLs can express arbitrarily complex models. For instance, these language constructs can be used to incorporate probabilistic context free grammars [43], statistical phylogenetics [52], and even physics simulations [3] into probabilistic models. However, expressivity of the PPL comes at the cost of complicating the posterior inference. Consider, for example, the following problem from [41, 42].

Example 1.1 (Pedestrian)**.** A pedestrian has gotten lost on a long road and only knows that they are a random distance between 0 and 3 km from their home. They repeatedly walk a uniform random distance of at most 1 km in either direction, until they find their home. When they arrive, a step counter tells them that they have traveled a distance of 1.1 km in total. Assuming that the measured distance is normally distributed around the true distance with standard deviation 0.1 km, what is the posterior distribution of the starting point? We can model this with a probabilistic program:

$$
\begin{array}{l}
\textbf{let } \mathit{start} = 3 \times \textbf{sample } \mathrm{uniform}(0,1) \textbf{ in} \\
\textbf{letrec } \mathit{walk}\ x = \textbf{if } x \le 0 \textbf{ then } 0 \textbf{ else} \\
\quad \textbf{let } \mathit{step} = \textbf{sample } \mathrm{uniform}(0,1) \textbf{ in} \\
\quad \mathit{step} + \mathit{walk}\big((x+\mathit{step}) \oplus_{0.5} (x-\mathit{step})\big) \\
\textbf{let } \mathit{distance} = \mathit{walk}\ \mathit{start} \textbf{ in} \\
\textbf{observe } \mathit{distance} \textbf{ from } \mathrm{Normal}(1.1, 0.1); \\
\mathit{start}
\end{array}
$$

Here **sample** uniform(a, b) samples a uniformly distributed value in $[a, b]$, $\oplus_{0.5}$ is probabilistic branching, and **observe** M **from** D observes the value of M from distribution D.

Example 1.1 is a challenging model for inference algorithms in several regards: not only does the program use

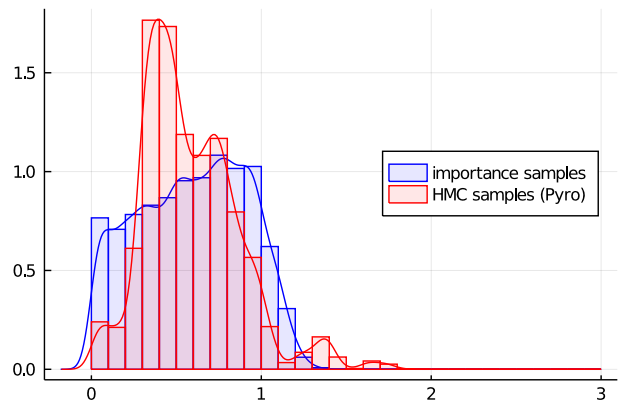


Figure 1. Histogram of samples from the posterior distribution of Example 1.1 and wrong samples produced by the probabilistic programming system Pyro.

stochastic branching and recursion, but the number of random variables generated is unbounded – it's *nonparametric* [29, 37, 42]. To approximate the posterior distribution of the program, we apply two standard inference algorithms: likelihood-weighted importance sampling (IS), a simple algorithm that works well on low-dimensional models with few observations [49]; and Hamiltonian Monte Carlo (HMC) [21], a successful MCMC algorithm that uses gradient information to efficiently explore the parameter space of high-dimensional models. Figure 1 shows the results of the two inference methods as implemented in Anglican [61] (for IS) and Pyro [5] (for HMC): they clearly disagree! But how is the user supposed to know which (if any) of the two results is correct?

Note that *exact* inference methods (i.e. methods that try to compute a closed-form solution of the posterior inference problem using computer algebra and other forms of symbolic computation) such as PSI [26, 27], Hakaru [47], Dice [38], and SPPL [55] are only applicable to non-recursive models, and so they don't work for Example 1.1.

1.1 Guaranteed Bounds

The above example illustrates central problems with both approximate stochastic and exact inference methods. For approximate methods, there are no guarantees for the results they output after a finite amount of time, leading to unclear inference results (as seen in Fig. 1).[1] For exact methods, the symbolic engine may fail to find a closed-form description of the posterior distribution and, more importantly, they are only applicable to very restricted classes of programs (most notably, non-recursive models).

Instead of computing approximate or exact results, this work is concerned with computing *guaranteed* bounds on the posterior distribution of a probabilistic program. Concretely, given a probabilistic program P and a measurable set $U \subseteq \mathbb{R}$ (given as an interval), we infer upper and lower bounds on $[\![P]\!](U)$ (formally defined in Section 2), i.e. the posterior probability of P on U.[2] Such bounds provide a ground truth to compare approximate inference results with: if the approximate results violate the bounds, the inference algorithm has not converged yet or is even ill-suited to the program in question. Crucially, our method is applicable to arbitrary (and in particular recursive) programs of a universal PPL. For Example 1.1, the bounds computed by our method (which we give in Section 7) are tight enough to separate the IS and HMC output. In this case, our method infers that the results given by HMC are wrong (i.e. violate the guaranteed bounds) whereas the IS results are plausible (i.e. lie within the guaranteed bounds). To the best of our knowledge, no existing methods can provide such definite answers for programs of a universal PPL.

1.2 Contributions

The starting point of our work is an interval-based operational semantics [4]. In our semantics, we evaluate a program on *interval traces* (i.e. sequences of intervals of reals with endpoints between 0 and 1) to approximate the outcomes of sampling, and use interval arithmetic [19] to approximate numerical operations (Section 3). Our semantics is sound in the sense that any (compatible and exhaustive) set of interval traces yields lower and upper bounds on the posterior distribution of a program. These lower/upper bounds are themselves super-/subadditive measures. Moreover, under mild conditions (mostly restrictions on primitive operations), our semantics is also complete, i.e. for any $\epsilon > 0$ there exists a countable set of interval traces that provides ϵ-tight bounds on the posterior. Our proofs hinge on a combination of stochastic symbolic execution and the convergence of Riemann sums, providing a natural correspondence between our interval trace semantics and the theory of (Riemann) integration (Section 4).

Based on our interval trace semantics, we present a practical algorithm to automate the computation of guaranteed bounds. It employs an interval type system (together with constraint-based type inference) that bounds both the value of an expression in a refinement-type fashion *and* the score weight of any evaluation thereof. The (interval) bounds inferred by our type system fit naturally in the domain of our semantics. This enables a sound approximation of the

[1] Take MCMC sampling algorithms. Even though the Markov chain will eventually converge to the target distribution, we do not know how long to iterate the chain to ensure convergence [49, 53]. Likewise for variational inference [64]: given a variational family, there is no guarantee that a given value for the KL-divergence (from the approximating to the posterior distribution) is attainable by the minimising distribution.

[2] By repeated application of our method on a discretisation of the domain we can compute histogram-like bounds.

behaviour of a program with finitely many interval traces (Section 5).

We implemented our approach in a tool called GuBPI[3] (**Gu**aranteed **B**ounds for **P**osterior **I**nference), described in Section 6, and evaluate it on a suite of benchmark programs from the literature. We find that the bounds computed by GuBPI are competitive in many cases where the posterior could already be inferred exactly. Moreover, GuBPI's bounds are useful (in the sense that they are precise enough to rule out erroneous approximate results as in Fig. 1, for instance) for recursive models that could not be handled rigorously by any method before (Section 7).

1.3 Scope and Limitations

The contributions of this paper are of both theoretical and practical interest. On the theoretical side, our novel semantics underpins a sound and deterministic method to compute guaranteed bounds on program denotations. As shown by our completeness theorem, this analysis is applicable—in the sense that it computes arbitrarily tight bounds—to a very broad class of programs. On the practical side, our analyser GuBPI implements (an optimised version of) our semantics. As is usual for exact/guaranteed[4] methods, our semantics considers an exponential number of program paths, and partitions each sampled value into a finite number of interval approximations. Consequently, GuBPI generally struggles with high-dimensional models. We believe GuBPI to be most useful for unit-testing of implementations of Bayesian inference algorithms such as Example 1.1, or to compute results on (recursive) programs when non-stochastic, guaranteed bounds are needed.

2 Background

2.1 Basic Probability Theory and Notation

We assume familiarity with basic probability theory, and refer to [51] for details. Here we just fix the notation. A *measurable space* is a pair (Ω, Σ_Ω) where Ω is a set (of outcomes) and $\Sigma_\Omega \subseteq 2^\Omega$ is a σ-algebra defining the measurable subsets of Ω. A *measure* on (Ω, Σ_Ω) is a function $\mu : \Sigma_\Omega \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ that satisfies $\mu(\emptyset) = 0$ and is σ-additive. For $\mathbb{R}^n$, we write $\Sigma_{\mathbb{R}^n}$ for the Borel σ-algebra and λ_n for the Lebesgue measure on $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$. The Lebesgue integral of a measurable function f with respect to a measure μ is written $\int f \, \mathrm{d}\mu$ or $\int f(x) \, \mu(\mathrm{d}x)$. Given a predicate ψ on Ω, we define the Iverson brackets $[\psi] : \Omega \to \mathbb{R}$ by mapping all elements that satisfy ψ to 1 and all others to 0. For $A \in \Sigma_\Omega$ we define the bounded integral $\int_A f \, \mathrm{d}\mu := \int f(x) \cdot [x \in A] \mu(\mathrm{d}x)$.

[3] GuBPI (pronounced "guppy") is available at gubpi-tool.github.io.

[4] By "exact/guaranteed methods", we mean inference algorithms that compute deterministic (non-stochastic) results about the mathematical denotation of a program. In particular, they are correct with probability 1, contrary to stochastic methods.

$$\frac{}{((\lambda x.M)V, s, w) \to (M[V/x], s, w)} \qquad \frac{}{(\mathsf{sample}, r\, s, w) \to (\underline{r}, s, w)}$$

$$\frac{}{((\mu^{\varphi}_{x}.\, M)V, s, w) \to (M[V/x, (\mu^{\varphi}_{x}.\, M)/\varphi], s, w)}$$

$$\frac{}{(f(\underline{r_1}, \ldots, \underline{r_{|f|}}), s, w) \to (\underline{f(r_1, \ldots, r_{|f|})}, s, w)}$$

$$\frac{r \leq 0}{(\mathsf{if}(\underline{r}, N, P), s, w) \to (N, s, w)} \qquad \frac{r > 0}{(\mathsf{if}(\underline{r}, N, P), s, w) \to (P, s, w)}$$

$$\frac{r \geq 0}{(\mathsf{score}(\underline{r}), s, w) \to (\underline{r}, s, w \cdot r)} \qquad \frac{(R, s, w) \to (M, s', w')}{(E[R], s, w) \to (E[M], s', w')}$$

Figure 2. Standard (CbV) reduction rules for SPCF ($\to$).

2.2 Statistical PCF (SPCF)

As our probabilistic programming language of study, we use *statistical PCF* (SPCF) [41], a typed variant of [7]. SPCF includes primitive operations which are measurable functions $f : \mathbb{R}^{|f|} \to \mathbb{R}$, where $|f| \geq 0$ denotes the arity of the function. *Values* and *terms* of SPCF are defined as follows:

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

where x and φ are variables, f is a primitive operation, and $\underline{r}$ a constant with $r \in \mathbb{R}$. Note that we write $\mu^{\varphi}_{x}.\, M$ instead of $\mathsf{Y}(\lambda \varphi x.M)$ for the fixpoint construct. The branching construct is $\mathsf{if}(M, N, P)$, which evaluates to N if $M \leq 0$ and P otherwise. In SPCF, sample draws a random value from the uniform distribution on $[0, 1]$, and $\mathsf{score}(M)$ weights the current execution with the value of M. Samples from a different real-valued distribution D can be obtained by applying the inverse of the cumulative distribution function for D to a uniform sample [54]. Most PPLs feature an **observe** statement instead of manipulating the likelihood weight directly with score, but they are equally expressive [59].[5] As usual, we write $\mathsf{let}\, x = M \,\mathsf{in}\, N$ for $(\lambda x.N)M$, $M; N$ for $\mathsf{let}\, \_ = M \,\mathsf{in}\, N$ and $M \oplus_p N$ for $\mathsf{if}(\mathsf{sample} - p, M, N)$. The type system of our language is as expected, with simple types being generated by $\alpha, \beta := \mathbf{R} \mid \alpha \to \beta$. Selected rules are given below:

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

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

[5] In Bayesian terms, an **observe** statement multiplies the likelihood function by the probability (density) of the observation [33] (as we have used in Example 1.1). Scoring makes this explicit by keeping a weight for each program execution [7]. Observing a value v from a distribution D then simply multiplies the current weight by $\mathrm{pdf}_D(v)$ where pdf_D is the probability density function of D (for continuous distributions) or the probability mass function of D (for discrete distributions).

2.3 Trace Semantics

Following [7], we endow SPCF with a trace-based operational semantics. We evaluate a probabilistic program P on a fixed ***trace*** $\boldsymbol{s} = \langle r_1, \dots, r_n \rangle \in \mathbb{T} := \bigcup_{n\in\mathbb{N}} [0,1]^n$, which *predetermines* the probabilistic choices made during the evaluation. Our semantics therefore operates on configurations of the form $(M, \boldsymbol{s}, w)$ where M is an SPCF term, $\boldsymbol{s}$ is a trace and $w \in \mathbb{R}_{\geq 0}$ a weight. The call-by-value (CbV) reduction is given by the rules in Fig. 2, where $E[\cdot]$ denotes a CbV evaluation context. The definition is standard [4, 7, 41]. Given a program $\vdash P : \mathbf{R}$, we call a trace $\boldsymbol{s}$ ***terminating*** just if $(P, \boldsymbol{s}, 1) \to^* (V, \langle\rangle, w)$ for some value V and weight w, i.e. if the samples drawn are as specified by $\boldsymbol{s}$, the program P terminates. Note that we require the trace $\boldsymbol{s}$ to be completely used up. As P is of type $\mathbf{R}$ we can assume that $V = \underline{r}$ for some $r \in \mathbb{R}$. Each terminating trace $\boldsymbol{s}$ therefore uniquely determines the returned *value* $\underline{r}$ where $r =: \mathrm{val}_P(\boldsymbol{s}) \in \mathbb{R}$, and the *weight* $w =: \mathrm{wt}_P(\boldsymbol{s}) \in \mathbb{R}_{\geq 0}$, of the execution. For a nonterminating trace $\boldsymbol{s}$, $\mathrm{val}_P(\boldsymbol{s})$ is undefined and $\mathrm{wt}_P(\boldsymbol{s}) := 0$.

Example 2.1. Consider Example 1.1. On the trace $\boldsymbol{s} = \langle 0.1, 0.2, 0.4, 0.7, 0.8\rangle \in [0,1]^5 \subseteq \mathbb{T}$, the pedestrian walks 0.2 away from their home (taking the left branch of $\oplus_{0.5}$ as $0.4 \leq 0.5$) and 0.7 towards their home (as $0.8 > 0.5$), hence:

$$\mathrm{val}_P(\boldsymbol{s}) = 3 \times 0.1 = 0.3, \;\; \mathrm{wt}_P(\boldsymbol{s}) = \mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}(0.9).$$

In order to do measure theory, we need to turn our set of traces into a measurable space. The trace space $\mathbb{T}$ is equipped with the σ-algebra $\Sigma_{\mathbb{T}} := \{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \Sigma_{[0,1]^n}\}$ where $\Sigma_{[0,1]^n}$ is the Borel σ-algebra on $[0,1]^n$. We define a measure $\mu_{\mathbb{T}}$ by $\mu_{\mathbb{T}}(U) := \sum_{n\in\mathbb{N}} \lambda_n(U \cap [0,1]^n)$, as in [7].

We can now define the semantics of an SPCF program $\vdash P : \mathbf{R}$ by using the weight and returned value of (executions of P determined by) individual traces. Given $U \in \Sigma_{\mathbb{R}}$, we need to define the likelihood of P evaluating to a value in U. To this end, we set $\mathrm{val}_P^{-1}(U) := \{\boldsymbol{s} \in \mathbb{T} \mid (P, \boldsymbol{s}, 1) \to^* (\underline{r}, \langle\rangle, w), r \in U\}$, i.e. the set of traces on which the program P reduces to a value in U. As shown in [7, Lem. 9], $\mathrm{val}_P^{-1}(U)$ is measurable. Thus, we can define (cf. [7, 41])

$$[\![P]\!](U) := \int_{\mathrm{val}_P^{-1}(U)} \mathrm{wt}_P(\boldsymbol{s})\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{s}).$$

That is, the integral takes all traces $\boldsymbol{s}$ on which P evaluates to a value in U, weighting each $\boldsymbol{s}$ with the weight $\mathrm{wt}_P(\boldsymbol{s})$ of the corresponding execution. A program P is called ***almost surely terminating (AST)*** if it terminates with probability 1, i.e. $\mu_{\mathbb{T}}(\mathrm{val}_P^{-1}(\mathbb{R})) = 1$. This is a necessary assumption for approximate inference algorithms (since they execute the program). See [7] for a more in-depth discussion of this (standard) sampling-style semantics.

Normalizing constant and integrability. In Bayesian statistics, one is usually interested in the *normalised* posterior, which is a conditional probability distribution. We obtain the normalised denotation as $\mathrm{posterior}_P := \frac{[\![P]\!]}{Z_P}$ where $Z_P := [\![P]\!](\mathbb{R})$ is the ***normalising constant***. We call P ***integrable*** if $0 < Z_P < \infty$. The bounds computed in this paper (on the unnormalised denotation $[\![P]\!]$) allow us to compute bounds on the normalizing constant Z_P, and thereby also on the normalised denotation. All bounds reported in this paper (in particular in Section 7) refer to the *normalised* denotation.

3 Interval Trace Semantics

In order to obtain guaranteed bounds on the distribution denotation $[\![P]\!]$ (and also on $\mathrm{posterior}_P$) of a program P, we present an interval-based semantics. In our semantics, we approximate the outcomes of sample with intervals and handle arithmetic operations by means of interval arithmetic (which is similar to the approach by Beutner and Ong [4] in the context of termination analysis). Our semantics enables us to reason about the denotation of a program *without* considering the uncountable space of traces explicitly.

3.1 Interval Arithmetic

For our purposes, an ***interval*** has the form $[a, b]$ which denotes the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$, where $a \in \mathbb{R} \cup \{-\infty\}$, $b \in \mathbb{R} \cup \{\infty\}$, and $a \leq b$. For consistency, we write $[0, \infty]$ instead of the more typical $[0, \infty)$. For $X \subseteq \mathbb{R} \cup \{-\infty, \infty\}$, we denote by $\mathbb{I}_X$ the set of intervals with endpoints in X, and simply write $\mathbb{I}$ for $\mathbb{I}_{\mathbb{R}\cup\{-\infty,\infty\}}$. An n-dimensional ***box*** is the Cartesian product of n intervals.

We can lift functions on real numbers to intervals as follows: for each $f : \mathbb{R}^n \to \mathbb{R}$ we define $f^{\mathbb{I}} : \mathbb{I}^n \to \mathbb{I}$ by

$$f^{\mathbb{I}}([a_1, b_1], \dots, [a_n, b_n]) := [\inf F, \sup F]$$

where $F := f([a_1, b_1], \dots, [a_n, b_n])$. For common functions like $+$, $-$, $\times$, $|\cdot|$, min, max, and monotonically increasing or decreasing functions $f : \mathbb{R} \to \mathbb{R}$, their interval-lifted counterparts can easily be computed, from the values of the original function on just the endpoints of the input interval. For example, addition lifts to $[a_1, b_1] +^{\mathbb{I}} [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$; similarly for multiplication $\times^{\mathbb{I}}$.

3.2 Interval Traces and Interval SPCF

In our interval interpretation, probabilistic programs are run on *interval traces*. An ***interval trace***, $\langle I_1, \dots, I_n\rangle \in \mathbb{T}_{\mathbb{I}} := \bigcup_{n\in\mathbb{N}} (\mathbb{I}_{[0,1]})^n$, is a finite sequence of intervals $I_1, \dots, I_n$, each with endpoints between 0 and 1. To distinguish ordinary traces $\boldsymbol{s} \in \mathbb{T}$ from interval traces $\boldsymbol{t} \in \mathbb{T}_{\mathbb{I}}$, we call the former *concrete* traces.

We define the ***refinement*** relation $\triangleleft$ between concrete and interval traces as follows: for $\boldsymbol{s} = \langle r_1, \dots, r_n\rangle \in \mathbb{T}$ and $\boldsymbol{t} = \langle I_1, \dots, I_m\rangle \in \mathbb{T}_{\mathbb{I}}$, we define $\boldsymbol{s} \triangleleft \boldsymbol{t}$ just if $n = m$ and for all i, $r_i \in I_i$. For each interval trace $\boldsymbol{t}$, we denote by $(\!|\boldsymbol{t}|\!) := \{\boldsymbol{s} \in \mathbb{T} \mid \boldsymbol{s} \triangleleft \boldsymbol{t}\}$ the set of all refinements of $\boldsymbol{t}$.

To define a reduction of a term on an interval trace, we extend SPCF with *interval literals* $\underline{[a, b]}$, which replace the literals $\underline{r}$ but are still considered values of type $\mathbf{R}$. In fact, $\underline{r}$

$$\frac{}{((\lambda x.M)V, \boldsymbol{t}, w) \to_{\mathbb{I}} (M[V/x], \boldsymbol{t}, w)} \qquad \frac{}{(\mathsf{sample}, I\,\boldsymbol{t}, w) \to_{\mathbb{I}} (\underline{I}, \boldsymbol{t}, w)}$$

$$\frac{}{((\mu^{\varphi}_{x}.\,M)V, \boldsymbol{t}, w) \to_{\mathbb{I}} (M[V/x, (\mu^{\varphi}_{x}.\,M)/\varphi], \boldsymbol{t}, w)}$$

$$\frac{b \le 0}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (N, \boldsymbol{t}, w)}$$

$$\frac{a > 0}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (P, \boldsymbol{t}, w)}$$

$$\frac{}{(f(\underline{I_1}, \dots, \underline{I_{|f|}}), \boldsymbol{t}, w) \to_{\mathbb{I}} (\underline{f^{\mathbb{I}}(I_1, \dots, I_{|f|})}, \boldsymbol{t}, w)}$$

$$\frac{a \ge 0}{(\mathsf{score}(\underline{[a,b]}), \boldsymbol{t}, w) \to_{\mathbb{I}} (\underline{[a,b]}, \boldsymbol{t}, w \times^{\mathbb{I}} [a,b])}$$

$$\frac{(R, \boldsymbol{t}, w) \to_{\mathbb{I}} (M, \boldsymbol{t}', w')}{(E[R], \boldsymbol{t}, w) \to_{\mathbb{I}} (E[M], \boldsymbol{t}', w')}$$

Figure 3. Interval reduction rules for (interval) SPCF ($\to_{\mathbb{I}}$).

can be read as an abbreviation for $\underline{[r,r]}$. We call such terms ***interval terms***, and the resulting language ***Interval SPCF***.

Reduction. The interval-based reduction $\to_{\mathbb{I}}$ now operates on configurations $(M, \boldsymbol{t}, w)$ of interval terms M, interval traces $\boldsymbol{t} \in \mathbb{T}_{\mathbb{I}}$, and interval weights $w \in \mathbb{I}_{\mathbb{R}_{\ge 0} \cup \{\infty\}}$. The redexes and evaluation contexts of SPCF extend naturally to interval terms. The reduction rules are given in Fig. 3.[6]

Given a program $\vdash P : \mathbf{R}$, the reduction relation $\to_{\mathbb{I}}$ allows us to define the *interval weight* function ($\mathsf{wt}^{\mathbb{I}}_P : \mathbb{T}_{\mathbb{I}} \to \mathbb{I}_{\mathbb{R}_{\ge 0} \cup \{\infty\}}$) and *interval value* function ($\mathsf{val}^{\mathbb{I}}_P : \mathbb{T}_{\mathbb{I}} \to \mathbb{I}$) by:

$$\mathsf{wt}^{\mathbb{I}}_P(\boldsymbol{t}) := \begin{cases} w & \text{if } (P, \boldsymbol{t}, 1) \to^*_{\mathbb{I}} (\underline{I}, \langle\rangle, w) \\ [0, \infty] & \text{otherwise,} \end{cases}$$

$$\mathsf{val}^{\mathbb{I}}_P(\boldsymbol{t}) := \begin{cases} I & \text{if } (P, \boldsymbol{t}, 1) \to^*_{\mathbb{I}} (\underline{I}, \langle\rangle, w) \\ [-\infty, \infty] & \text{otherwise.} \end{cases}$$

It is not difficult to prove the following relationship between standard and interval reduction.

Lemma 3.1. *Let $\vdash P : \boldsymbol{R}$ be a program. For any interval trace $\boldsymbol{t}$ and concrete trace $\boldsymbol{s} \triangleleft \boldsymbol{t}$, we have $\mathsf{wt}_P(\boldsymbol{s}) \in \mathsf{wt}^{\mathbb{I}}_P(\boldsymbol{t})$ and $\mathsf{val}_P(\boldsymbol{s}) \in \mathsf{val}^{\mathbb{I}}_P(\boldsymbol{t})$ (provided $\mathsf{val}_P(\boldsymbol{s})$ is defined).*

3.3 Bounds from Interval Traces

Lower bounds. How can we use this interval trace semantics to obtain lower bounds on $[\![P]\!]$? We need a few definitions. Two intervals $[a_1, b_1], [a_2, b_2] \in \mathbb{I}$ are called ***almost disjoint*** if $b_1 \le a_2$ or $b_2 \le a_1$. Interval traces $\langle I_1, \dots, I_m\rangle$ and $\langle J_1, \dots, J_n\rangle \in \mathbb{T}_{\mathbb{I}}$ are called ***compatible*** if there is an index $i \in \{1, \dots, \min(m, n)\}$ such that I_i and J_i are almost disjoint. A set of interval traces is called *compatible* if its elements are pairwise compatible. We define the ***volume*** of an interval trace $\boldsymbol{t} = \langle [a_1, b_1], \dots, [a_n, b_n]\rangle$ as $\mathrm{vol}(\boldsymbol{t}) := \prod_{i=1}^{n}(b_i - a_i)$.

Let $\mathcal{T} \subseteq \mathbb{T}_{\mathbb{I}}$ be a countable and compatible set of interval traces. Define the *lower bound* on $[\![P]\!]$ by

$$\mathsf{lowerBd}^{\mathcal{T}}_P(U) := \sum_{\boldsymbol{t} \in \mathcal{T}} \mathrm{vol}(\boldsymbol{t}) \cdot (\min \mathsf{wt}^{\mathbb{I}}_P(\boldsymbol{t})) \cdot \left[\mathsf{val}^{\mathbb{I}}_P(\boldsymbol{t}) \subseteq U\right]$$

for $U \in \Sigma_{\mathbb{R}}$. That is, we sum over each interval trace in $\mathcal{T}$ whose value is *guaranteed* to be in U, weighted by its volume and the lower bound of its weight interval. Note that, in general, $\mathsf{lowerBd}^{\mathcal{T}}_P$ is not a measure, but merely a *superadditive measure*.[7]

Upper bounds. For upper bounds, we require the notion of a set of interval traces being *exhaustive*, which is easiest to express in terms of infinite traces. Let $\mathbb{T}_\infty := [0,1]^\omega$ be the set of infinite traces. Every interval trace $\boldsymbol{t}$ *covers* the set of infinite traces with a prefix contained in $\boldsymbol{t}$, i.e. $\mathit{cover}(\boldsymbol{t}) := (\!|\boldsymbol{t}|\!) \times \mathbb{T}_\infty$ (where the Cartesian product $\times$ can be viewed as trace concatenation). A countable set of (finite) interval traces $\mathcal{T} \subseteq \mathbb{T}_{\mathbb{I}}$ is called ***exhaustive*** if $\bigcup_{\boldsymbol{t} \in \mathcal{T}} \mathit{cover}(\boldsymbol{t})$ covers almost all of $\mathbb{T}_\infty$, i.e. $\mu_{\mathbb{T}_\infty}(\mathbb{T}_\infty \setminus \bigcup_{\boldsymbol{t} \in \mathcal{T}} \mathit{cover}(\boldsymbol{t})) = 0$.[8] Phrased differently, almost all concrete traces must have a finite prefix that is contained in some interval trace in $\mathcal{T}$. Therefore, the analysis in the interval semantics on $\mathcal{T}$ covers the behaviour on almost all concrete traces (in the original semantics).

Example 3.1. (i) The singleton set $\{\langle [0,1], [0,0.6]\rangle\}$ is not exhaustive as, e.g. all infinite traces $\langle r_1, r_2, \dots\rangle$ with $r_2 > 0.6$ are not covered. (ii) The set $\{\langle [0,0.6]\rangle, \langle [0.3,1]\rangle\}$ is exhaustive, but not compatible. (iii) Define $\mathcal{T}_1 := \{\langle [\frac{1}{2}, 1]^{\cdots n}, [0, \frac{1}{3}]\rangle \mid n \in \mathbb{N}\}$ and $\mathcal{T}_2 := \{\langle [\frac{1}{2}, 1]^{\cdots n}, [0, \frac{1}{2}]\rangle \mid n \in \mathbb{N}\}$ where $x^{\cdots n}$ denotes n-fold repetition of x. $\mathcal{T}_1$ is compatible but not exhaustive. For example, it doesn't cover the set $[\frac{1}{2}, 1] \times (\frac{1}{3}, \frac{1}{2}) \times \mathbb{T}_\infty$, i.e. all traces $\langle r_1, r_2, \dots\rangle$ where $r_1 \in [\frac{1}{2}, 1]$ and $r_2 \in (\frac{1}{3}, \frac{1}{2})$. $\mathcal{T}_2$ is compatible and exhaustive (the set of non-covered traces $(\frac{1}{2}, 1]^\omega$ has measure 0).

Let $\mathcal{T} \subseteq \mathbb{T}_{\mathbb{I}}$ be a countable and exhaustive set of interval traces. Define the *upper bound* on $[\![P]\!]$ by

$$\mathsf{upperBd}^{\mathcal{T}}_P(U) := \sum_{\boldsymbol{t} \in \mathcal{T}} \mathrm{vol}(\boldsymbol{t}) \cdot (\sup \mathsf{wt}^{\mathbb{I}}_P(\boldsymbol{t})) \cdot \left[\mathsf{val}^{\mathbb{I}}_P(\boldsymbol{t}) \cap U \neq \emptyset\right]$$

for $U \in \Sigma_{\mathbb{R}}$. That is, we sum over each interval trace in $\mathcal{T}$ whose value *may* be in U, weighted by its volume and the upper bound of its weight interval. Note that $\mathsf{upperBd}^{\mathcal{T}}_P$ is not a measure but only a *subadditive measure*.[9]

[6] For conditionals, the interval bound is not always precise enough to decide which branch to take, so the reduction can get stuck if $a \le 0 < b$. We could include additional rules to overapproximate the branching behaviour (see Appendix A.4). But the rules given here simplify the presentation and are sufficient to prove soundness and completeness.

[7] A *superadditive measure* μ on (Ω, Σ_Ω) is a measure, except that σ-additivity is replaced by σ-superadditivity: $\mu(\bigcup_{i \in \mathbb{N}} U_i) \ge \sum_{i \in \mathbb{N}} \mu(U_i)$ for a countable, pairwise disjoint family $(U_i)_{i \in \mathbb{N}} \in \Sigma_\Omega$.

[8] The σ-algebra on $\mathbb{T}_\infty$ is defined as the smallest σ-algebra that contains all sets $U \times \mathbb{T}_\infty$ where $U \in \Sigma_{[0,1]^n}$ for some $n \in \mathbb{N}$. The measure $\mu_{\mathbb{T}_\infty}$ is the unique measure with $\mu_{\mathbb{T}_\infty}(U \times \mathbb{T}_\infty) = \lambda_n(U)$ when $U \in \Sigma_{[0,1]^n}$.

[9] A *subadditive measure* μ on (Ω, Σ_Ω) is a measure, except that σ-additivity is replaced by σ-subadditivity: $\mu(\bigcup_{i \in \mathbb{N}} U_i) \le \sum_{i \in \mathbb{N}} \mu(U_i)$ for a countable, pairwise disjoint family $(U_i)_{i \in \mathbb{N}} \in \Sigma_\Omega$.

4 Soundness and Completeness

4.1 Soundness

We show that the two bounds described above are *sound*, in the following sense.

Theorem 4.1 (Sound lower bounds). *Let $\mathcal{T}$ be a countable and compatible set of interval traces and $\vdash P : \boldsymbol{R}$ a program. Then $\mathsf{lowerBd}_P^{\mathcal{T}} \leq [\![P]\!]$.*

Proof. For any $U \in \Sigma_{\mathbb{R}}$, we have:

$$\begin{aligned}
\mathsf{lowerBd}_P^{\mathcal{T}}(U) &= \sum_{\boldsymbol{t}\in\mathcal{T}} \mathrm{vol}(\boldsymbol{t})(\min \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}))\left[\mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t}) \subseteq U\right] \\
&= \sum_{\boldsymbol{t}\in\mathcal{T}} \int_{(\!|\boldsymbol{t}|\!)} (\min \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}))\left[\mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t}) \subseteq U\right] \mathrm{d}\boldsymbol{s} \\
&\leq \sum_{\boldsymbol{t}\in\mathcal{T}} \int_{(\!|\boldsymbol{t}|\!)} \mathrm{wt}_P(\boldsymbol{s})\left[\mathrm{val}_P(\boldsymbol{s}) \in U\right] \mathrm{d}\boldsymbol{s} \qquad (1) \\
&= \int_{\bigcup_{\boldsymbol{t}\in\mathcal{T}}(\!|\boldsymbol{t}|\!)} \mathrm{wt}_P(\boldsymbol{s})\left[\mathrm{val}_P(\boldsymbol{s}) \in U\right] \mathrm{d}\boldsymbol{s} \qquad (2) \\
&\leq \int_{\mathbb{T}} \mathrm{wt}_P(\boldsymbol{s})\left[\mathrm{val}_P(\boldsymbol{s}) \in U\right] \mathrm{d}\boldsymbol{s} = [\![P]\!](U) \qquad (3)
\end{aligned}$$

where Eq. (1) follows from Lemma 3.1, Eq. (2) from compatibility, and Eq. (3) from $\bigcup_{\boldsymbol{t}\in\mathcal{T}}(\!|\boldsymbol{t}|\!) \subseteq \mathbb{T}$. □

Theorem 4.2 (Sound upper bounds). *Let $\mathcal{T}$ be a countable and exhaustive set of interval traces and $\vdash P : \boldsymbol{R}$ a program. Then $[\![P]\!] \leq \mathsf{upperBd}_P^{\mathcal{T}}$.*

Proof sketch. The formal proof is similar to the soundness proof for the lower bound in Theorem 4.1, but needs an infinite trace semantics [16] for probabilistic programs and is given in Appendix C.1. The idea is that each interval trace $\boldsymbol{t}$ summarises all infinite traces starting with $(\!|\boldsymbol{t}|\!)$, i.e. all traces in $\mathit{cover}(\boldsymbol{t})$. Exhaustivity ensures that almost all infinite traces are "covered". □

4.2 Completeness

The soundness results for upper and lower bounds allow us to derive bounds on the denotation of a program. One would expect that a finer partition of interval traces will yield more precise bounds. In this section, we show that for a program P and an interval $I \in \mathbb{I}$, the approximations $\mathsf{lowerBd}_P^{\mathcal{T}}(I)$ and $\mathsf{upperBd}_P^{\mathcal{T}}(I)$ can in fact come arbitrarily close to $[\![P]\!](I)$ for suitable $\mathcal{T}$. However, this is only possible under certain assumptions.

Assumption 1: use of sampled values. Interval arithmetic is imprecise if the same value is used more than once: consider, for instance, $\mathsf{let}\, s = \mathsf{sample}\, \mathsf{in}\, \mathsf{if}(s - s, 0, 1)$ which deterministically evaluates to 0. However, in interval arithmetic, if x is approximated by an interval $[a, b]$ with $a < b$, the difference $x - x$ is approximated as $[a - b, b - a]$, which always contains both positive and negative values. So no non-trivial interval trace can separate the two branches.

To avoid this, we could consider a call-by-name semantics (as done in [4]) where sample values can only be used once by definition. However, many of our examples cannot be expressed in the call-by-name setting, so we instead propose a less restrictive criterion to guarantee completeness for call-by-value: we allow sample values to be used more than once, but at most once in the guard of each conditional, at most once in each score expression, and at most once in the return value. While this prohibits terms like the one above, it allows, e.g. $\mathsf{let}\, s = \mathsf{sample}\, \mathsf{in}\, \mathsf{if}(s, \underline{f}(s), \underline{g}(s))$. This sufficient condition is formalised in Appendix C.3. Most examples we encountered in the literature satisfy this assumption.

Assumption 2: primitive functions. In addition, we require mild assumptions on the primitive functions, called *boxwise continuity* and *interval separability*.

We need to be able to approximate a program's weight function by step functions in order to obtain tight bounds on its integral. A function $f : \mathbb{R}^n \to \mathbb{R}$ is ***boxwise continuous*** if it can be written as the countable union of continuous functions on boxes, i.e. if there is a countable union of pairwise almost disjoint boxes B_i such that $\bigcup B_i = \mathbb{R}^n$ and the restriction $f|_{B_i}$ is continuous for each B_i.

Furthermore, we need to approximate preimages. Formally, we say that A is a ***tight subset*** of B (written $A \Subset B$) if $A \subseteq B$ and $B \setminus A$ is a null set. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called ***interval separable*** if for every interval $[a, b] \in \mathbb{I}$, there is a countable set $\mathcal{B}$ of boxes in $\mathbb{R}^n$ that tightly approximates the preimage, i.e. $\bigcup \mathcal{B} \Subset f^{-1}([a, b])$. A sufficient condition for checking this is the following. If f is boxwise continuous and preimages of points have measure zero, then f is already interval separable (cf. Lemma C.4).

We assume the set $\mathcal{F}$ of primitive functions is closed under composition and each $f \in \mathcal{F}$ is boxwise continuous and interval separable.

The completeness theorem. Using these two assumptions, we can state completeness of our interval semantics.

Theorem 4.3 (Completeness of interval approximations). *Let $I \in \mathbb{I}$ and $\vdash P : \boldsymbol{R}$ be an almost surely terminating program satisfying the two assumptions discussed above. Then, for all $\epsilon > 0$, there is a countable set of interval traces $\mathcal{T} \subseteq \mathbb{T}_{\mathbb{I}}$ that is compatible and exhaustive such that*

$$\mathsf{upperBd}_P^{\mathcal{T}}(I) - \epsilon \leq [\![P]\!](I) \leq \mathsf{lowerBd}_P^{\mathcal{T}}(I) + \epsilon.$$

Proof sketch. We consider each branching path through the program separately. The set of relevant traces for a given path is a preimage of intervals under compositions of interval separable functions, hence can essentially be partitioned into boxes. By boxwise continuity, we can refine this partition such that the weight function is continuous on each box. To approximate the integral, we pass to a refined partition again, essentially computing Riemann sums. The latter converge

to the Riemann integral, which agrees with the Lebesgue integral under our conditions, as desired. □

For the lower bound, we can actually derive ϵ-close bounds using only finitely many interval traces:

Corollary 4.4. *Let $I \in \mathbb{I}$ and $\vdash P : \boldsymbol{R}$ be as in Theorem 4.3. There is a sequence of finite, compatible sets of interval traces $\mathcal{T}_1, \mathcal{T}_2, \ldots \subseteq \mathbb{T}_{\mathbb{I}}$ s.t. $\lim_{n\to\infty} \mathsf{lowerBd}_P^{\mathcal{T}_n}(I) = [\![P]\!](I)$.*

For the upper bound, a restriction to finite sets $\mathcal{T}$ of interval traces is, in general, not possible: if the weight function for a program is unbounded, it is also unbounded on some $\boldsymbol{t} \in \mathcal{T}$. Then $\mathsf{wt}_P^{\mathbb{I}}(\boldsymbol{t})$ is an infinite interval, implying $\mathsf{upperBd}_P^{\mathcal{T}}(I) = \infty$ (see Example C.3 for details). Despite the (theoretical) need for countably infinite many interval traces, we can, in many cases, compute finite upper bounds by making use of an interval-based static approximation, formalised as a type system in the next section.

5 Weight-aware Interval Type System

To obtain sound bounds on the denotation with only finitely many interval traces, we present an interval-based type system that can derive static bounds on a program. Crucially, our type-system is *weight-aware*: we bound not only the return value of a program but also the weight of an execution. Our analyzer GuBPI uses it for two purposes. First, it allows us to derive upper bounds even for areas of the sample space not covered with interval traces. Second, we can use our analysis to derive a *finite* (and sound) approximation of the infinite number of symbolic execution paths of a program (more details are given in Section 6). Note that the bounds inferred by our system are *interval bounds*, which allow for seamless integration with our interval trace semantics. In this section, we present the interval type system and sketch a constraint-based type inference method.

5.1 Interval Types

We define interval types by the following grammar:

$$\sigma := I \mid \sigma \to \mathcal{A} \qquad \mathcal{A} := \begin{Bmatrix} \sigma \\ I \end{Bmatrix}$$

where $I \in \mathbb{I}$ is an interval. For readers familiar with refinement types, it is easiest to view the type $\sigma = I$ as the refinement type $\{x : \mathbb{R} \mid x \in I\}$. The definition of the syntactic category $\mathcal{A}$ by mutual recursion with σ gives a bound on the weight of the execution. We call a type σ *weightless* and a type $\mathcal{A}$ *weighted*. The following examples should give some intuition about the types.

Example 5.1. Consider the example term

$$\left(\mu_x^{\varphi}.\, 5 \times x \oplus_{0.5} \mathit{sigm}(\varphi\, x + \mathsf{score}\,\mathsf{sample})\right)(4 \times \mathsf{sample})$$

where $\mathit{sigm} : \mathbb{R} \to [0,1]$ is the sigmoid function. In our type system, this term can be typed with the weighted type $\begin{Bmatrix}[0,20]\\[0,1]\end{Bmatrix}$, which indicates that any terminating execution of the term reduces to a value (a number) within $[0, 20]$ and the weight of any such execution lies within $[0, 1]$.

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \begin{Bmatrix}\sigma\\ \mathbf{1}\end{Bmatrix}} \qquad \frac{\Gamma \vdash M : \mathcal{A} \quad \mathcal{A} \sqsubseteq_{\mathcal{A}} \mathcal{B}}{\Gamma \vdash M : \mathcal{B}} \qquad \frac{\Gamma; \varphi : \sigma \to \mathcal{A}; x : \sigma \vdash M : \mathcal{A}}{\Gamma \vdash \mu_x^{\varphi}.M : \begin{Bmatrix}\sigma \to \mathcal{A}\\ \mathbf{1}\end{Bmatrix}}$$

$$\frac{\Gamma; x : \sigma \vdash M : \mathcal{A}}{\Gamma \vdash \lambda x.M : \begin{Bmatrix}\sigma \to \mathcal{A}\\ \mathbf{1}\end{Bmatrix}} \qquad \frac{\Gamma \vdash M : \begin{Bmatrix}\sigma_1 \to \begin{Bmatrix}\sigma_2\\ [e,f]\end{Bmatrix}\\ [a,b]\end{Bmatrix} \quad \Gamma \vdash N : \begin{Bmatrix}\sigma_1\\ [c,d]\end{Bmatrix}}{\Gamma \vdash MN : \begin{Bmatrix}\sigma_2\\ [a,b] \times^{\mathbb{I}} [c,d] \times^{\mathbb{I}} [e,f]\end{Bmatrix}}$$

$$\frac{}{\Gamma \vdash \underline{r} : \begin{Bmatrix}[r,r]\\ \mathbf{1}\end{Bmatrix}} \qquad \frac{\Gamma \vdash M : \begin{Bmatrix}[\_,\_]\\ [a,b]\end{Bmatrix} \quad \Gamma \vdash N : \begin{Bmatrix}\sigma\\ [c,d]\end{Bmatrix} \quad \Gamma \vdash P : \begin{Bmatrix}\sigma\\ [c,d]\end{Bmatrix}}{\Gamma \vdash \mathsf{if}(M,N,P) : \begin{Bmatrix}\sigma\\ [a,b] \times^{\mathbb{I}} [c,d]\end{Bmatrix}}$$

$$\frac{}{\Gamma \vdash \mathsf{sample} : \begin{Bmatrix}[0,1]\\ \mathbf{1}\end{Bmatrix}} \qquad \frac{\Gamma \vdash M : \begin{Bmatrix}[a,b]\\ [c,d]\end{Bmatrix}}{\Gamma \vdash \mathsf{score}(M) : \begin{Bmatrix}[a,b] \sqcap [0,\infty]\\ [c,d] \times^{\mathbb{I}} ([a,b] \sqcap [0,\infty])\end{Bmatrix}}$$

$$\frac{\Gamma \vdash M_1 : \begin{Bmatrix}[a_1,b_1]\\ [c_1,d_1]\end{Bmatrix} \quad \cdots \quad \Gamma \vdash M_{|f|} : \begin{Bmatrix}[a_{|f|},b_{|f|}]\\ [c_{|f|},d_{|f|}]\end{Bmatrix}}{\Gamma \vdash f(M_1,\ldots,M_{|f|}) : \begin{Bmatrix}f^{\mathbb{I}}([a_1,b_1],\ldots,[a_{|f|},b_{|f|}])\\ (\times^{\mathbb{I}})_{i=1}^{|f|}[c_i,d_i]\end{Bmatrix}}$$

Figure 4. Weight-aware interval type system for SPCF. We abbreviate $\mathbf{1} := [1,1]$.

Example 5.2. We consider the fixpoint subexpression of the pedestrian example in Example 1.1 which is

$$\mu_x^{\varphi}.\mathsf{if}(x, 0, \left(\lambda \mathit{step}.\mathit{step} + \varphi((x{+}\mathit{step}) \oplus_{0.5} (x{-}\mathit{step}))\right)\mathsf{sample}).$$

Using the typing rules (defined below), we can infer the type $\begin{Bmatrix}[a,b] \to \begin{Bmatrix}[0,\infty]\\ [1,1]\end{Bmatrix}\\ [1,1]\end{Bmatrix}$ for any a, b. This type indicates that any terminating execution reduces to a function value (of simple type $\mathbf{R} \to \mathbf{R}$) with weight within $[1,1]$. If this function value is then called on a value within $[a,b]$, any terminating execution reduces to a value within $[0,\infty]$ with a weight within $[1,1]$.

Subtyping. The partial order on intervals naturally extends to our type system. For base types I_1 and I_2, we define $I_1 \sqsubseteq_\sigma I_2$ just if $I_1 \sqsubseteq I_2$, where $\sqsubseteq$ is interval inclusion. We then extend this via:

$$\frac{\sigma_2 \sqsubseteq_\sigma \sigma_1 \qquad \mathcal{A}_1 \sqsubseteq_{\mathcal{A}} \mathcal{A}_2}{\sigma_1 \to \mathcal{A}_1 \sqsubseteq_\sigma \sigma_2 \to \mathcal{A}_2} \qquad \frac{\sigma_1 \sqsubseteq_\sigma \sigma_2 \qquad I_1 \sqsubseteq I_2}{\begin{Bmatrix}\sigma_1\\ I_1\end{Bmatrix} \sqsubseteq_{\mathcal{A}} \begin{Bmatrix}\sigma_2\\ I_2\end{Bmatrix}}$$

Note that in the case of weighted types, the subtyping requires not only that the weightless types be subtype-related ($\sigma_1 \sqsubseteq_\sigma \sigma_2$) but also that the weight bound be refined $I_1 \sqsubseteq I_2$. It is easy to see that both $\sqsubseteq_{\mathcal{A}}$ and $\sqsubseteq_\sigma$ are partial orders on types with the same underlying base type.

5.2 Type System

As for the interval-based semantics, we assume that every primitive operation $f : \mathbb{R}^n \to \mathbb{R}$ has an overapproximating interval abstraction $f^{\mathbb{I}} : \mathbb{I}^n \to \mathbb{I}$ (cf. Section 3.1). Interval typing judgments have the form $\Gamma \vdash M : \mathcal{A}$ where Γ is a typing context mapping variables to types σ. They are given via the rules in Fig. 4. Our system is sound in the following sense (which we here only state for first-order programs).

Theorem 5.1. *Let $\vdash P : \boldsymbol{R}$ be a simply-typed program. If $\vdash P : \left\{\begin{smallmatrix}[a,b]\\[c,d]\end{smallmatrix}\right\}$ and $(P, \boldsymbol{s}, 1) \to^* (\underline{r}, \langle\rangle, w)$ for some $\boldsymbol{s} \in \mathbb{T}$ and $r, w \in \mathbb{R}$, then $r \in [a, b]$ and $w \in [c, d]$.*

Note that the bounds derived by our type system only refer to terminating executions, i.e. they are partial correctness statements. Theorem 5.1 formalises the intuition of an interval type, i.e. every type derivation in our system bounds *both* the returned value (in typical refinement-type fashion [24]) and the weight of this derivation. Our type system also comes with a weak completeness statement: for each term, we can derive some bounds in our system.

Proposition 5.2. *Let $\vdash P : \alpha$ be a simply-typed program. There exists a weighted interval type $\mathcal{A}$ such that $\vdash P : \mathcal{A}$.*

5.3 Constraint-based Type Inference

In this section, we briefly discuss the automated type *inference* in our system, as needed in our tool GuBPI. For space reasons, we restrict ourselves to an informal overview (see Appendix D for a full account).

Given a program P, we can derive the symbolic skeleton of a type derivation (the structure of which is determined by P), where each concrete interval is replaced by a placeholder variable. The validity of a typing judgment within this skeleton can then be encoded as constraints. Crucially, as we work in the fixed interval domain and the subtyping structure $\sqsubseteq_{\mathcal{A}}$ is compositional, they are simple constraints over the placeholder variables in the abstract interval domain. Solving the resulting constraints naïvely might not terminate since the interval abstract domain is not chain-complete. Instead, we approximate the least fixpoint (where the fixpoint denotes a solution to the constraints) using *widening*, a standard approach to ensure termination of static analysis on domains with infinite chains [13, 14]. This is computationally much cheaper compared to, say, types with general first-order refinements where constraints are typically phrased as constrained Horn clauses (see e.g. [11]). This gain in efficiency is crucial to making our GuBPI tool practical.

6 Symbolic Execution and GuBPI

In this section, we describe the overall structure of our tool GuBPI (gubpi-tool.github.io), which builds upon symbolic execution. We also outline how the interval-based semantics can be accelerated for programs containing linear subexpressions.

6.1 Symbolic Execution

The starting point of our analysis is a *symbolic exploration* of the term in question [10, 28, 41]. For space reasons we only give an informal overview of the approach. A detailed and formal discussion can be found in Appendix B.

The idea of symbolic execution is to treat outcomes of sample expressions fully symbolically: each sample evaluates to a fresh variable ($\alpha_1, \alpha_2, \dots$), called *sample variable*. The result of symbolic execution is thus a symbolic value: a term consisting of sample variables and delayed primitive function applications. We postpone branching decisions and the weighting with score expressions because the value in question is symbolic. During execution, we therefore explore both branches of a conditional and keep track of the (symbolic) conditions on the sample variables that need to hold in the current branch. Similarly, we record the (symbolic) values of score expressions. Formally, our symbolic execution operates on ***symbolic configurations*** of the form $\psi = (\mathcal{M}, n, \Delta, \Xi)$ where $\mathcal{M}$ is a symbolic term containing sample variables instead of sample outcomes; $n \in \mathbb{N}$ is a natural number used to obtain fresh sample variables; Δ is a list of symbolic constraints of the form $\mathcal{V} \bowtie r$, where $\mathcal{V}$ is a symbolic value, $r \in \mathbb{R}$ and $\bowtie \in \{\leq, <, >, \geq\}$, to keep track of the conditions for the current execution path; and Ξ is a set of values that records all symbolic values of score expressions encountered along the current path. The symbolic reduction relation $\rightsquigarrow$ includes the following key rules.

$$(\mathsf{sample}, n, \Delta, \Xi) \rightsquigarrow (\alpha_{n+1}, n+1, \Delta, \Xi)$$
$$(\mathsf{if}(\mathcal{V}, \mathcal{N}, \mathcal{P})), n, \Delta, \Xi) \rightsquigarrow (\mathcal{N}, n, \Delta \cup \{\mathcal{V} \leq 0\}, \Xi)$$
$$(\mathsf{if}(\mathcal{V}, \mathcal{N}, \mathcal{P})), n, \Delta, \Xi) \rightsquigarrow (\mathcal{P}, n, \Delta \cup \{\mathcal{V} > 0\}, \Xi)$$
$$(\mathsf{score}(\mathcal{V}), n, \Delta, \Xi) \rightsquigarrow (\mathcal{V}, n, \Delta \cup \{\mathcal{V} \geq 0\}, \Xi \cup \{\mathcal{V}\})$$

That is, we replace sample outcomes with fresh sample variables (first rule), explore both paths of a conditional (second and third rule), and record all score values (fourth rule).

Example 6.1. Consider the symbolic execution of Example 1.1 where the first step moves the pedestrian towards their home (taking the right branch of $\oplus_{0.5}$) and the second step moves away from their home (the left branch of $\oplus_{0.5}$). We reach a configuration $(\mathcal{M}, 5, \Delta, \Xi)$ where $\mathcal{M}$ is

$$\mathsf{score}\big(\,\mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}\,(\alpha_2+\alpha_4+(\mu_x^{\varphi}.\mathcal{N})(3\alpha_1-\alpha_2+\alpha_4))\big); 3\alpha_1.$$

Here α_1 is the initial sample for *start*; α_2, α_4 the two samples of *step*; and α_3, α_5 the samples involved in the $\oplus_{0.5}$ operator. The fixpoint $\mu_x^{\varphi}.\mathcal{N}$ is already given in Example 5.2, $\Xi = \emptyset$ and $\Delta = \{3\alpha_1 > 0, \alpha_3 > \frac{1}{2}, 3\alpha_1 - \alpha_2 > 0, \alpha_5 \leq \frac{1}{2}\}$.

For a symbolic value $\mathcal{V}$ using sample variables $\overline{\alpha} = \alpha_1, \dots, \alpha_n$ and $\boldsymbol{s} \in [0,1]^n$, we write $\mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \in \mathbb{R}$ for the substitution of concrete values in $\boldsymbol{s}$ for the sample variables. Call a symbolic configuration of the form $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ (i.e. a configuration that has reached a symbolic value $\mathcal{V}$) a ***symbolic path***. We write $\mathit{symPaths}(\psi)$ for the (countable) set of

Algorithm 1 Symbolic Analysis in GuBPI.

1: **Input:** Program $\vdash P : \mathbf{R}$, depth limit $D \in \mathbb{N}$, and $I \in \mathbb{I}$
2: $\psi_{\text{init}} := (P, 0, \emptyset, \emptyset); S := \{(\psi_{\text{init}}, 0)\}; T := \emptyset$
3: **while** $\exists(\psi, \mathit{depth}) \in S$ **do**
4: **if** ψ has terminated **then**
5: $T := T \cup \{\psi\}; S := S \setminus \{(\psi, \mathit{depth})\}$
6: **else if** ψ contains no fixpoints or $\mathit{depth} \leq D$ **then**
7: $S := S \setminus \{(\psi, \mathit{depth})\}$
8: **for** ψ' with $\psi \rightsquigarrow \psi'$ **do**
9: $S := S \cup \{(\psi', \mathit{depth} + 1)\}$
10: **else**
11: $S := (S \setminus \{(\psi, \mathit{depth})\}) \cup \{(\mathit{approxFix}(\psi), \mathit{depth})\}$
12: **return** $\left[\sum_{\Psi \in T} [\![\Psi]\!]_{lb}(I), \sum_{\Psi \in T} [\![\Psi]\!]_{ub}(I)\right]$

symbolic paths reached when evaluating from configuration ψ. Given a symbolic path $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ and a set $U \in \Sigma_{\mathbb{R}}$, we define the denotation along Ψ, written $[\![\Psi]\!](U)$, as

$$\int_{[0,1]^n} \left[\mathcal{V}[s/\overline{\alpha}] \in U\right] \prod_{C \bowtie r \in \Delta} \left[C[s/\overline{\alpha}] \bowtie r\right] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}] \, \mathrm{d}s,$$

i.e. the integral of the product of the score weights Ξ over the traces of length n where the result value is in U and all the constraints Δ are satisfied. We can recover the denotation of a program P (as defined in Section 2) from all its symbolic paths starting from the configuration $(P, 0, \emptyset, \emptyset)$.

Theorem 6.1. *Let $\vdash P : \boldsymbol{R}$ be a program and $U \in \Sigma_{\mathbb{R}}$. Then*

$$[\![P]\!](U) = \textstyle\sum_{\Psi \in \mathit{symPaths}(P,0,\emptyset,\emptyset)} [\![\Psi]\!](U).$$

Analogously to interval SPCF (Section 3), we define ***symbolic interval terms*** as symbolic terms that may contain intervals (and similarly for symbolic interval values, symbolic interval configurations, and symbolic interval paths).

6.2 GuBPI

With symbolic execution at hand, we can outline the structure of our analysis tool GuBPI (sketched in Algorithm 1). GuBPI's analysis begins with symbolic execution of the input term to accumulate a set of symbolic *interval* paths T. If a symbolic configuration ψ has exceeded the user-defined depth limit D and still contains a fixpoint, we overapproximate all paths that extend ψ to ensure a finite set T. We accomplish this by using the interval type system (Section 5) to overapproximate all fixpoint subexpressions, thereby obtaining strongly normalizing terms (in line 11). Formally, given a symbolic configuration $\psi = (\mathcal{M}, n, \Delta, \Xi)$ we derive a typing judgment for the term $\mathcal{M}$ in the system from Section 5. Each first-order fixpoint subterm is thus given a (weightless) type of the form $[a,b] \to \left\{\begin{smallmatrix}[c,d]\\[e,f]\end{smallmatrix}\right\}$. We replace this fixpoint with $\lambda\_.\big(\text{score}([e,f])\,;\,[c,d]\big)$. We denote this operation on configurations by $\mathit{approxFix}(\psi)$ (it extends to higher-order fixpoints as expected). Note that $\mathit{approxFix}(\psi)$ is a symbolic *interval* configuration.

Example 6.2. Consider the symbolic configuration given in Example 6.1. As in Example 5.2 we infer the type of $\mu_x^{\varphi}.\mathcal{N}$ to be $[-1,4] \to \left\{\begin{smallmatrix}[0,\infty]\\[1,1]\end{smallmatrix}\right\}$. The function *approxFix* replaces $\mu_x^{\varphi}.\mathcal{N}$ with $\lambda\_.\text{score}([1,1]); [0,\infty]$. By evaluating the resulting symbolic interval configuration further, we obtain the symbolic interval path $(3\alpha_1, 5, \Delta, \Xi)$ where Δ is as in Example 6.1 and $\Xi = \{\text{pdf}_{\text{Normal}(1.1,0.1)}(\alpha_2 + \alpha_4 + [0,\infty])\}$. Note that, in general, the further evaluation of $\mathit{approxFix}(\psi)$ can result in multiple symbolic interval paths.

Afterwards, we're left with a finite set T of symbolic interval paths. Due to the presence of intervals, we cannot define a denotation of such paths directly and instead define lower and upper bounds. For a symbolic interval value $\mathcal{V}$ that contains *no* sample variables, we define $\ulcorner \mathcal{V} \urcorner \subseteq \mathbb{R}$ as the set of all values that the term can evaluate to by replacing every interval $[a,b]$ with some value $r \in [a,b]$. Given a symbolic interval path $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ and $U \in \Sigma_{\mathbb{R}}$ we define $[\![\Psi]\!]_{lb}(U)$ by considering only those concrete traces that fulfill the constraints in Ψ for *all* concrete values in the intervals and take the infimum over all scoring expressions:

$$\int \left[\ulcorner \mathcal{V}[s/\overline{\alpha}] \urcorner \subseteq U\right] \prod_{C \bowtie r \in \Delta} \left[\forall t \in \ulcorner C[s/\overline{\alpha}] \urcorner . t \bowtie r\right] \prod_{\mathcal{W} \in \Xi} \inf \ulcorner \mathcal{W}[s/\overline{\alpha}] \urcorner \, \mathrm{d}s.$$

Similarly, we define $[\![\Psi]\!]_{ub}(U)$ as

$$\int \left[\ulcorner \mathcal{V}[s/\overline{\alpha}] \urcorner \cap U \neq \emptyset\right] \prod_{C \bowtie r \in \Delta} \left[\exists t \in \ulcorner C[s/\overline{\alpha}] \urcorner . t \bowtie r\right] \prod_{\mathcal{W} \in \Xi} \sup \ulcorner \mathcal{W}[s/\overline{\alpha}] \urcorner \, \mathrm{d}s.$$

We note that, if Ψ contains no intervals, $[\![\Psi]\!]$ is defined and we have $[\![\Psi]\!]_{lb} = [\![\Psi]\!]_{ub} = [\![\Psi]\!]$. We can now state the following theorem that formalises the observation that $\mathit{approxFix}(\psi)$ soundly approximates all symbolic paths that result from ψ.

Theorem 6.2. *Let ψ be a symbolic (interval-free) configuration and $U \in \Sigma_{\mathbb{R}}$. Define $A = \mathit{symPaths}(\psi)$ as the (possibly infinite) set of all symbolic paths reached when evaluating ψ and $B = \mathit{symPaths}(\mathit{approxFix}(\psi))$ as the (finite) set of symbolic* interval *paths reached when evaluating $\mathit{approxFix}(\psi)$. Then*

$$\textstyle\sum_{\Psi \in B} [\![\Psi]\!]_{lb}(U) \leq \sum_{\Psi \in A} [\![\Psi]\!](U) \leq \sum_{\Psi \in B} [\![\Psi]\!]_{ub}(U).$$

The correctness of Algorithm 1 is then a direct consequence of Theorems 6.1 and 6.2:

Corollary 6.3. *Let T be the set of symbolic interval paths computed when at line 12 of Algorithm 1 and $U \in \Sigma_{\mathbb{R}}$. Then*

$$\textstyle\sum_{\Psi \in T} [\![\Psi]\!]_{lb}(U) \leq [\![P]\!](U) \leq \sum_{\Psi \in T} [\![\Psi]\!]_{ub}(U).$$

What remains is to compute (lower bounds on) $[\![\Psi]\!]_{lb}(I)$ and (upper bounds on) $[\![\Psi]\!]_{ub}(I)$ for a symbolic interval path $\Psi \in T$ and $I \in \mathbb{I}$. We first present the standard interval trace semantics (Section 6.3) and then a more efficient analysis for the case that Ψ contains only linear functions (Section 6.4).

6.3 Standard Interval Trace Semantics

For any symbolic interval path Ψ, we can employ the semantics as introduced in Section 3. Instead of applying the

analysis to the entire program, we can restrict to the current path Ψ (intuitively, by adding a score(0) to all other program paths). The interval traces split the domain of each sample variable in Ψ into intervals. It is easy to see that for any compatible and exhaustive set of interval traces $\mathcal{T}$, we have $\text{lowerBd}_\Psi^{\mathcal{T}}(U) \leq [\![\Psi]\!]_{lb}(U)$ and $[\![\Psi]\!]_{ub}(U) \leq \text{upperBd}_\Psi^{\mathcal{T}}(U)$ for any $U \in \Sigma_{\mathbb{R}}$ (see Theorem 4.1 and 4.2). Applying the interval-based semantics on the level of symbolic interval paths maintains the attractive features, namely soundness and completeness (relative to the current path) of the semantics. Note that the intervals occurring in Ψ seamlessly integrate with our interval-based semantics.

6.4 Linear Interval Trace Semantics

In case the score values and the guards of all conditionals are linear, we can improve and speed up the interval-based semantics.

Assume all symbolic interval values appearing in Ψ are interval linear functions of $\overline{\alpha}$ (i.e. functions $\overline{\alpha} \mapsto \mathbf{w}^\intercal \overline{\alpha} +^{\mathbb{I}} [a,b]$ for some $\mathbf{w} \in \mathbb{R}^n$ and $[a,b] \in \mathbb{I}$). We assume, for now, that each symbolic value $\mathcal{W} \in \Xi$ denotes an interval-free linear function (i.e. a function $\overline{\alpha} \mapsto \mathbf{w}^\intercal \overline{\alpha} + r$). Fix some interval $I \in \mathbb{I}$. We first note that both $[\![\Psi]\!]_{lb}(I)$ and $[\![\Psi]\!]_{ub}(I)$ are the integral of a polynomial over a convex polytope: define

$$\mathfrak{P}_{lb} := \left\{ s \in \mathbb{R}^n \mid \ulcorner \mathcal{V}[s/\overline{\alpha}] \urcorner \subseteq I \wedge \bigwedge_{C \bowtie r \in \Delta} \forall t \in \ulcorner C[s/\overline{\alpha}] \urcorner . t \bowtie r \right\}$$

which is a polytope.[10] Then $[\![\Psi]\!]_{lb}(I)$ is the integral of the polynomial $\overline{\alpha} \mapsto \prod_{\mathcal{W} \in \Xi} \mathcal{W}$ over $\mathfrak{P}_{lb}$. The definition of $\mathfrak{P}_{ub}$ (as the region of integration for $[\![\Psi]\!]_{ub}(I)$) is similar. Such integrals can be computed exactly [2], e.g. with the LattE tool [20]. Unfortunately, our experiments showed that this does not scale to interesting probabilistic programs.

Instead, we derive guaranteed bounds on the denotation by means of iterated volume computations. This has the additional benefit that we can handle non-uniform samples and non-liner expressions in Ξ. We follow an approach similar to that of the interval-based semantics in Section 4 but do not split/bound *individual sample variables* and instead directly bound *linear functions* over the sample variables. Let $\Xi = \{\mathcal{W}_1, \dots, \mathcal{W}_k\}$. We define a ***box*** (by abuse of language) as an element $\boldsymbol{t} = \langle [a_1,b_1], \dots, [a_k,b_k] \rangle$, where $[a_i,b_i]$ gives a bound on $\mathcal{W}_i$.[11] We define $lb(\boldsymbol{t}) := \prod_{i=1}^k a_i$ and $ub(\boldsymbol{t}) := \prod_{i=1}^k b_i$. The box $\boldsymbol{t}$ naturally defines a subset of $\mathfrak{P}_{lb}$ given by $\mathfrak{P}_{lb}^{\boldsymbol{t}} = \left\{ s \in \mathfrak{P}_{lb} \mid \bigwedge_{i=1}^k \mathcal{W}_i[s/\overline{\alpha}] \in [a_i,b_i] \right\}$. Then $\mathfrak{P}_{lb}^{\boldsymbol{t}}$ is again a polytope and we write $\text{vol}(\mathfrak{P}_{lb}^{\boldsymbol{t}})$ for its volume. The definition of $\mathfrak{P}_{ub}^{\boldsymbol{t}}$ and $\text{vol}(\mathfrak{P}_{ub}^{\boldsymbol{t}})$ is analogous. As for interval traces, we call two boxes $\boldsymbol{t}_1, \boldsymbol{t}_2$ *compatible* if the intervals are almost disjoint in at least one position. A set of boxes B is *compatible* if its elements are pairwise compatible and *exhaustive* if $\bigcup_{\boldsymbol{t} \in B} \mathfrak{P}_{lb}^{\boldsymbol{t}} = \mathfrak{P}_{lb}$ and $\bigcup_{\boldsymbol{t} \in B} \mathfrak{P}_{ub}^{\boldsymbol{t}} = \mathfrak{P}_{ub}$ (cf. Section 3.3).

Proposition 6.4. *Let B be a compatible and exhaustive set of boxes. Then $\sum_{\boldsymbol{t} \in B} lb(\boldsymbol{t}) \cdot \text{vol}\left(\mathfrak{P}_{lb}^{\boldsymbol{t}}\right) \leq [\![\Psi]\!]_{lb}(I)$ and $[\![\Psi]\!]_{ub}(I) \leq \sum_{\boldsymbol{t} \in B} ub(\boldsymbol{t}) \cdot \text{vol}\left(\mathfrak{P}_{ub}^{\boldsymbol{t}}\right)$.*

As in the standard interval semantics, a finer partition into boxes yields more precise bounds. While the volume computation involved in Proposition 6.4 is expensive [22], the number of splits on the linear functions is much smaller than that needed in the standard interval-based semantics. Our experiments empirically demonstrate that the direct splitting of linear functions (if applicable) is usually superior to the standard splitting. In GuBPI we compute a set of exhaustive boxes by first computing a lower and upper bound on each $\mathcal{W}_i \in \Xi$ over $\mathfrak{P}_{lb}$ (or $\mathfrak{P}_{ub}$) by solving a linear program (LP) and splitting the resulting range in evenly sized chunks.

Beyond uniform samples and linear scores. We can extend our linear optimization to non-uniform samples and arbitrary symbolic values in Ξ. We accomplish the former by *combining* the optimised semantics (where we bound linear expressions) with the standard interval-trace semantics (where we bound individual sample variables). For the latter, we can identify linear sub-expressions of the expressions in Ξ, use boxes to bound each linear sub-expression, and use interval arithmetic to infer bounds on the entire expression from bounds on its linear sub-expressions. More details can be found in Appendix E.1.

Example 6.3. Consider the path $\Psi = (3\alpha_1, 5, \Delta, \Xi)$ derived in Example 6.2. We use 1-dimensional boxes to bound $\alpha_2 + \alpha_4$ (the single linear sub-expression of the symbolic values in Ξ). To obtain a lower bound on $[\![\Psi]\!]_{lb}(I)$, we sum over all boxes $\boldsymbol{t} = \langle [a_1,b_1] \rangle$ and take the product of $\text{vol}\left(\mathfrak{P}_{lb}^{\boldsymbol{t}}\right)$ with the lower interval bound of $\text{pdf}_{\text{Normal}(1.1,0.1)}([a_1,b_1] + [0,\infty])$ (evaluated in interval arithmetic). Analogously, for the upper bound we take the product of $\text{vol}\left(\mathfrak{P}_{ub}^{\boldsymbol{t}}\right)$ with the upper interval bound of $\text{pdf}_{\text{Normal}(1.1,0.1)}([a_1,b_1] + [0,\infty])$.

7 Practical Evaluation

We have implemented our semantics in the prototype GuBPI (gubpi-tool.github.io), written in F#. In cases where we apply the linear optimisation of our semantics, we use Vinci [8] to discharge volume computations of convex polytopes. We set out to answer the following questions:

1. How does GuBPI perform on instances that could already be solved (e.g. by PSI [26])?
2. Is GuBPI able to infer useful bounds on recursive programs that could not be handled rigorously before?

7.1 Probability Estimation

We collected a suite of 18 benchmarks from [56]. Each benchmark consists of a program P and a query ϕ over the variables

[10] For example, if C denotes the function $\overline{\alpha} \mapsto \mathbf{w}^\intercal \overline{\alpha} + [a,b]$ we can transform a constraint $\forall t \in \ulcorner C[s/\overline{\alpha}] \urcorner . t \leq r$ into the linear constraint $\mathbf{w}^\intercal \overline{\alpha} + b \leq r$.

[11] Note the similarity to the interval trace semantics. While the ith position in an interval trace bounds the value of the ith sample variable, the ith entry of a box bounds the ith score value.

Table 1. Evaluation on selected benchmarks from [56]. We give the times (in seconds) and bounds computed by [56] and GuBPI. Details on the exact queries (the Q column) can be found in Table 4 in the appendix.

		Tool from [56]		GuBPI	
Program	**Q**	t	**Result**	t	**Result**
tug-of-war	Q1	1.29	[0.6126, 0.6227]	0.72	[0.6134, 0.6135]
tug-of-war	Q2	1.09	[0.5973, 0.6266]	0.79	[0.6134, 0.6135]
beauquier-3	Q1	1.15	[0.5000, 0.5261]	22.5	[0.4999, 0.5001]
ex-book-s	Q1	8.48	[0.6633, 0.7234]	6.52	[0.7417, 0.7418]
ex-book-s	Q2*	10.3	[0.3365, 0.3848]	8.01	[0.4137, 0.4138]
ex-cart	Q1	2.41	[0.8980, 1.1573]	67.3	[0.9999, 1.0001]
ex-cart	Q2	2.40	[0.8897, 1.1573]	68.5	[0.9999, 1.0001]
ex-cart	Q3	0.15	[0.0000, 0.1150]	67.4	[0.0000, 0.0001]
ex-ckd-epi-s	Q1*	0.15	[0.5515, 0.5632]	0.86	[0.0003, 0.0004]
ex-ckd-epi-s	Q2*	0.08	[0.3019, 0.3149]	0.84	[0.0003, 0.0004]
ex-fig6	Q1	1.31	[0.1619, 0.7956]	21.2	[0.1899, 0.1903]
ex-fig6	Q2	1.80	[0.2916, 1.0571]	21.4	[0.3705, 0.3720]
ex-fig6	Q3	1.51	[0.4314, 2.0155]	24.7	[0.7438, 0.7668]
ex-fig6	Q4	3.96	[0.4400, 3.0956]	27.4	[0.8682, 0.9666]
ex-fig7	Q1	0.04	[0.9921, 1.0000]	0.18	[0.9980, 0.9981]
example4	Q1	0.02	[0.1910, 0.1966]	0.31	[0.1918, 0.1919]
example5	Q1	0.06	[0.4478, 0.4708]	0.27	[0.4540, 0.4541]
herman-3	Q1	0.47	[0.3750, 0.4091]	124	[0.3749, 0.3751]

of P. We bound the probability of the event described by ϕ using the tool from [56] and GuBPI (Table 1). While our tool is generally slower than the one in [56], the completion times are still reasonable. Moreover, in many cases, the bounds returned by GuBPI are tighter than those of [56]. In addition, for benchmarks marked with a $\star$, the two pairs of bounds contradict each other.[12] We should also remark that GuBPI cannot handle all benchmarks proposed in [56] because the heavy use of conditionals causes our precise symbolic analysis to suffer from the well-documented path explosion problem [6, 9, 30]. Perhaps unsurprisingly, [56] can handle such examples much better, as one of their core contributions is a stochastic method to reduce the number of paths considered (see Section 8). Also note that [56] is restricted to uniform samples, linear guards and score-free programs, whereas we tackle a much more general problem.

7.2 Exact Inference

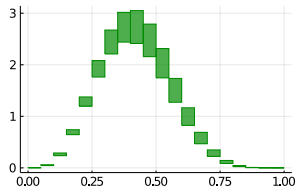
To evaluate our tool on instances that can be solved exactly, we compared it with PSI [26, 27], a symbolic solver which can, in certain cases, compute a closed-form solution of the posterior. We note that whenever exact inference is possible, exact solutions will always be superior to mere bounds and, due to the overhead of our semantics, will often be found faster. Because of the different output formats (i.e. exact results vs. bounds), a direct comparison between exact methods and GuBPI is challenging. As a consistency check, we collected benchmarks from the PSI repository where the output domain is finite and GuBPI can therefore compute *exact* results (tight bounds). They agree with PSI in all cases, which includes 8 of the 21 benchmarks from [26]. We report the computation times in Table 2.

We then considered examples where GuBPI computes non-tight bounds. For space reasons, we can only include a selection of examples in this paper. The bounds computed by GuBPI and a short description of each example are shown in Fig. 5. We can see that, despite the relatively loose bounds, they are still useful and provide the user with a rough—and most importantly, *guaranteed to be correct*—idea of the denotation.

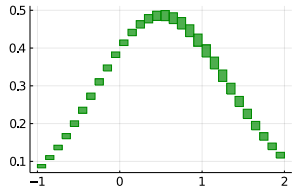
[12] A stochastic simulation using 10^6 samples in Anglican [61] yielded results that fall within GuBPI's bounds but violate those computed by [56].

Table 2. Probabilistic programs with discrete domains from PSI [26]. The times for PSI and GuBPI are given in seconds.

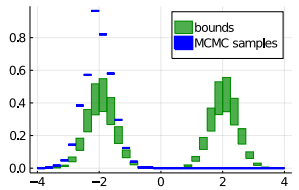
Instance	t_{PSI}	t_{GuBPI}	**Instance**	t_{PSI}	t_{GuBPI}
burglarAlarm	0.06	0.21	coins	0.04	0.18
twoCoins	0.04	0.21	ev-model1	0.04	0.21
grass	0.06	0.37	ev-model2	0.04	0.20
noisyOr	0.14	0.72	murderMystery	0.04	0.19
bertrand	0.04	0.22	coinBiasSmall	0.13	1.92
coinPattern	0.04	0.19	gossip	0.08	0.24



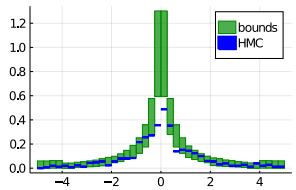
(a) `coinBias` example from [26]. The program samples a beta prior on the bias of a coin and observes repeated coin flips (26 seconds).



(b) `max` example from [26]. The program compute the maximum of two i.i.d. normal samples (31 seconds).

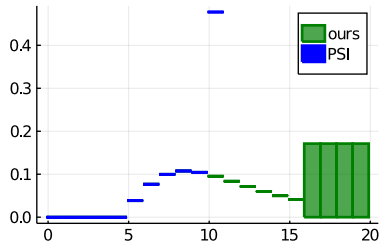


(c) Binary Gaussian Mixture Model from [65] (39 seconds). MCMC methods usually find only one mode.

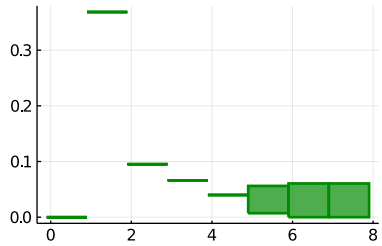


(d) Neal's funnel from [34, 48] (2.8 seconds). HMC misses some probability mass around 0.

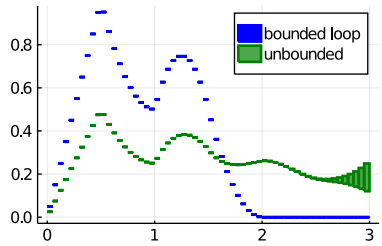
Figure 5. Guaranteed Bounds computed by GuBPI for a selection of non-recursive models from [26, 27, 48, 65].



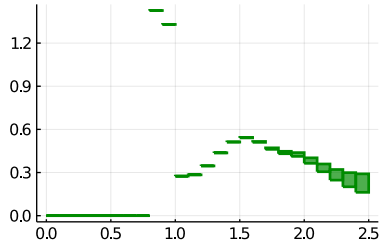
(a) `cav-example-7`. Program taken from the PSI repository. PSI bounds the depth resulting in a spike at 10, whereas GuBPI can compute bounds on the denotation of the unbounded program (112 seconds).



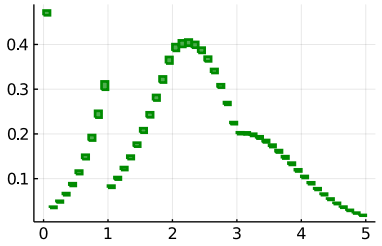
(b) `cav-example-5`. Program taken from the PSI repository. PSI cannot handle this program due to the unbounded loops (192 seconds).



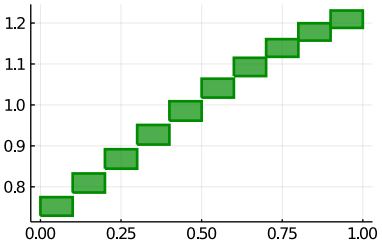
(c) `add_uniform_with_counter_large`. Program taken from the PSI repository. GuBPI can handle the unbounded loop, whereas PSI unrolls to a fixed depth (21 seconds).



(d) `random-box-walk`. The program models the cumulative distance traveled by a biased random walk. If a uniformly sampled step s has length less than $\frac{1}{2}$, we move s to the left, otherwise s to the right. The walk stops when it crosses a threshold (167 seconds).



(e) `growing-walk`. The program models a geometric random walk where (with increasing distance) the step size of the walk is increased. The cumulative distance is observed from a normal distribution centered at 3 (67 seconds).



(f) `param-estimation-recursive`. We sample a uniform prior p and (in each step) travel to the left with probability p and to the right with probability $(1-p)$. We observe the walk to come to a halt at location 1 (observed from a normal) and wish to find the posterior on p (162 seconds).

Figure 6. Guaranteed bounds computed by GuBPI for a selection of recursive models.

The success of exact solvers such as PSI depends on the underlying symbolic solver (and the optimisations implemented). Consequently, there are instances where the symbolic solver cannot compute a closed-form (integral-free) solution. Conversely, while our method is (theoretically) applicable to a very broad class of programs, there exist programs where the symbolic solver finds solutions but the analysis in GuBPI becomes infeasible due to the large number of interval traces required.

7.3 Recursive Models

We also evaluated our tool on complex models that *cannot* be handled by any of the existing methods. For space reasons, we only give an overview of some examples. Unexpectedly, we found recursive models in the PSI repository: there are examples that are created by unrolling loops to a fixed depth. This fixed unrolling changes the posterior of the model. Using our method we can handle those examples *without* bounding the loop. Three such examples are shown in Figs. 6a to 6c. In Fig. 6a, PSI bounds the iterations resulting in a spike at 10 (the unrolling bound). For Fig. 6b, PSI does not provide any solution whereas GuBPI provides useful bounds. For Fig. 6c, PSI bounds the loop to compute results (displayed in blue) whereas GuBPI computes the green bounds on the unbounded program. It is obvious that the bounds differ significantly, highlighting the impact that unrolling to a fixed depth can have on the denotation. This again strengthens the claim that rigorous methods that can handle unbounded loops/recursion are needed. There also exist unbounded discrete examples where PSI computes results for the bounded version that differ from the denotation of the unbounded program. Figs. 6d to 6f depict further recursive examples (alongside a small description).

Lastly, as a *very* challenging example, we consider the pedestrian example (Example 1.1) again. The bounds computed by GuBPI are given in Fig. 7 together with the two stochastic results from Fig. 1. The bounds are clearly precise enough to rule out the HMC samples. Since this example has infinite expected running time, it is very challenging and GuBPI takes about 1.5h (84 min).[13] In fact, guaranteed bounds are the only method to recognise the wrong samples with certainty (see the next section for statistical methods).

[13] While the running time seems high, we note that Pyro HMC took about an hour to generate 10^4 samples and produce the (wrong) histogram. Diagnostic methods like simulation-based calibration took even longer (>30h) and delivered inconclusive results (see Section 7.4 for details).

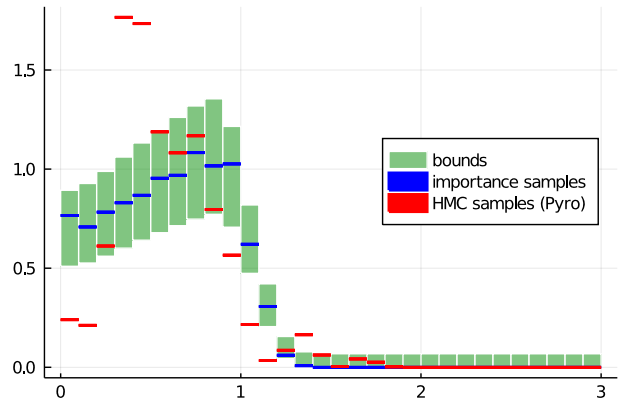


Figure 7. Bounds for the pedestrian example (Example 1.1).

Table 3. Running times of GuBPI and SBC for (Pyro's) HMC. Times are given in seconds (s) and hours (h).

Instance	t_{GuBPI}	t_{SBC}
Binary GMM (1-dimensional) (Fig. 5c)	39s	1h
Binary GMM (2-dimensional)	4h	1.5h
Pedestrian Example (Fig. 7)	1.5h	>300h

7.4 Comparison with Statistical Validation

A general approach to validating inference algorithms for a generative Bayesian model is *simulation-based calibration* (SBC) [12, 60]. SBC draws a sample θ from the prior distribution of the parameters, generates data y for these parameters, and runs the inference algorithm to produce posterior samples $\theta_1, \ldots, \theta_L$ given y. If the posterior samples follow the true posterior distribution, the rank statistic of the prior sample θ relative to the posterior samples will be uniformly distributed. If the empirical distribution of the rank statistic after many such simulations is non-uniform, this indicates a problem with the inference. While SBC is very general, it is computationally expensive because it performs inference in every simulation. Moreover, as SBC is a stochastic validation approach, any fixed number of samples may fail to diagnose inference errors that only occur on a very low probability region.

We compare the running times of GuBPI and SBC for three examples where Pyro's HMC yields wrong results (Table 3). Running SBC on the pedestrian example (with a reduced sample size and using the parameters recommended in [60]) took 32 hours and was still inconclusive because of strong autocorrelation. Reducing the latter via thinning requires more samples, and would increase the running time to >300 hours. Similarly, GuBPI diagnoses the problem with the mixture model in Fig. 5c in significantly less time than SBC. However, for higher-dimensional versions of this mixture model, SBC clearly outperforms GuBPI. We give a more detailed discussion of SBC for these examples in Appendix F.3.

7.5 Limitations and Future Improvements

The theoretical foundations of our interval-based semantics ensure that GuBPI is applicable to a very broad class of programs (cf. Section 4). In practice, as usual for exact methods, GuBPI does not handle all examples equally well.

Firstly, as we already saw in Section 7.1, the symbolic execution—which forms the entry point of the analysis—suffers from path explosion. On some extreme loop/recursion-free programs (such as example-ckd-epi from [56]), our tool cannot compute all (finitely many) symbolic paths in reasonable time, let alone analyse them in our semantics. Extending the approach from [56], to sample representative program paths (in the presence of conditioning), is an interesting future direction that we can combine with the rigorous analysis provided by our interval type system.

Secondly, our interval-based semantics imposes bounds on each sampled variable and thus scales exponentially with the dimension of the model; this is amplified in the case where the optimised semantics (Section 6.4) is not applicable. It would be interesting to explore whether this can be alleviated using different trace splitting techniques.

Lastly, the bounds inferred by our interval type system take the form of a single interval with no information about the exact distribution on that interval. For example, the most precise bound derivable for the term $\mu^{\varphi}_{x}.x \oplus \left[\varphi(x+\text{sample}) \oplus \varphi(x-\text{sample})\right]$ is $_{[a,b]} \to \left\{ {}^{[-\infty,\infty]}_{[1,1]} \right\}$ for any a, b. After unrolling to a fixed depth, the approximation of the paths not terminating within the fixed depth is therefore imprecise. For future work, it would be interesting to improve the bounds in our type system to provide more information about the distribution by means of rigorous approximations of the denotation of the fixpoint in question (i.e. a probabilistic summary of the fixpoint [46, 50, 63]).

8 Related Work

Interval trace semantics and Interval SPCF. Our interval trace semantics to compute bounds on the denotation is similar to the semantics introduced by Beutner and Ong [4], who study an interval approximation to obtain *lower* bounds on the termination probability. By contrast, we study the more challenging problem of bounding the program denotation which requires us to track the weight of an execution, and to prove that the denotation approximates a Lebesgue integral, which requires novel proof ideas. Moreover, whereas the termination probability of a program is always upper bounded by 1, here we derive both lower and *upper* bounds.

Probability estimation. Sankaranarayanan et al. [56] introduced a static analysis framework to infer bounds on a class of definable events in (*score-free*) probabilistic programs. The idea of their approach is that if we find a finite set $\mathcal{T}$ of symbolic traces with cumulative probability at least $1 - c$, and a given event ϕ occurs with probability at most b on the

traces in $\mathcal{T}$, then ϕ occurs with probability at most $b + c$ on the entire program. In the presence of conditioning, the problem becomes vastly more difficult, as the aggregate weight on the unexplored paths can be unbounded, giving ∞ as the only derivable upper bound (and therefore also ∞ as the best upper bound on the normalising constant). In order to infer guaranteed bounds, it is necessary to analyse *all* paths in the program, which we accomplish via static analysis and in particular our interval type system. The approach from [56] was extended by Albarghouthi et al. [1] to compute the probability of events defined by arbitrary SMT constraints but is restricted to score-free and non-recursive programs. Our interval-based approach, which may be seen as a variant of theirs, is founded on a complete semantics (Theorem 4.3), can handle recursive program with (soft) scoring, and is applicable to a broad class of primitive functions.

Note that we consider programs with *soft* conditioning in which scoring cannot be reduced to volume computation directly.[14] Intuitively, soft conditioning performs a (global) re-weighting of the set of traces, which cannot be captured by (local) volume computations. In our interval trace semantics, we instead track an approximation of the weight along each interval trace.

Exact inference. There are numerous approaches to inferring the exact denotation of a probabilistic program. Holtzen et al. [38] introduced an inference method to efficiently compute the denotation of programs with discrete distributions. By exploiting program structure to factorise inference, their system Dice can perform exact inference on programs with hundreds of thousands of random variables. Gehr et al. [26] introduced PSI, an exact inference system that uses symbolic manipulation and integration. A later extension, λPSI [27], adds support for higher-order functions and nested inference. The PPL Hakaru [47] supports a variety of inference algorithms on programs with both discrete and continuous distributions. Using program transformation and partial evaluation, Hakaru can perform exact inference via symbolic disintegration [57] on a limited class of programs. Saad et al. [55] introduced SPPL, a system that can compute exact answers to a range of probabilistic inference queries, by translating a restricted class of programs to sum-product expressions, which are highly effective representations for inference.

While exact results are obviously desirable, this kind of inference only works for a restricted family of programs: none of the above exact inference systems allow (unbounded) recursion. Unlike our tool, they are therefore unable to handle, for instance, the challenging Example 1.1 or the programs in Fig. 6.

[14] For programs including only hard-conditioning (i.e. scoring is only possible with 0 or 1), the posterior probability of an event φ can be computed by dividing the probability of all traces with weight 1 on which φ holds by the probability of all traces with weight 1.

Abstract interpretation. There are various approaches to probabilistic abstract interpretation, so we can only discuss a selection. Monniaux [44, 45] developed an abstract domain for (score-free) probabilistic programs given by a weighted sum of abstract regions. Smith [58] considered truncated normal distributions as an abstract domain and developed analyses restricted to score-free programs with only linear expressions. Extending both approaches to support soft conditioning is non-trivial as it requires the computation of integrals on the abstract regions. In our interval-based semantics, we abstract the concrete traces (by means of interval traces) and not the denotation. This allows us to derive bounds on the weight along the abstracted paths.

Huang et al. [39] discretise the domain of continuous samples into interval cubes and derive posterior approximations on each cube. The resulting approximation converges to the true posterior (similarly to approximate/stochastic methods) but does not provide exact/guaranteed bounds and is not applicable to recursive programs.

Refinement types. Our interval type system (Section 5) may be viewed as a type system that refines not just the value of an expression but also its weight [24]. To our knowledge, no existing type refinement system can bound the weight of program executions. Moreover, the seamless integration with our interval trace semantics by design allows for much cheaper type inference, without resorting to an SMT or Horn constraint solver. This is a crucial advantage since a typical GuBPI execution queries the analysis numerous times.

Stochastic methods. A general approach to validating inference algorithms for a generative Bayesian model is *simulation-based calibration* (SBC) [12, 60], discussed in Section 7.4. Grosse et al. [36] introduced a method to estimate the log marginal likelihood of a model by constructing stochastic lower/upper bounds. They show that the true value can be sandwiched between these two stochastic bounds with high probability. In closely related work [17, 35], this was applied to measure the accuracy of approximate probabilistic inference algorithms on a specified dataset. By contrast, our bounds are non-stochastic and our method is applicable to arbitrary programs of a universal PPL.

9 Conclusion

We have studied the problem of inferring guaranteed bounds on the posterior of programs written in a universal PPL. Our work is based on the interval trace semantics, and our weight-aware interval type system gives rise to a tool that can infer useful bounds on the posterior of interesting recursive programs. This is a capability beyond the reach of existing methods, such as exact inference. As a method of Bayesian inference for statistical probabilistic programs, we can view our framework as occupying a useful middle ground between approximate stochastic inference and exact inference.

References

[1] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: probabilistic verification of program fairness. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017). https://doi.org/10.1145/3133904

[2] Velleda Baldoni, Nicole Berline, Jesus De Loera, Matthias Köppe, and Michèle Vergne. 2011. How to integrate a polynomial over a simplex. *Math. Comp.* 80, 273 (2011). https://doi.org/10.1090/S0025-5718-2010-02378-6

[3] Atilim Günes Baydin, Lei Shao, Wahid Bhimji, Lukas Alexander Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip H. S. Torr, Victor W. Lee, Kyle Cranmer, Prabhat, and Frank Wood. 2019. Etalumis: bringing probabilistic programming to scientific simulators at scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019*. ACM. https://doi.org/10.1145/3295500.3356180

[4] Raven Beutner and Luke Ong. 2021. On probabilistic termination of functional programs with continuous distributions. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021*. ACM. https://doi.org/10.1145/3453483.3454111

[5] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20 (2019). http://jmlr.org/papers/v20/18-403.html

[6] Peter Boonstoppel, Cristian Cadar, and Dawson R. Engler. 2008. RWset: Attacking Path Explosion in Constraint-Based Test Generation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008*. Springer. https://doi.org/10.1007/978-3-540-78800-3_27

[7] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *ACM SIGPLAN International Conference on Functional Programming, ICFP 2016*. ACM. https://doi.org/10.1145/2951913.2951942

[8] Benno Büeler, Andreas Enge, and Komei Fukuda. 2000. Exact Volume Computation for Polytopes: A Practical Study. In *Polytopes—combinatorics and computation*. DMV Sem., Vol. 29. Birkhäuser, Basel. https://doi.org/10.1007/978-3-0348-8438-9_6

[9] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. 2008. EXE: Automatically Generating Inputs of Death. *ACM Transactions on Information and System Security* 12, 2 (2008). https://doi.org/10.1145/1455518.1455522

[10] Arun Tejasvi Chaganty, Aditya V. Nori, and Sriram K. Rajamani. 2013. Efficiently Sampling Probabilistic Programs via Program Analysis. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2013 (JMLR, Vol. 31)*. http://proceedings.mlr.press/v31/chaganty13a.html

[11] Adrien Champion, Tomoya Chiba, Naoki Kobayashi, and Ryosuke Sato. 2020. ICE-Based Refinement Type Discovery for Higher-Order Functional Programs. *Journal of Automated Reasoning* 64, 7 (2020). https://doi.org/10.1007/s10817-020-09571-y

[12] Samantha R Cook, Andrew Gelman, and Donald B Rubin. 2006. Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics* 15, 3 (2006). https://doi.org/10.1198/106186006X136976

[13] Patrick Cousot and Radhia Cousot. 1976. Static determination of dynamic properties of programs. In *International Symposium on Programming*. Dunod.

[14] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *ACM Symposium on Principles of Programming Languages, POPL 1977*. ACM. https://doi.org/10.1145/512950.512973

[15] Patrick Cousot and Radhia Cousot. 2002. Modular Static Program Analysis. In *International Conference on Compiler Construction, CC 2002 (LNCS, Vol. 2304)*. Springer. https://doi.org/10.1007/3-540-45937-5_13

[16] Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *European Symposium on Programming, ESOP 2017 (LNCS, Vol. 10201)*. Springer. https://doi.org/10.1007/978-3-662-54434-1_14

[17] Marco F. Cusumano-Towner and Vikash K. Mansinghka. 2017. AIDE: An algorithm for measuring the accuracy of probabilistic inference algorithms. In *Annual Conference on Neural Information Processing Systems, NIPS 2017*. https://proceedings.neurips.cc/paper/2017/hash/acab0116c354964a558e65bdd07ff047-Abstract.html

[18] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: a general-purpose probabilistic programming system with programmable inference. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2019*. ACM. https://doi.org/10.1145/3314221.3314642

[19] Hend Dawood. 2011. *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. LAP Lambert Academic Publishing.

[20] Jesús A De Loera, Brandon Dutra, Matthias Koeppe, Stanislav Moreinis, Gregory Pinto, and Jianqiu Wu. 2013. Software for exact integration of polynomials over polyhedra. *Computational Geometry* 46, 3 (2013). https://doi.org/10.1016/j.comgeo.2012.09.001

[21] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. 1987. Hybrid Monte Carlo. *Physics letters B* 195, 2 (1987). https://doi.org/10.1016/0370-2693(87)91197-X

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

[23] Christian Fecht and Helmut Seidl. 1999. A Faster Solver for General Systems of Equations. *Science of Computer Programming* 35, 2 (1999). https://doi.org/10.1016/S0167-6423(99)00009-X

[24] Timothy S. Freeman and Frank Pfenning. 1991. Refinement Types for ML. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 1991*. ACM. https://doi.org/10.1145/113445.113468

[25] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018 (PMLR, Vol. 84)*. https://proceedings.mlr.press/v84/ge18b.html

[26] Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *International Conference on Computer Aided Verification, CAV 2016 (LNCS, Vol. 9779)*. Springer. https://doi.org/10.1007/978-3-319-41528-4_4

[27] Timon Gehr, Samuel Steffen, and Martin T. Vechev. 2020. λPSI: exact inference for higher-order probabilistic programs. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020*. ACM. https://doi.org/10.1145/3385412.3386006

[28] Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. 2012. Probabilistic symbolic execution. In *International Symposium on Software Testing and Analysis, ISSTA 2012*. ACM. https://doi.org/10.1145/2338965.2336773

[29] Zoubin Ghahramani. 2013. Bayesian non-parametrics and the probabilistic approach to modelling. *Philosophical Transactions of the Royal Society A* 371 (2013). https://doi.org/10.1098/rsta.2011.0553

[30] Patrice Godefroid. 2007. Compositional dynamic test generation. In *ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2007*. ACM. https://doi.org/10.1145/1190216.1190226

[31] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *Conference on Uncertainty in Artificial Intelligence, UAI 2008*. AUAI Press.

[32] Noah D. Goodman and Andreas Stuhlmüller. 2014. *The Design and Implementation of Probabilistic Programming Languages*.

[33] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Future of Software Engineering, FOSE 2014*. ACM. https://doi.org/10.1145/2593882.2593900

[34] Maria I. Gorinova, Dave Moore, and Matthew D. Hoffman. 2020. Automatic Reparameterisation of Probabilistic Programs. In *International Conference on Machine Learning, ICML 2020 (PMLR, Vol. 119)*. http://proceedings.mlr.press/v119/gorinova20a.html

[35] Roger B. Grosse, Siddharth Ancha, and Daniel M. Roy. 2016. Measuring the reliability of MCMC inference with bidirectional Monte Carlo. In *Annual Conference on Neural Information Processing Systems, NIPS 2016*. https://proceedings.neurips.cc/paper/2016/hash/0e9fa1f3e9e66792401a6972d477dcc3-Abstract.html

[36] Roger B. Grosse, Zoubin Ghahramani, and Ryan P. Adams. 2015. Sandwiching the marginal likelihood using bidirectional Monte Carlo. *CoRR* abs/1511.02543 (2015). https://doi.org/10.48550/arXiv.1511.02543

[37] N. L. Hjort, Chris Homes, Peter Muller, and Stephen G. Walker. 2010. *Bayesian Nonparametrics*. Cambridge University Press. https://doi.org/10.1017/CBO9780511802478

[38] Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. 2020. Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020). https://doi.org/10.1145/3428208

[39] Zixin Huang, Saikat Dutta, and Sasa Misailovic. 2021. AQUA: Automated Quantized Inference for Probabilistic Programs. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2021 (LNCS, Vol. 12971)*. Springer. https://doi.org/10.1007/978-3-030-88885-5_16

[40] Andrew Kenyon-Roberts and C.-H. Luke Ong. 2021. Supermartingales, Ranking Functions and Probabilistic Lambda Calculus. In *Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*. IEEE. https://doi.org/10.1109/LICS52264.2021.9470550

[41] Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. 2021. Densities of Almost Surely Terminating Probabilistic Programs are Differentiable Almost Everywhere. In *European Symposium on Programming, ESOP 2021 (LNCS, Vol. 12648)*. Springer. https://doi.org/10.1007/978-3-030-72019-3_16

[42] Carol Mak, Fabian Zaiser, and Luke Ong. 2021. Nonparametric Hamiltonian Monte Carlo. In *International Conference on Machine Learning, ICML 2021 (PMLR, Vol. 139)*. http://proceedings.mlr.press/v139/mak21a.html

[43] Chris Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press. Cambridge, MA.

[44] David Monniaux. 2000. Abstract Interpretation of Probabilistic Semantics. In *International Symposium on Static Analysis, SAS 2000 (LNCS, Vol. 1824)*. Springer. https://doi.org/10.1007/978-3-540-45099-3_17

[45] David Monniaux. 2001. An abstract Monte-Carlo method for the analysis of probabilistic programs. In *ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2001*. ACM. https://doi.org/10.1145/360204.360211

[46] Markus Müller-Olm and Helmut Seidl. 2004. Precise interprocedural analysis through linear algebra. In *ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2004*. ACM. https://doi.org/10.1145/964001.964029

[47] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *International Symposium on Functional and Logic Programming, FLOPS 2016 (LNCS, Vol. 9613)*. Springer. https://doi.org/10.1007/978-3-319-29604-3_5

[48] Radford M Neal. 2003. Slice sampling. *The annals of statistics* 31, 3 (2003). https://doi.org/10.1214/aos/1056562461

[49] Art B. Owen. 2013. *Monte Carlo theory, methods and examples*.

[50] Andreas Podelski, Ina Schaefer, and Silke Wagner. 2005. Summaries for While Programs with Recursion. In *European Symposium on Programming, ESOP 2005 (LNCS, Vol. 3444)*. Springer. https://doi.org/10.1007/978-3-540-31987-0_8

[51] David Pollard. 2002. *A User's Guide to Measure-Theoretic Probability*. Cambridge University Press. https://doi.org/10.1017/CBO9780511811555

[52] Fredrik Ronquist, Jan Kudlicka, Viktor Senderov, Johannes Borgström, Nicolas Lartillot, Daniel Lundén, Lawrence Murray, Thomas B Schön, and David Broman. 2021. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *Communications biology* 4, 1 (2021). https://doi.org/10.1038/s42003-021-01753-7

[53] Vivekananda Roy. 2020. Convergence Diagnostics for Markov Chain Monte Carlo. *Annual Review of Statistics and Its Application* 7 (2020). https://doi.org/10.1146/annurev-statistics-031219-041300

[54] Reuven Y. Rubinstein and Dirk P. Kroese. 2017. *Simulation and the Monte Carlo Method* (3rd ed.). Wiley. https://doi.org/10.1002/9781118631980

[55] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: probabilistic programming with fast exact symbolic inference. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021*. ACM. https://doi.org/10.1145/3453483.3454078

[56] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2013*. ACM. https://doi.org/10.1145/2491956.2462179

[57] Chung-chieh Shan and Norman Ramsey. 2017. Exact Bayesian inference by symbolic disintegration. In *ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*. ACM. https://doi.org/10.1145/3009837.3009852

[58] Michael J. A. Smith. 2008. Probabilistic Abstract Interpretation of Imperative Programs using Truncated Normal Distributions. *Electronic Notes in Theoretical Computer Science* 220, 3 (2008). https://doi.org/10.1016/j.entcs.2008.11.018

[59] Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *European Symposium on Programming, ESOP 2017 (LNCS, Vol. 10201)*. Springer. https://doi.org/10.1007/978-3-662-54434-1_32

[60] Sean Talts, Michael Betancourta, Daniel Simpson, Aki Vehtari, and Andrew Gelman. 2018. Validating Bayesian Inference Algorithms with Simulation-Based Calibration. *arXiv* 1804.06788 (2018). https://doi.org/10.48550/arXiv.1804.06788

[61] David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2015 (LNCS, Vol. 9286)*. Springer. https://doi.org/10.1007/978-3-319-23461-8_36

[62] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). https://doi.org/10.48550/arXiv.1809.10756

[63] Di Wang, Jan Hoffmann, and Thomas W. Reps. 2018. PMAF: an algebraic framework for static analysis of probabilistic programs. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2018*. ACM. https://doi.org/10.1145/3192366.3192408

[64] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. 2019. Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 8 (2019). https://doi.org/10.1109/TPAMI.2018.2889774

[65] Yuan Zhou, Bradley J. Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. 2019. LF-PPL: A Low-Level First Order Probabilistic Programming Language for Non-Differentiable Models. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2019 (PMLR, Vol. 89)*. http://proceedings.mlr.press/v89/zhou19b.html

A Supplementary Material for Section 3

A.1 Intervals as a lattice

Intervals $\mathbb{I}$ form a partially ordered set under interval inclusion ($\sqsubseteq$). We will sometimes need the meet $\sqcap$ and join $\sqcup$ operations, corresponding to the greatest lower bound and the least upper bound of two intervals. Note that the meet of two intervals does not exist if the two intervals are disjoint. Concretely, these two operations are given by $[a,b] \sqcap [c,d] := [\max(a,c), \min(b,d)]$ (if the two intervals overlap) and $[a,b] \sqcup [c,d] := [\min(a,c), \max(b,d)]$.

For some applications (e.g. the interval type system), we need the interval domain to be a true lattice. To turn $\mathbb{I}$ into a lattice, we add a bottom element $\bot$ (signifying an empty interval). The definition of the meet $\sqcap$ and join $\sqcup$ is extended in the natural way. The meet $\sqcap$ is extended by defining $I_1 \sqcap \bot = \bot \sqcap I_2 = \bot$ and $I_1 \sqcap I_2 = \bot$ if the two intervals $I_1, I_2 \in \mathbb{I}$ are disjoint. The join $\sqcup$ satisfies $I \sqcup \bot = \bot \sqcup I = I$.

A.2 Lifting Functions to Intervals

For constants $c \in \mathbb{R}$ (i.e. nullary functions), for common functions like $+, -, \times, |\cdot|$, min, max, for monotonically increasing functions $f_\nearrow : \mathbb{R} \to \mathbb{R}$, and for monotonically decreasing functions $f_\searrow : \mathbb{R} \to \mathbb{R}$, it is easy to describe the interval-lifted functions $+^{\mathbb{I}}$, $-^{\mathbb{I}}$, $\times^{\mathbb{I}}$, $|\cdot|^{\mathbb{I}}$, $\min^{\mathbb{I}}$, $\max^{\mathbb{I}}$, $f_\nearrow^{\mathbb{I}}$, and $f_\searrow^{\mathbb{I}}$:

$$
\begin{aligned}
c^{\mathbb{I}} &= [c,c] \\
-^{\mathbb{I}}[x_1,y_1] &= [-y_1, -x_1] \\
|[x_1,y_1]|^{\mathbb{I}} &= \begin{cases} [0, \max(|x_1|,|y_1|)] \text{ if } x_1 \le 0 \le y_1 \\ [\min(|x_1|,|y_1|), \max(|x_1|,|y_1|)] \text{ else} \end{cases} \\
[x_1,y_1] +^{\mathbb{I}} [x_2,y_2] &= [x_1+x_2, y_1+y_2] \\
[x_1,y_1] -^{\mathbb{I}} [x_2,y_2] &= [x_1-y_2, y_1-x_2] \\
[x_1,y_1] \times^{\mathbb{I}} [x_2,y_2] &= [\min(x_1x_2, x_1y_2, y_1x_2, y_1y_2), \\
&\qquad \max(x_1x_2, x_1y_2, y_1x_2, y_1y_2)] \\
\min^{\mathbb{I}}([x_1,y_1],[x_2,y_2]) &= [\min(x_1,x_2), \min(y_1,y_2)] \\
\max^{\mathbb{I}}([x_1,y_1],[x_n,y_n]) &= [\max(x_1,x_2), \max(y_1,y_2)] \\
f_\nearrow^{\mathbb{I}}([x_1,y_1]) &= [f_\nearrow(x_1), f_\nearrow(y_1)] \\
f_\searrow^{\mathbb{I}}([x_1,y_1]) &= [f_\searrow(y_1), f_\searrow(x_1)]
\end{aligned}
$$

where we write $f(\pm\infty)$ for $\lim_{x\to\pm\infty} f(x) \in \mathbb{R}_\infty$, respectively.

A.3 Properties of Interval Reduction

We can define a refinement relation $M \triangleleft M'$ ("M refines M'") between a standard term M and an interval term M', if M is obtained from M' by replacing every occurrence of $\underline{[a,b]}$ with some $r \in [a,b]$.

Lemma 3.1. *Let $\vdash P : \boldsymbol{R}$ be a program. For any interval trace $\boldsymbol{t}$ and concrete trace $\boldsymbol{s} \triangleleft \boldsymbol{t}$, we have $\mathrm{wt}_P(\boldsymbol{s}) \in \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t})$ and $\mathrm{val}_P(\boldsymbol{s}) \in \mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t})$ (provided $\mathrm{val}_P(\boldsymbol{s})$ is defined).*

Proof. If the interval reduction $\to_{\mathbb{I}}$ gets stuck, $\mathrm{wt}_P^{\mathbb{I}}$ is $[0,\infty]$ and $\mathrm{val}_P^{\mathbb{I}}$ is $[-\infty,\infty]$, so the claim is certainly true. Otherwise, for each $(M_{\mathbb{I}}, \boldsymbol{t}, w_{\mathbb{I}}) \to_{\mathbb{I}} (M'_{\mathbb{I}}, \boldsymbol{t}', w'_{\mathbb{I}})$ reduction step, we can do a reduction step $(M, \boldsymbol{s}, w) \to (M', \boldsymbol{s}', w')$ where $M' \triangleleft M'_{\mathbb{I}}$ and $w' \in w'_{\mathbb{I}}$, and $\boldsymbol{s}' \triangleleft \boldsymbol{t}'$ if $M \triangleleft M_{\mathbb{I}}$, $w \in w_{\mathbb{I}}$, and $\boldsymbol{s} \triangleleft \boldsymbol{t}$. Since the reduction doesn't get stuck, we end up with a value $\underline{r} \triangleleft \underline{[a,b]}$, so $\mathrm{val}_P(\boldsymbol{s}) = r \in [a,b] = \mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t})$. □

A.4 Additional Possible Reduction Rules

The interval semantics as presented has the unfortunate property that even a simple program like

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

requires infinitely many interval traces to achieve a finite upper bound. The reason is that the right branch $\mathsf{score}(1)$ is taken if the sampled value is in the open interval $(0,1]$. To approximate this using closed intervals $[a,b]$ that our analysis supports, we need infinitely many intervals, e.g. $\{[2^{-n-1}, 2^{-n}] \mid n \in \mathbb{N}\}$. Adding (half-)open intervals to the semantics would solve this specific problem, but not more general ones, where the guard condition is for example

$$\mathsf{sample} - \mathsf{sample} \le 0.$$

In that case, we have to approximate the set $\{(x,y) \in [0,1]^2 \mid x \le y\}$. For the lower bounds, that is not an issue, but for the upper bounds, we need an infinite number of interval traces again. We would like to use the interval traces $\langle [0, \frac{1}{2}], [0, \frac{1}{2}] \rangle$ and $\langle [\frac{1}{2}, 1], [0,1] \rangle$ to cover this set, but the reduction gets stuck on them because it is not clear which branch should be taken.

To remedy this, we could add the following two rules.

$$\frac{a \le 0 < b}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (N, \boldsymbol{t}, w \times^{\mathbb{I}} [0,1])}$$

$$\frac{a \le 0 < b}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (P, \boldsymbol{t}, w \times^{\mathbb{I}} [0,1])}$$

They basically express that if the interval bounds are not precise enough to decide what branch to take, we can take both, but have to allow the weight to be zero because it's not guaranteed that the taken branch can actually happen. This change can only improve the upper bounds, not the lower bounds because the lower bound on each weight is zero if the additional rules are used. Then the definition of upper bound can be modified in the following way:

$$\mathrm{upperBd}_P^{\mathcal{T}}(U) := \sum_{\boldsymbol{t} \in \mathcal{T}} \sum_{\substack{(P,\boldsymbol{t},[1,1]) \to_{\mathbb{I}} \\ (\underline{[a,b]}, \langle\rangle, [w_1,w_2])}} \mathrm{vol}(\boldsymbol{t}) \cdot w_2 \cdot \left[[a,b] \cap U \neq \emptyset \right]$$

This is the strategy we use for our implementation and is a natural extension of the existing semantics: it requires very few changes to the soundness and completeness proofs.

A downside of the previous approach is that the bounds are not always very tight: for the term $\mathsf{if}(\dots)\,\mathsf{score}(50)$

$$\frac{}{((\lambda x.\mathcal{M})\mathcal{V}, n, \Delta, \Xi) \rightsquigarrow (\mathcal{M}[\mathcal{V}/x], n, \Delta, \Xi)}$$

$$\frac{}{(\mathsf{score}(\mathcal{V}), n, \Delta, \Xi) \rightsquigarrow (\mathcal{V}, n, \Delta \cup \{\mathcal{V} \geq 0\}, \Xi \cup \{\mathcal{V}\})}$$

$$\frac{}{((\mu^{\varphi}_{x}.\,\mathcal{M})\mathcal{V}, n, \Delta, \Xi) \rightsquigarrow (\mathcal{M}[\mathcal{V}/x, (\mu^{\varphi}_{x}.\,\mathcal{M})/\varphi], n, \Delta, \Xi)}$$

$$\frac{}{(\mathsf{sample}, n, \Delta, \Xi) \rightsquigarrow (\alpha_{n+1}, n+1, \Delta, \Xi)}$$

$$\frac{}{(\mathsf{if}(\mathcal{V}, \mathcal{N}, \mathcal{P})), n, \Delta, \Xi) \rightsquigarrow (\mathcal{N}, n, \Delta \cup \{\mathcal{V} \leq 0\}, \Xi)}$$

$$\frac{}{(\mathsf{if}(\mathcal{V}, \mathcal{N}, \mathcal{P})), n, \Delta, \Xi) \rightsquigarrow (\mathcal{P}, n, \Delta \cup \{\mathcal{V} > 0\}, \Xi)}$$

$$\frac{(\mathcal{R}, n, \Delta, \Xi) \rightsquigarrow (\mathcal{M}, n', \Delta', \Xi')}{(\mathcal{E}[\mathcal{R}], n, \Delta, \Xi) \rightsquigarrow (\mathcal{E}[\mathcal{M}], n', \Delta', \Xi')}$$

Figure 8. Reduction rules for symbolic execution

else score(100), it returns bounds $[0, 150]$ instead of $[50, 100]$. To improve this, we could omit the multiplication with $[0, 1]$.

$$\frac{a \leq 0 < b}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (N, \boldsymbol{t}, w)}$$

$$\frac{a \leq 0 < b}{(\mathsf{if}(\underline{[a,b]}, N, P), \boldsymbol{t}, w) \to_{\mathbb{I}} (P, \boldsymbol{t}, w)}$$

However, this complicates the equations of our bounds. With this semantics, we have to compute minima and suprema instead of a simple sum:

$$\mathsf{lowerBd}^{\mathcal{T}}_{P}(U) := \sum_{\boldsymbol{t} \in \mathcal{T}} \min_{\substack{(P,\boldsymbol{t},[1,1]) \to_{\mathbb{I}} \\ (\underline{[a,b]}, \langle\rangle, [w_1, w_2])}} \mathrm{vol}(\boldsymbol{t}) \cdot w_1 \cdot \big[[a,b] \subseteq U\big]$$

$$\mathsf{upperBd}^{\mathcal{T}}_{P}(U) := \sum_{\boldsymbol{t} \in \mathcal{T}} \sup_{\substack{(P,\boldsymbol{t},[1,1]) \to_{\mathbb{I}} \\ (\underline{[a,b]}, \langle\rangle, [w_1, w_2])}} \mathrm{vol}(\boldsymbol{t}) \cdot w_2 \cdot \big[[a,b] \cap U \neq \emptyset\big]$$

This is harder to implement because the sums cannot be computed incrementally, but many temporary results have to be kept in memory to compute the minima and suprema. Proving soundness and completeness for this would require more substantial changes to the proofs.

B Symbolic Execution

In this section we formally introduce stochastic symbolic execution. We make use of this form of symbolic execution in two separate ways. First, our completeness proof hinges on guarantees provided by the symbolic execution in order to identify a suitable set of interval traces. Second, our tool GuBPI relies on the symbolic execution as a first step in the program analysis, in order to identify relevant paths and independent subexpressions, and to avoid repeated evaluation in a small-step semantics.

High-level idea. The overarching idea of symbolic execution is to postpone the evaluation of sample expressions and instead use a *sample variable* to symbolically represent its outcome. As a consequence, branching and scoring steps cannot be executed concretely, so we record them symbolically instead.

Symbolic terms. To postpone concrete sample decisions we introduce *sample variables* $\alpha_1, \alpha_2, \ldots$ into our language. We then define *symbolic terms* and *symbolic values* by extending interval terms and values by adding two new constructs: every sample variable α_j is a symbolic value and for every primitive function f and symbolic values $\mathcal{V}_1, \ldots, \mathcal{V}_{|f|}$, the symbolic term $\underline{f}(\mathcal{V}_1, \ldots, \mathcal{V}_{|f|})$ is a symbolic value, denoting a function application that is postponed until all sample variables are instantiated. We denote symbolic terms by $\mathcal{M}, \mathcal{N}, \mathcal{P}$ and symbolic values by $\mathcal{V}, \mathcal{W}$. Formally we define

$$\mathcal{V} := x \mid \underline{r} \mid \lambda x.\mathcal{M} \mid \mu^{\varphi}_{x}.\,\mathcal{M} \mid \alpha_i \mid \underline{f}(\mathcal{V}_1, \ldots, \mathcal{V}_{|f|})$$
$$\mathcal{M}, \mathcal{N}, \mathcal{P} := \mathcal{V} \mid \mathcal{M}\mathcal{N} \mid \mathsf{if}(\mathcal{M}, \mathcal{N}, \mathcal{P}) \mid \underline{f}(\mathcal{M}_1, \ldots, \mathcal{M}_{|f|})$$
$$\mid \mathsf{sample} \mid \mathsf{score}(\mathcal{M})$$

The definition of redex and evaluation context extends naturally (recall that we regard α_j as a value).

Symbolic execution. A symbolic constraint is a pair $(\mathcal{V} \bowtie r)$ where $\mathcal{V}$ is a symbolic value, $\bowtie \in \{\leq, <, \geq, >\}$ and $r \in \mathbb{R}$. A ***symbolic configuration*** has the form $\psi = (\mathcal{M}, n, \Delta, \Xi)$ where $\mathcal{M}$ is a symbolic term, $n \in \mathbb{N}$ a natural number (used to obtain fresh sample variables), Δ a set of symbolic constraints (which track the symbolic conditions on the current execution path), and Ξ is a set of symbolic values (which records all symbolic values scored on the current path). When executing symbolically: (1) we evaluate each sample to a fresh sample variable, (2) we postpone function application, (3) for each conditional, we explore both branches (our reduction is nondeterministic) and record the symbolic inequalities that must hold along the current path, and (4) we record the symbolic values that we scored with. We give the reduction rules in Fig. 8.

We call a tuple $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ (a symbolic configuration where the symbolic term is a value) a ***symbolic path***. For a symbolic configuration ψ, we write $\mathit{symPaths}(\psi)$ for the set of symbolic paths reached when evaluating from ψ. Note that $\mathit{symPaths}(\psi)$ is countable.

Let $\mathcal{V}$ be a symbolic value of type $\mathbf{R}$ (no λ-abstraction or fixed point) with sample variables within $\{\alpha_1, \ldots, \alpha_n\}$. For a trace $\boldsymbol{s} = \langle r_1, \ldots, r_n \rangle \in [0,1]^n$, we define $\mathcal{V}[\boldsymbol{s}/\overline{\alpha}]$ as the value (in $\mathbb{R}$) obtained by replacing the sample variables in $\overline{\alpha}$ with $\boldsymbol{s}$ and evaluate the postponed primitive function applications. For a symbolic path $\Psi = (\mathcal{V}, n, \Delta, \Xi)$, we define $[\![\Psi]\!](U)$ as

$$\int_{[0,1]^n} \big[\mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \in U\big] \prod_{C \bowtie r \in \Delta} \big[C[\boldsymbol{s}/\overline{\alpha}] \bowtie r\big] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}]\, \mathrm{d}\boldsymbol{s}.$$

Solution to symbolic constraints. To simplify notation (and avoid extensive use of Iverson brackets) we introduce notation for the set of traces that satisfy a set of symbolic constraints. Given a set of symbolic constraints Δ with sample

variables contained in $\{\alpha_1, \dots, \alpha_n\}$ we define

$$\mathrm{Sat}_n(\Delta) := \bigcap_{(\mathcal{V} \bowtie r) \in \Delta} \{s \in [0,1]^n \mid \mathcal{V}[s/\overline{\alpha}] \bowtie r\}$$

as the set of actual values for the sample variables that satisfy all constraints. It follows immediately from the definitions that we can replace the Iverson brackets in $[\![\Psi]\!]$ by directly restricting the integral to the traces in $\mathrm{Sat}_n(\Delta)$.

Lemma B.1. *For any symbolic path $(\mathcal{V}, n, \Delta, \Xi)$ and any $U \in \Sigma_{\mathbb{R}}$ we have*

$$[\![(\mathcal{V}, n, \Delta, \Xi)]\!](U) = \int_{\mathrm{Sat}_n(\Delta)} [\mathcal{V}[s/\overline{\alpha}] \in U] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}] \, \mathrm{d}s.$$

Correctness of symbolic execution. We can now establish a correspondence between symbolic execution and the ordinary reduction. If we wish to symbolically analyse a term P, we consider the (symbolic) reductions starting from $(P, 0, \emptyset, \emptyset)$, resulting in the symbolic paths $\mathit{symPaths}(P, 0, \emptyset, \emptyset)$.

Lemma B.2. *Let $\vdash P : \boldsymbol{R}$ and suppose we have $(\mathcal{V}, n, \Delta, \Xi) \in \mathit{symPaths}(P, 0, \emptyset, \emptyset)$, where P is interpreted as a symbolic term. Then for any $s \in \mathit{Sat}_n(\Delta)$, we have*

$$(P, s, 1) \to^* (\mathcal{V}[s/\overline{\alpha}], \langle\rangle, \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}]).$$

Proof. A similar proof can be found in [41, Theorem 1]. □

Lemma B.3. *Let $\vdash P : \boldsymbol{R}$ and suppose $(P, s, 1) \to^* (\underline{r}, \langle\rangle, w)$ for some $r \in \mathbb{R}$. Then there exists a* unique *$(\mathcal{V}, n, \Delta, \Xi) \in \mathit{symPaths}(P, 0, \emptyset, \emptyset)$ such that $s \in \mathit{Sat}_n(\Delta)$. For this unique symbolic path we have $w = \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}]$ and $r = \mathcal{V}[s/\overline{\alpha}]$.*

Proof sketch. Choose the same branches in the $\rightsquigarrow$-reduction of P as in its $\to$-reduction. Then it is straightforward to see that this correspondence holds at every symbolic reduction step: if $(P, 0, \emptyset, \emptyset) \rightsquigarrow^* (\mathcal{P}', n, \Delta, \Xi)$ then the corresponding $\to$-reduction steps yield $(P, ss', 1) \to^* (P', s', w)$ where s has length n, P' is $\mathcal{P}'[s/\overline{\alpha}]$ (after evaluating delayed primitve function applications), Δ records the guards $C[s/\overline{\alpha}] \le 0$ or $C[s/\overline{\alpha}] > 0$ that need to hold for the trace s, and finally, the weight w is given by $\prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}]$ at any point. □

Theorem 6.1. *Let $\vdash P : \boldsymbol{R}$ be a program and $U \in \Sigma_{\mathbb{R}}$. Then*

$$[\![P]\!](U) = \textstyle\sum_{\Psi \in \mathit{symPaths}(P,0,\emptyset,\emptyset)} [\![\Psi]\!](U).$$

Proof.

$$\begin{aligned}
[\![P]\!](U) &= \sum_{n \in \mathbb{N}} \int_{[0,1]^n} [\mathrm{val}_P(s) \in U] \mathrm{wt}_P(s) \, \mathrm{d}s \\
&= \sum_{n \in \mathbb{N}} \int_{[0,1]^n} \sum_{(\mathcal{V},n,\Delta,\Xi)} \\
&\quad \left([s \in \mathrm{Sat}_n(\Delta)] [\mathcal{V}[s/\overline{\alpha}] \in U] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}] \, \mathrm{d}s \right) \\
&= \sum_{(\mathcal{V},n,\Delta,\Xi)} \int_{\mathrm{Sat}_n(\Delta)} [\mathcal{V}[s/\overline{\alpha}] \in U] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[s/\overline{\alpha}] \, \mathrm{d}s \\
&= \sum_{(\mathcal{V},n,\Delta,\Xi)} [\![(\mathcal{V}, n, \Delta, \Xi)]\!](U)
\end{aligned}$$

where the sum ranges over symbolic paths $(\mathcal{V}, n, \Delta, \Xi) \in \mathit{symPaths}(P, 0, \emptyset, \emptyset)$. The first equality is by definition, the second one by Lemmas B.2 and B.3, the third by noting that $\mathrm{Sat}_n(\Delta) \subseteq [0,1]^n$ and exchanging the infinite sum and integral (which is allowed because everything is nonnegative) and the fourth by Lemma B.1. □

C Supplementary Material for Section 4

C.1 Infinite Trace Semantics

A convenient alternative to the (finite) trace semantics is using infinite traces $\mathbb{T}_\infty := [0,1]^{\mathbb{N}}$ with a suitable σ-algebra and measure $\mu_{\mathbb{T}_\infty}$ [16, 40]. The σ-algebra on $\mathbb{T}_\infty$ is defined as the smallest σ-algebra that contains all sets $U \times \mathbb{T}_\infty$ where $U \in \Sigma_{[0,1]^n}$ for some $n \in \mathbb{N}$. The measure $\mu_{\mathbb{T}_\infty}$ is the unique measure with $\mu_{\mathbb{T}_\infty}(U \times \mathbb{T}_\infty) = \lambda_n(U)$ for $U \in \Sigma_{[0,1]^n}$. We use the symbol $\boldsymbol{u}$ for an infinite trace in $\mathbb{T}_\infty$. For a finite trace s and infinite trace $\boldsymbol{u}$ we write $s\boldsymbol{u} \in \mathbb{T}_\infty$ for their concatenation. For any infinite trace $\boldsymbol{u} \in \mathbb{T}_\infty$, there is at most one prefix $s \in \mathbb{T}$ with $\mathrm{wt}_P(s) > 0$ since the reduction is deterministic. We can therefore define $\mathrm{wt}_P^\infty(\boldsymbol{u}) := \mathrm{wt}_P(s)$ and $\mathrm{val}_P^\infty(\boldsymbol{u}) := \mathrm{val}_P(s)$ if such a prefix s exists, and $\mathrm{wt}_P^\infty(\boldsymbol{u}) := 0$ and $\mathrm{val}_P^\infty(\boldsymbol{u})$ is undefined otherwise. The infinite trace semantics of a term is then defined as

$$[\![P]\!](U) := \int_{(\mathrm{val}_P^\infty)^{-1}(U)} \mathrm{wt}_P^\infty(\boldsymbol{u}) \, \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u}).$$

Lemma C.1. *The finite and infinite trace semantics agree, that is:*

$$\int_{(\mathrm{val}_P)^{-1}(U)} \mathrm{wt}_P(s) \, \mu_{\mathbb{T}}(\mathrm{d}s) = \int_{(\mathrm{val}_P^\infty)^{-1}(U)} \mathrm{wt}_P^\infty(\boldsymbol{u}) \, \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u}).$$

Proof. Observe that $\mathrm{val}_P^\infty(s\boldsymbol{u}) = \mathrm{val}_P(s)$ and $\mathrm{wt}_P^\infty(s\boldsymbol{u}) = \mathrm{wt}_P(s)$ for all $\boldsymbol{u} \in \mathbb{T}_\infty$ if $\mathrm{wt}_P(s) > 0$. Then we get:

$$\begin{aligned}
&\int_{(\mathrm{val}_P)^{-1}(U)} \mathrm{wt}_P(s) \, \mu_{\mathbb{T}}(\mathrm{d}s) \\
&= \int_{(\mathrm{wt}_P)^{-1}(\mathbb{R}_{>0})} [\mathrm{val}_P(s) \in U] \mathrm{wt}_P(s) \int_{\mathbb{T}_\infty} \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u}) \mu_{\mathbb{T}}(\mathrm{d}s) \\
&= \int_{(\mathrm{wt}_P)^{-1}(\mathbb{R}_{>0})} \int_{\mathbb{T}_\infty} [\mathrm{val}_P^\infty(s\boldsymbol{u}) \in U] \mathrm{wt}_P^\infty(s\boldsymbol{u}) \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u}) \mu_{\mathbb{T}}(\mathrm{d}s) \\
&= \int_{(\mathrm{wt}_P)^{-1}(\mathbb{R}_{>0}) \times \mathbb{T}_\infty} [\mathrm{val}_P^\infty(\boldsymbol{u}) \in U] \mathrm{wt}_P^\infty(\boldsymbol{u}) \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u}) \\
&= \int_{(\mathrm{val}_P^\infty)^{-1}(U)} \mathrm{wt}_P^\infty(\boldsymbol{u}) \mu_{\mathbb{T}_\infty}(\mathrm{d}\boldsymbol{u})
\end{aligned}$$

where we used the fact that the sets $\{s\} \times \mathbb{T}_\infty$ are disjoint for different $s \in \mathrm{wt}_P^{-1}(\mathbb{R}_{>0})$ because otherwise we would be able to find a trace s as a prefix of s' and both having positive weight, which is impossible due to the deterministic reduction. Therefore $(\mathrm{wt}_P)^{-1}(\mathbb{R}_{>0}) \times \mathbb{T}_\infty = (\mathrm{wt}_P^\infty)^{-1}(\mathbb{R}_{>0})$

and everything works as desired. Note that the second to last equality follows from Fubini's theorem and the fact that the product measure of $\mu_{\mathbb{T}}$ and $\mu_{\mathbb{T}_\infty}$ is $\mu_{\mathbb{T}_\infty}$ again. □

C.2 Exhaustivity and Soundness

Example C.1 (more examples of exhaustive sets)**.** Here are more examples and counterexamples for exhaustivity.

(i) $\{\langle\rangle\}$ is an (uninteresting) exhaustive set and only useful for deterministic programs.

(ii) $\{\langle [2^{-n-1}, 2^{-n}]\rangle \mid n \in \mathbb{N}\}$ is exhaustive because only the trace $\langle 0\rangle$ (with measure 0) is not covered.

(iii) Let $a_n \geq 0$ be a converging series, i.e. $\sum_{i=1}^{n} a_n < \infty$, for example $a_n = n^{-2}$. Define

$$\mathcal{T} := \{\langle [0, e^{-a_1}], \dots, [0, e^{-a_n}], [e^{-a_{n+1}}, 1]\rangle \mid n \in \mathbb{N}\}.$$

This is not an exhaustive set since it doesn't cover any of the traces in

$$[0, e^{-a_1}) \times [0, e^{-a_2}, 1) \times \cdots$$

which has measure $\prod_{i=1}^{\infty} e^{-a_i} = \exp\left(-\sum_{i=1}^{\infty} a_i\right) > 0$.

We also note that exhaustivity can be expressed just in terms of finite traces as well, at the cost of a more complicated definition.

Lemma C.2. *A set of interval traces $\mathcal{T}$ is exhaustive if and only if*

$$\mu_{\mathbb{T}}\left([0,1]^n \setminus \left(\bigcup_{\langle I_1,\dots,I_m\rangle \in \mathcal{T}, m \leq n} I_1 \times \cdots \times I_m \times [0,1]^{n-m}\right)\right) \to 0$$

as $n \to \infty$.

Proof. Let $S := \mathbb{T}_\infty \setminus \bigcup_{\boldsymbol{t} \in \mathcal{T}} \mathit{cover}(\boldsymbol{t})$. By the definition of exhaustivity, $\mu_{\mathbb{T}_\infty}(S) = 0$. Let

$$S_n = [0,1]^n \setminus \left(\bigcup_{\langle I_1,\dots,I_m\rangle \in \mathcal{T}, m \leq n} I_1 \times \cdots \times I_m \times [0,1]^{n-m}\right).$$

It's easy to see that $S = \bigcap_{n=0}^{\infty} S_n \times \mathbb{T}_\infty$ where $S_n \times \mathbb{T}_\infty$ is a decreasing sequence of sets: $S_1 \times \mathbb{T}_\infty \supseteq S_2 \times \mathbb{T}_\infty \supseteq \cdots$. Since measures are continuous from above, we have $\lim_{n\to\infty} \mu_{\mathbb{T}}(S_n) = \lim_{n\to\infty} \mu_{\mathbb{T}_\infty}(S_n \times \mathbb{T}_\infty) = \mu_{\mathbb{T}_\infty}(S) = 0$, as desired. □

The following lemma establishes a correspondence between infinite trace semantics and interval trace semantics.

Lemma C.3. *For any interval trace $\boldsymbol{t}$ and infinite trace $\boldsymbol{s}_\infty$ with a prefix $\boldsymbol{s}$ such that $\boldsymbol{s} \triangleleft \boldsymbol{t}$, we have $\mathrm{wt}_P^\infty(\boldsymbol{s}_\infty) \in \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t})$ and $\mathrm{val}_P^\infty(\boldsymbol{s}_\infty) \in \mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t})$.*

Proof. Follows directly from the definition of infinite trace semantics and Lemma 3.1. □

Using the previous results, we can prove soundness of upper bounds.

Theorem 4.2 (Sound upper bounds)**.** *Let $\mathcal{T}$ be a countable and exhaustive set of interval traces and $\vdash P : \boldsymbol{R}$ a program. Then $[\![P]\!] \leq \mathrm{upperBd}_P^{\mathcal{T}}$.*

Proof. For any $U \in \Sigma_{\mathbb{R}}$, we have

$$\begin{aligned}
&\mathrm{upperBd}_P^{\mathcal{T}}(U) \\
&= \sum_{\boldsymbol{t}\in\mathcal{T}} \mathrm{vol}(\boldsymbol{t})(\sup \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}))[\mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t}) \cap U \neq \emptyset] \\
&= \sum_{\boldsymbol{t}\in\mathcal{T}} \int_{(\!|\boldsymbol{t}|\!)} (\sup \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}))[\mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t}) \cap U \neq \emptyset]\, \mathrm{d}\boldsymbol{s} \\
&\geq \sum_{\boldsymbol{t}\in\mathcal{T}} \int_{(\!|\boldsymbol{t}|\!)} \int_{\mathbb{T}_\infty} \mathrm{wt}_P^\infty(\boldsymbol{su})[\mathrm{val}_P^\infty(\boldsymbol{su}) \in U]\, \mathrm{d}\boldsymbol{u}\, \mathrm{d}\boldsymbol{s} \qquad (4) \\
&\geq \int_{\bigcup_{\boldsymbol{t}\in\mathcal{T}} (\!|\boldsymbol{t}|\!)\times\mathbb{T}_\infty} \mathrm{wt}_P^\infty(\boldsymbol{u})[\mathrm{val}_P^\infty(\boldsymbol{u}) \in U]\, \mathrm{d}\boldsymbol{s} \\
&\geq \int_{\mathbb{T}_\infty} \mathrm{wt}_P^\infty(\boldsymbol{u})[\mathrm{val}_P^\infty(\boldsymbol{u}) \in U]\, \mathrm{d}\boldsymbol{u} \qquad (5) \\
&= [\![P]\!](U) \qquad (6)
\end{aligned}$$

where Eq. (4) follows from Lemma C.3, Eq. (5) from exhaustivity and Eq. (6) from Lemma C.1. □

C.3 Assumptions for Completeness

Remarks on Assumption 1. We can formally express Assumption 1 from Section 4 about a given program $\vdash P : \mathbf{R}$ as follows. For each symbolic path $\Psi = (\mathcal{V}, n, \Delta, \Xi)$, we require that $\mathcal{V}$, each C with $C \bowtie 0 \in \Delta$, and each $\mathcal{W} \in \Xi$ contain each sample variable α_i at most once.

Example C.2. The pedestrian example (Example 1.1) satisfies Assumption 1 because the symbolic paths have the form $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ with:

$$\begin{aligned}
\mathcal{V} &= 3\alpha_1 \\
n &= 2k+1 \\
\Delta &= \{\alpha_3 - \tfrac{1}{2} \bowtie 0, \alpha_5 - \tfrac{1}{2} \bowtie 0, \dots, \alpha_{2k+1} - \tfrac{1}{2} \bowtie 0\} \\
&\quad \cup \{3\alpha_1 > 0, \\
&\qquad 3\alpha_1 \pm \alpha_2 > 0, \\
&\qquad \dots, \\
&\qquad 3\alpha_1 \pm \alpha_2 \pm \alpha_4 \cdots \pm \alpha_{2k-2} > 0, \\
&\qquad 3\alpha_1 \pm \alpha_2 \pm \alpha_4 \cdots \pm \alpha_{2k} \leq 0\} \\
\Xi &= \{\mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}(\alpha_2 + \alpha_4 + \cdots + \alpha_{2k})\}
\end{aligned}$$

As we can see, none of the symbolic values contains a sample variable twice, so the assumption is satisfied.

Remarks on Assumption 2. We first prove the sufficient condition for interval separability from Section 4.2.

Lemma C.4. *If a function $f : \mathbb{R}^n \to \mathbb{R}$ is boxwise continuous and preimages of points are null sets then f is interval separable.*

Proof. We decompose $f^{-1}([a,b]) = f^{-1}((a,b)) \cup f^{-1}(\{a,b\})$ and deal with the former set first. By boxwise continuity, $f = \bigcup_i f|_{B_i}$ where $\bigcup_i B_i = \mathbb{R}^n$ and each $f|_{B_i}$ is continuous on B_i. To show that the preimage $f^{-1}((a,b))$ can be tightly approximated by a countable set of boxes, it suffices to show this for each $(f|_{B_i})^{-1}((a,b))$. This set is open in B_i by continuity of $f|_{B_i}$, so it can be written as a countable union of boxes (e.g. by taking a box within B_i around each rational point, which exists because it's an open set). By the assumption, the preimage $f^{-1}(\{a,b\})$ is a null set. Hence $f^{-1}([a,b])$ can be approximated by a null set. □

Note that a composition of interval separable functions need *not* be interval separable. This is an incorrect assumption made in the completeness proof of [4]. (To fix their Theorem 3.8, one needs to make the additional assumption that the set of primitive functions be closed under composition.) To see this, let $f, g : \mathbb{R} \to \mathbb{R}$ be interval separable functions and I an interval. By definition, there are intervals B_i such that $\bigcup_i B_i \cup N = f^{-1}(I)$ where N is a null set. Then by interval separability, the preimage $g^{-1}(\bigcup_i B_i)$ can be tightly approximated by interals B'_j, but the preimage $g^{-1}(N)$ need not be a null set. It is also not clear at all whether one can approximate the preimage $(f \circ g)^{-1}(I)$ tightly using intervals without further restrictions on f and g. For this reason, we require the assumption that the set of primitive functions be closed under composition.

It is not immediately obvious that such a set of functions exists. One example is given by the following. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called a ***submersion*** if it is continuously differentiable and its gradient is nonzero everywhere.

Lemma C.5. *The set $\mathcal{F}_{\text{subm}}$ of submersions is closed under composition and each of its functions is boxwise continuous and interval separable.*

Proof. Boxwise continuity is obvious given that the functions are even continuously differentiable. For interval separability, we use Lemma C.4. Let $f : \mathbb{R}^n \to \mathbb{R} \in \mathcal{F}_{\text{subm}}$. Since f is a submersion, the preimage $f^{-1}(x)$ of any point $x \in \mathbb{R}$ is an $(n-1)$-dimensional submanifold of $\mathbb{R}^n$ by the preimage theorem (a variation of the implicit function theorem). Submanifolds of codimension > 1 have measure zero. (This well-known fact can be shown by writing the submanifold as a countable union of graphs and applying Fubini's theorem to each of them.) Therefore, the lemma applies.

For closure under composition, let $f : \mathbb{R}^m \to \mathbb{R}$ and $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$ for $i \in \{1, \dots, m\}$, all in $\mathcal{F}_{\text{subm}}$. The composition $g := f \circ (f_1 \times \cdots \times f_m)$ is clearly C^1 again, so we just have to check the submersion property. By the chain rule, we find that the gradient of the composition

$$\nabla g(x_1, \dots, x_m) = \begin{pmatrix} \partial_1 f(f_1(x_1), \dots, f_m(x_m)) \cdot \nabla f_1(x_1) \\ \vdots \\ \partial_m f(f_1(x_1), \dots, f_m(x_m)) \cdot \nabla f_m(x_m) \end{pmatrix}$$

is nonzero because at least one of the $\partial_i f$ is nonzero and $\nabla f_i(x_i) \neq 0$ by assumption. Hence the composition is a submersion again. □

Unfortunately, the set of submersions does not contain constant functions. This is a problem because then it is not guaranteed that partially applying a primitive function to a constant is still an admissible primitive function. (For example, this would break Lemma C.8.) Hence we need to assume that *all constant functions be primitive functions.* Luckily, the set $\mathcal{F}_{\text{subm}}$ of submersions can be easily extended to accommodate this.

Lemma C.6. *Let $\mathcal{F}^*_{\text{subm}}$ be the set of functions $f : \mathbb{R}^n \to \mathbb{R}$ (for all $n \in \mathbb{N}$) such that whenever the partial derivative $\partial_i f(x)$ is zero for some $i \in \{1, \dots, n\}$ and $x \in \mathbb{R}^n$ then f is constant in its i-th argument, i.e. there is a function $f^* : \mathbb{R}^{n-1} \to \mathbb{R}$ such that $f(x_1, \dots, x_n) = f^*(x_1, \dots, x_{i-1}, x_{i+1}, \dots x_n)$. This set satisfies all the assumptions about sets of primitive functions: it is closed under composition, contains all constant functions, and all its functions are boxwise continuous and interval separable.*

Proof. Boxwise continuity is obvious given that the functions are even continuously differentiable. Similarly, it is clear that $\mathcal{F}^*_{\text{subm}}$ contains all constant functions.

For interval separability, let $f : \mathbb{R}^n \to \mathbb{R} \in \mathcal{F}^*_{\text{subm}}$ and $J \subseteq \{1, \dots, n\}$ be the set of indices in which f is not constant, and J' its complement. Hence there is a submersion $f_J : \mathbb{R}^{|J|} \to \mathbb{R}$ such that $f(x) = f_J(x_J)$ where x_J stands for the vector of coordinates of x with index in J. The preimage of $f^{-1}(U) \subseteq \mathbb{R}^{|J|}$ of any set $U \subseteq \mathbb{R}$ can be tightly approximated by boxes if and only if $f_J^{-1}(U)$ can because $f^{-1}(U)$ is a Cartesian product of $f_J^{-1}(U)$ and $\mathbb{R}^{|J'|}$. Since f_J is interval separable by the previous lemma, this shows that f is as well.

For closure under composition, let $f : \mathbb{R}^m \to \mathbb{R}$ and $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$ for $i \in \{1, \dots, m\}$, all in $\mathcal{F}^*_{\text{subm}}$. The composition $g := f \circ (f_1 \times \cdots \times f_m)$ is clearly C^1 again, so we just have to check the property of the partial derivatives. By the chain rule, the partial derivatives of the composition are

$$\partial_i g(x_1, \dots, x_m) = \partial_j f(f_1(x_1), \dots, f_m(x_m)) \partial_k f_j(x_{j1}, \dots, x_{jn_j})$$

for some $j \in \{1, \dots m\}$ and $k \in \{1, \dots n_j\}$, and for all $x_1 \in \mathbb{R}^{n_1}, \dots, x_m \in \mathbb{R}^{n_m}$. So if this partial derivative is zero, there are two cases. First, if $\partial_j f(f_1(x_1), \dots, f_m(x_m)) = 0$ then f must be constant in its j-th argument (because $f \in \mathcal{F}^*_{\text{subm}}$) and thus g is constant in x_j, and in particular the i-th argument (which is an entry of x_j). Second, if $\partial_k f_j(x_{j1}, \dots, x_{jn_j}) = 0$ then f_j must be constant in its k-th argument (because $f_j \in \mathcal{F}^*_{\text{subm}}$) and thus g is constant in its corresponding i-th argument as well. This proves $g \in \mathcal{F}^*_{\text{subm}}$, as desired. □

Note that exp, sinh, arctan, n-th roots for n odd, and all linear functions are in $\mathcal{F}^*_{\text{subm}}$. So this is a useful set of primitive functions already. Unfortunately, it does not include

multiplication because the gradient of $(x, y) \mapsto xy$ is zero at $(0, 0)$. To fix this issue, we need to restrict the domain.[15]

For simplicity, we required primitive functions to be defined on all of $\mathbb{R}^n$. Suppose we allow for primitive functions to be defined only on an open subset of $\mathbb{R}^n$, and applying them to a value outside their domain is disallowed in SPCF programs. Then we can also include multiplication (on $\mathbb{R}^2 \setminus \{(0,0)\}$), logarithms (on $(0, \infty)$), non-constant univariate polynomials (on the complement of their stationary points), quantile functions of continuous distributions with nonzero density (on $(0, 1)$), and probability density functions (on the complement of their stationary points). The fact that some points in the domain are missing is inconvenient for functions that can be continuously extended to the these points, but one can work around this in a program by checking for the points that are not in the domain and returning the function values as constants for those cases.

C.4 Completeness Proof

This section uses the definitions of *boxwise continuity* and *interval separability* from Section 4.2. As discussed there, we assume that for each symbolic path $\Psi = (\mathcal{V}, n, \Delta, \Xi)$, we have that $\mathcal{V}$, each C with $C \bowtie 0 \in \Delta$, and each $\mathcal{W} \in \Xi$ contains each sample variable α_i at most once (Assumption 1). We also assume that the primitive functions are boxwise continuous, interval separable and closed under composition (Assumption 2). Furthermore, we say that $\mathcal{T}$ is a ***subdivision*** of $\boldsymbol{t}$ if $\mathcal{T}$ is compatible and $\bigcup_{\boldsymbol{t}' \in \mathcal{T}} (\!|\boldsymbol{t}'|\!) = (\!|\boldsymbol{t}|\!)$.

Theorem 4.3 (Completeness of interval approximations)**.** *Let $I \in \mathbb{I}$ and $\vdash P : \boldsymbol{R}$ be an almost surely terminating program satisfying the two assumptions discussed above. Then, for all $\epsilon > 0$, there is a countable set of interval traces $\mathcal{T} \subseteq \mathbb{T}_{\mathbb{I}}$ that is compatible and exhaustive such that*

$$\mathsf{upperBd}_P^{\mathcal{T}}(I) - \epsilon \leq [\![P]\!](I) \leq \mathsf{lowerBd}_P^{\mathcal{T}}(I) + \epsilon.$$

Proof. First, we give a brief outline of how the proof works. The idea is to cover $\{\boldsymbol{s} \in \mathbb{T} \mid \mathsf{val}_P(\boldsymbol{s}) \in I\}$ using boxes (interval traces). We can achieve this using symbolic execution: for a fixed path through the program, the result value is just a composition of primitive functions applied to the samples. Similarly, the weight function is a product of such functions, hence boxwise continuous. By passing to smaller boxes, we can assume that it is continuous on each box. In order to approximate the integral of the weight function, we use Riemann sums (as used in the definition of the Riemann integral). We partition the domain into smaller and smaller boxes such that the lower bound and the upper bound of the weight function come arbitrarily close (by continuity). Then by properties of the Riemann integral, the bounds arising from the interval traces representing the boxes in this partition converge to the desired integral of the weight function. The details of the proof are as follows.

Step 1: approximating the branching inequalities. Let $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ a symbolic path of P. To find a countable set $\mathcal{T}_\Psi \subseteq \mathbb{T}_{\mathbb{I}}$ such that $\bigcup_{\boldsymbol{t} \in \mathcal{T}_\Psi} (\!|\boldsymbol{t}|\!) \Subset \mathsf{Sat}_n(\Delta)$. Note that $\mathsf{Sat}_n(\Delta)$ is a finite intersection of sets of the form $\{\boldsymbol{s} \in [0,1]^n \mid C[\boldsymbol{s}/\overline{\alpha}] \bowtie 0\}$ where $\bowtie \in \{\leq, >\}$. In the $\leq$ case, we can write this constraint as $C[\boldsymbol{s}/\overline{\alpha}] \in \bigcup_{n \in \mathbb{N}} [-n, 0]$ and in the $>$ case as $C[\boldsymbol{s}/\overline{\alpha}] \in \bigcup_{n \in \mathbb{N}} [1/n, n]$. By applying Lemma C.8 to each of the compact intervals in these unions, we obtain a countable union of boxes that is a tight subset of $\{\boldsymbol{s} \in [0,1]^n \mid C[\boldsymbol{s}/\overline{\alpha}] \bowtie 0\}$. Since the intersection of two boxes is a box and since $\mathsf{Sat}_n(\Delta)$ is a finite intersection of such countable unions of boxes, it can be rewritten as a countable union of boxes. This yields $\mathcal{T}_\Psi$, such that $\bigcup_{\boldsymbol{t} \in \mathcal{T}_\Psi} (\!|\boldsymbol{t}|\!) \Subset \mathsf{Sat}_n(\Delta)$.

Step 2: handling the result value. By applying Lemma C.8 and intersecting the obtained interval traces with $\mathcal{T}_\Psi$, we obtain a countable set $\mathcal{T}'_{\Psi,I} \subseteq \mathbb{T}_{\mathbb{I}}$ such that $\bigcup_{\boldsymbol{t} \in \mathcal{T}'_{\Psi,I}} (\!|\boldsymbol{t}|\!) \Subset \{\boldsymbol{s} \in \mathsf{Sat}(\Delta) \mid \mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \in I\}$. By the same lemma, we find a countable set $\mathcal{T}'_{\Psi,I^c} \subseteq \mathbb{T}_{\mathbb{I}}$ such that $\bigcup_{\boldsymbol{t} \in \mathcal{T}'_{\Psi,I^c}} (\!|\boldsymbol{t}|\!) \Subset \{\boldsymbol{s} \in \mathsf{Sat}(\Delta) \mid \mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \notin I\}$ because the complement of I can be written as a countable union of intervals. By Lemma C.10, we find subdivisions $\mathcal{T}_{\Psi,I}$ and $\mathcal{T}_{\Psi,I^c}$ that even satisfy $\bigcup_{\boldsymbol{t} \in \mathcal{T}_{\Psi,I}} (\!|\boldsymbol{t}|\!) \Subset \mathsf{val}_P^{-1}(I) \cap \mathsf{Sat}(\Delta)$ and $\bigcup_{\boldsymbol{t} \in \mathcal{T}_{\Psi,I^c}} (\!|\boldsymbol{t}|\!) \Subset \mathsf{val}_P^{-1}(\mathbb{R} \setminus I) \cap \mathsf{Sat}(\Delta)$. By Lemma C.7, we can assume that the interval traces $\mathcal{T}_{\Psi,I}$ are almost disjoint. Because of the almost sure termination assumption, the set of traces $\bigcup_{(\mathcal{V},n,\Delta,\Xi)} \mathsf{Sat}(\Delta) = \mathsf{val}_P^{-1}(\mathbb{R})$ where the union ranges over all symbolic paths of P has measure 1. As a consequence, $\bigcup_{\Psi \in symPaths(P,0,\emptyset,\emptyset)} (\mathcal{T}_{\Psi,I} \cup \mathcal{T}_{\Psi,I^c})$ is a compatible and exhaustive set of interval traces. Now

$$\begin{aligned}
[\![P]\!](I) &= \sum_{(\mathcal{V},n,\Delta,\Xi)} [\![(\mathcal{V}, n, \Delta, \Xi)]\!](I) \\
&= \sum_{(\mathcal{V},n,\Delta,\Xi)} \int_{\mathsf{Sat}_n(\Delta)} [\mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \in I] \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \, \mathrm{d}\boldsymbol{s} \\
&= \sum_{(\mathcal{V},n,\Delta,\Xi)} \sum_{\boldsymbol{t} \in \mathcal{T}_{(\mathcal{V},n,\Delta,\Xi),I}} \int_{(\!|\boldsymbol{t}|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \, \mathrm{d}\boldsymbol{s}
\end{aligned}$$

where the outer sum ranges over the symbolic paths $(\mathcal{V}, n, \Delta, \Xi) \in symPaths(P, 0, \emptyset, \emptyset)$. The first equality holds by Theorem 6.1, the second one by Lemma B.1 and the last one by the construction of $\mathcal{T}_{(\mathcal{V},n,\Delta,\Xi),I}$.

Step 3: approximating the weight function. Let

$$\mathcal{T}' := \bigcup_{\Psi \in symPaths(P,0,\emptyset,\emptyset)} \mathcal{T}_{\Psi,I}.$$

For each $\boldsymbol{t} \in \mathcal{T}'$, fix some $\epsilon_{\boldsymbol{t}} > 0$, such that $\sum_{\boldsymbol{t} \in \mathcal{T}'} \epsilon_{\boldsymbol{t}} = \epsilon$. This can be achieved, for example, by enumerating $\mathcal{T}'$ as $\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, \ldots$ and choosing $\epsilon_{\boldsymbol{t}^{(i)}} = 2^{-i}\epsilon$. By Lemma C.9, we

[15] Handling these issues at the level of primitive functions directly (without restricting the domain) seems challenging: even if a function has only one point with zero gradient, e.g. multiplication, its preimage under other primitive functions can become very complicated. We tried to handle this by allowing the primitive functions to be submersions except on a null set given by a union of lower-dimensional manifolds. However, the preimages of such manifolds need not be manifolds again. Hence it seems difficult to come up with a broader class of primitive functions satisfying the assumptions without restricting the domain.

can find for each $\boldsymbol{t} \in \mathcal{T}'$ a countable set $\mathcal{S}_{\boldsymbol{t}}$ of interval traces such that:

$$\begin{aligned}
&\sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \operatorname{vol}(\boldsymbol{t}') \sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] - \epsilon_{\boldsymbol{t}}/2 \\
&\leq \int_{(\!|\boldsymbol{t}|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \, \mathrm{d}\boldsymbol{s} \\
&\leq \sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \operatorname{vol}(\boldsymbol{t}') \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] + \epsilon_{\boldsymbol{t}}/2
\end{aligned}$$

Next, choose $\epsilon_{\boldsymbol{t}'} > 0$ for each $\boldsymbol{t}'$ in such a way that $\sum_{\boldsymbol{t}' \in \mathcal{S}'_{\boldsymbol{t}}} \epsilon_{\boldsymbol{t}'} < \epsilon_{\boldsymbol{t}}/2$. By Lemma C.11, we can find for each $\boldsymbol{t}'$ a finite subdivision $\mathcal{S}'_{\boldsymbol{t}'}$ such that for all $\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}$, we have

$$\begin{aligned}
\sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}''|\!)} \operatorname{wt}_P(\boldsymbol{s}) &= \sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}''|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \geq \sup \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') - \epsilon_{\boldsymbol{t}'} \\
\min_{\boldsymbol{s} \in (\!|\boldsymbol{t}''|\!)} \operatorname{wt}_P(\boldsymbol{s}) &= \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}''|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \leq \min \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') + \epsilon_{\boldsymbol{t}'}.
\end{aligned}$$

Multiplying by $\operatorname{vol}(\boldsymbol{t}'')$ and summing over all $\boldsymbol{t}''$, we find together with the previous inequality

$$\begin{aligned}
&\sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'') \sup \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') - \epsilon_{\boldsymbol{t}} \\
&\leq \sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'')(\sup \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') - \epsilon_{\boldsymbol{t}'}) - \epsilon_{\boldsymbol{t}}/2 \\
&\leq \int_{(\!|\boldsymbol{t}|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \, \mathrm{d}\boldsymbol{s} \\
&\leq \sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'')(\min \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') + \epsilon_{\boldsymbol{t}'}) + \epsilon_{\boldsymbol{t}}/2 \\
&\leq \sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'') \min \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') + \epsilon_{\boldsymbol{t}}
\end{aligned}$$

because $\sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'') \leq 1$ and thus the contribution of all the $\epsilon_{\boldsymbol{t}'}$ is at most $\epsilon_{\boldsymbol{t}}/2$.

Overall, the desired trace set is given by

$$\mathcal{T} \coloneqq \bigcup_{\Psi = (\mathcal{V}, n, \Delta, \Xi)} \left(\mathcal{T}_{\Psi, I^c} \cup \bigcup_{\boldsymbol{t} \in \mathcal{T}_{\Psi, I}} \bigcup_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \mathcal{S}'_{\boldsymbol{t}'} \right),$$

is compatible and exhaustive because it is a subdivision of $\mathcal{T}_{\Psi,I^c}$ and $\mathcal{T}_{\Psi,I}$. By construction, we have $\operatorname{val}^{\mathbb{I}}_P(\boldsymbol{t}) \subseteq I$ for $\boldsymbol{t} \in \mathcal{T}_{\Psi,I}$ and $\operatorname{val}^{\mathbb{I}}_P(\boldsymbol{t}) \cap I = \emptyset$ for $\boldsymbol{t} \in \mathcal{T}_{\Psi,I^c}$. Hence the $\mathcal{T}_{\Psi,I^c}$-summands vanish in the sum for the bounds and we obtain

$$\begin{aligned}
&\operatorname{upperBd}^{\mathcal{T}}_P(I) - \epsilon \\
&= \sum_{\boldsymbol{t} \in \mathcal{T}} \operatorname{vol}(\boldsymbol{t})(\sup \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}))[\operatorname{val}^{\mathbb{I}}_P(\boldsymbol{t}) \cap I \neq \emptyset] - \epsilon \\
&= \sum_{\Psi = (\mathcal{V}, n, \Delta, \Xi)} \sum_{\boldsymbol{t} \in \mathcal{T}_{\Psi, I}} \\
&\qquad \left(\sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'') \sup \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') - \epsilon_{\boldsymbol{t}} \right) \\
&\leq \underbrace{\sum_{\Psi = (\mathcal{V}, n, \Delta, \Xi)} \sum_{\boldsymbol{t} \in \mathcal{T}_{\Psi, I}} \int_{(\!|\boldsymbol{t}|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\mathbf{s}/\overline{\alpha}] \, \mathrm{d}\mathbf{s}}_{= [\![P]\!](I)} \\
&\leq \sum_{\Psi = (\mathcal{V}, n, \Delta, \Xi)} \sum_{\boldsymbol{t} \in \mathcal{T}_{\Psi, I}} \\
&\qquad \left(\sum_{\boldsymbol{t}' \in \mathcal{S}_{\boldsymbol{t}}} \sum_{\boldsymbol{t}'' \in \mathcal{S}'_{\boldsymbol{t}'}} \operatorname{vol}(\boldsymbol{t}'') \min \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}'') + \epsilon_{\boldsymbol{t}} \right) \\
&= \sum_{\boldsymbol{t} \in \mathcal{T}} \operatorname{vol}(\boldsymbol{t})(\min \operatorname{wt}^{\mathbb{I}}_P(\boldsymbol{t}))[\operatorname{val}^{\mathbb{I}}_P(\boldsymbol{t}) \subseteq I] + \epsilon \\
&= \operatorname{lowerBd}^{\mathcal{T}}_P(I) + \epsilon. \qquad \square
\end{aligned}$$

Lemma C.7. *Given a countable set of interval traces $\mathcal{T} \subseteq \mathbb{I}^n$, there is a countable set of interval traces $\mathcal{T}' \subseteq \mathbb{I}^n$ that is compatible and satisfies $\bigcup_{\boldsymbol{t} \in \mathcal{T}} (\!|\boldsymbol{t}|\!) = \bigcup_{\boldsymbol{t} \in \mathcal{T}'} (\!|\boldsymbol{t}|\!)$.*

Proof. Let $A : \mathbb{N} \to \mathcal{T}$ be an enumeration. Define $A' : \mathbb{N} \to \Sigma_{\mathbb{R}^n}$ by $m \mapsto \overline{A(m) \setminus \bigcup_{i=0}^{m-1} A(i)}$ where $\overline{S}$ denotes the closure of S. Then the collection $\{A'(m) \mid m \in \mathbb{N}\}$ is pairwise almost disjoint, and each $A'(m)$ can be written as a finite union of boxes, proving the claim. $\square$

Lemma C.8. *Let $\mathcal{V}$ a symbolic value of ground type containing each sample variable $\alpha_1, \ldots, \alpha_n$ at most once and $[x, y]$ an interval. Then there is a countable set of pairwise disjoint interval traces $\mathcal{T} \subset \mathbb{I}^n_{[0,1]}$ such that*

$$\bigcup_{\boldsymbol{t} \in \mathcal{T}} (\!|\boldsymbol{t}|\!) \Subset \{\boldsymbol{s} \in [0,1]^n \mid \mathcal{V}[\boldsymbol{s}/\overline{\alpha}] \in [x, y]\}.$$

Proof. If $\mathcal{V}$ is of ground type, then it is simply a composition of primitive functions applied to sample variables and literals. Since the set of primitive functions is closed under composition and since no sample variable occurs twice, this composition is still an interval separable function f of the sample variables. By definition of interval separability, there is a countable set of interval traces $\mathcal{J}$ such that $\bigcup_{\boldsymbol{t} \in \mathcal{J}} (\!|\boldsymbol{t}|\!) \Subset f^{-1}([x, y])$, as desired. $\square$

Lemma C.9. *Let $\boldsymbol{t} \in \mathbb{I}^n$ be an interval trace and Ξ a set of symbolic values with sample variables from $\overline{\alpha} = \alpha_1, \ldots, \alpha_n$. Then for any $\epsilon > 0$, there is a countable subdivision $\mathcal{T}$ of $\boldsymbol{t}$ such that*

$$\begin{aligned}
&\sum_{\boldsymbol{t}' \in \mathcal{T}} \operatorname{vol}(\boldsymbol{t}') \sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] - \epsilon \\
&\leq \int_{(\!|\boldsymbol{t}|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \, \mathrm{d}\boldsymbol{s} \\
&\leq \sum_{\boldsymbol{t}' \in \mathcal{T}} \operatorname{vol}(\boldsymbol{t}') \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] + \epsilon.
\end{aligned}$$

Proof. Values are simply boxwise continuous functions applied to the sample variables. Intersecting the boxes for each $\mathcal{W} \in \Xi$, we see that the function

$$f : (\!|\boldsymbol{t}|\!) \to \mathbb{R}, \quad \boldsymbol{s} \mapsto \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}]$$

is boxwise continuous. We can thus find a countable subdivision $\mathcal{T}_{\text{cont}}$ of $\boldsymbol{t}$ such that f is continuous on each $\boldsymbol{t}' \in \mathcal{T}_{\text{cont}}$.

Since we can sum over the $\boldsymbol{t}' \in \mathcal{T}_{\text{cont}}$, it suffices to prove that each integral $\int_{(\!|\boldsymbol{t}'|\!)} f(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$ can be approximated arbitrarily closely. Note that each such integral is finite because a continuous function is bounded on a compact set and the measure of $(\!|\boldsymbol{t}'|\!)$ is finite. But then such approximations are given by Riemann sums, i.e. the sums that are used to define the Riemann integral. As a concrete example, one can consider the subdivision $\mathcal{T}_m$ of $\boldsymbol{t}'$ in m equidistant sections in each dimension (consisting of m^n parts overall). Then $\sum_{\boldsymbol{t}'' \in \mathcal{T}_m} \mathrm{vol}(\boldsymbol{t}'') \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}''|\!)} f(\boldsymbol{s})$ converges to the Riemann integral $\int_{(\!|\boldsymbol{t}'|\!)} f(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$ as $m \to \infty$ (and similarly for the supremum). Since it is known that the Riemann integral and the Lebesgue integral have the same value for continuous functions on a Cartesian product of compact intervals, the claim follows immediately. □

Lemma C.10 (Relationship between symbolic execution and interval semantics)**.** *Let $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ be a symbolic path of P and $\boldsymbol{t}$ an interval trace with $(\!|\boldsymbol{t}|\!) \subseteq \mathrm{Sat}_n(\Delta)$. Suppose furthermore that all the symbolic values contain each of the sample variables $\overline{\alpha} = \alpha_1, \ldots, \alpha_n$ at most once. Then there is a subdivision $\mathcal{T}$ of $\boldsymbol{t}$ such that for all $\boldsymbol{t}' \in \mathcal{T}$, the interval semantics for the value is precise:*

$$\mathrm{val}_P^{\mathbb{I}}(\boldsymbol{t}') = \{\mathrm{val}_P(\boldsymbol{s}) \mid \boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)\}.$$

For each symbolic score value $\mathcal{W} \in \Xi$, let $[\mathcal{W}_{\boldsymbol{t}'}^-, \mathcal{W}_{\boldsymbol{t}'}^+]$ be its interval approximation, i.e. $\mathcal{W}_{\boldsymbol{t}'}^- = \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}]$ and $\mathcal{W}_{\boldsymbol{t}'}^+ = \sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}]$. Then

$$\mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}') = \left[\prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}'}^-, \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}'}^+\right].$$

Proof. The symbolic value $\mathcal{V}$ is a composition of primitive functions applied to α's. Hence the are boxwise continuous functions of the α's. We pick a suitable subdivision $\mathcal{T}$ such that all these functions are continuous when restricted to any $\boldsymbol{t}' \in \mathcal{T}$. For any such function f, we have

$$f_{\mathbb{I}}([x_1, y_1], \ldots, [x_m, y_m]) = [\inf F, \sup F]$$

where $F := f([x_1, y_1] \times \cdots \times [x_m, y_m]) \subseteq \mathbb{R}$, by definition. Then continuity implies that the image of any box is a compact and path-connected subset of $\mathbb{R}$, i.e. an interval. Hence we even have $f_{\mathbb{I}}([x_1, y_1], \ldots, [x_m, y_m]) = F$, i.e. the image of any box equals its interval approximation, proving the claim about the value semantics.

For the interval semantics of the weight, note that the previous argument applies to every symbolic score value $\mathcal{W} \in \Xi$, proving that

$$\{\mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \mid \boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)\} = [\mathcal{W}_{\boldsymbol{t}'}^-, \mathcal{W}_{\boldsymbol{t}'}^+].$$

Since the interval semantics multiplies the interval approximation of each score value in interval arithmetic, this implies the claim. □

Note that the interval approximation of the weight is imprecise in the following sense:

$$\mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}') \neq \left\{ \prod_{\mathcal{W} \in \Xi} \mathcal{W}[\boldsymbol{s}/\overline{\alpha}] \;\middle|\; \boldsymbol{s} \in (\!|\boldsymbol{t}'|\!) \right\}.$$

As an example, if $\Xi = \{\alpha_1, 1 - \alpha_1\}$ and $\boldsymbol{t}'' = \langle [0,1] \rangle$ then the left-hand side is $[0,1]$ because each of the weights is approximated by $[0,1]$, but the right-hand side is $[0, 1/4]$ because the function $\alpha_1(1-\alpha_1)$ attains its maximum at $1/4$, not 1.

Lemma C.11. *Let $\Psi = (\mathcal{V}, n, \Delta, \Xi)$ be a symbolic path of P and $\boldsymbol{t}$ an interval trace with $(\!|\boldsymbol{t}|\!) \subseteq \mathrm{Sat}_n(\Delta)$. Suppose furthermore that all the symbolic values contain each of the sample variables $\overline{\alpha} = \alpha_1, \ldots, \alpha_n$ at most once. Then for all $\epsilon > 0$, there is a subdivision $\mathcal{T}$ of $\boldsymbol{t}$ such that for all $\boldsymbol{t}' \in \mathcal{T}$, we have $\min_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \mathrm{wt}_P(\boldsymbol{s}) \leq \min \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}') + \epsilon$ and $\sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)} \mathrm{wt}_P(\boldsymbol{s}) \geq \sup \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}') - \epsilon$.*

Proof. Since for each $\mathcal{W} \in \Xi$, the function $f : (\!|\boldsymbol{t}|\!) \to \mathbb{R}, \boldsymbol{s} \mapsto \mathcal{W}[\boldsymbol{s}/\overline{\alpha}]$ is boxwise continuous (a property of primitive functions), we can find a countable subdivision $\mathcal{T}'$ of $\boldsymbol{t}$, such that for all $\boldsymbol{t}' \in \mathcal{T}'$, f is continuous on $(\!|\boldsymbol{t}'|\!)$. Hence it suffices to prove the statement for each $\boldsymbol{t}'$.

Since $(\!|\boldsymbol{t}'|\!)$ is compact (because it's closed and bounded), f attains a maximum $W < \infty$ on $(\!|\boldsymbol{t}'|\!)$ and is even uniformly continuous on $\boldsymbol{t}'$. Hence there is a $\delta > 0$ such that whenever $||\boldsymbol{s} - \boldsymbol{s}'|| < \delta$ then $|f(\boldsymbol{s}) - f(\boldsymbol{s}')| < \epsilon' := \frac{\epsilon}{W^{|\Xi|-1}}$.

Let $\mathcal{T}$ be a subdivision where each interval trace $\boldsymbol{t} \in \mathcal{T}$ has diameter less than δ. For $\boldsymbol{t} \in \mathcal{T}$ and $\mathcal{W} \in \Xi$, let $\mathcal{W}_{\boldsymbol{t}}^- := \min_{\boldsymbol{s} \in (\!|\boldsymbol{t}|\!)} \mathcal{W}[\boldsymbol{s}/\alpha]$ and $\mathcal{W}_{\boldsymbol{t}}^+ := \sup_{\boldsymbol{s} \in (\!|\boldsymbol{t}|\!)} \mathcal{W}[\boldsymbol{s}/\alpha]$. By the choice of $\mathcal{T}$, we have $\mathcal{W}_{\boldsymbol{t}}^+ \leq \mathcal{W}_{\boldsymbol{t}}^- + \epsilon'$ for $\boldsymbol{t} \in \mathcal{T}$. By Lemma C.10, we find that $\sup \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}) = \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}}^+$ and $\min \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}) = \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}}^-$. As a consequence, we have

$$\begin{aligned}
\sup \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}) - \min \mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t}) &= \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}}^+ - \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}}^- \\
&< \prod_{\mathcal{W} \in \Xi} (\mathcal{W}_{\boldsymbol{t}}^- + \epsilon') - \prod_{\mathcal{W} \in \Xi} \mathcal{W}_{\boldsymbol{t}}^- \\
&< \epsilon' W^{|\Xi|-1} = \epsilon
\end{aligned}$$

So the interval $\mathrm{wt}_P^{\mathbb{I}}(\boldsymbol{t})$ has diameter less than ϵ. Since the interval $\{\mathrm{wt}_P(\boldsymbol{s}) \mid \boldsymbol{s} \in (\!|\boldsymbol{t}'|\!)\}$ is contained in it (by soundness), the claim follows. □

Corollary 4.4. *Let $I \in \mathbb{I}$ and $\vdash P : \boldsymbol{R}$ be as in Theorem 4.3. There is a sequence of finite, compatible sets of interval traces $\mathcal{T}_1, \mathcal{T}_2, \ldots \subseteq \mathbb{T}_{\mathbb{I}}$ s.t. $\lim_{n \to \infty} \mathrm{lowerBd}_P^{\mathcal{T}_n}(I) = [\![P]\!](I)$.*

Proof. By Theorem 4.3, we can find for each $n \in \mathbb{N}$ a set of interval traces $\mathcal{T}_n'$ such that $\mathrm{lowerBd}_P^{\mathcal{T}_n'}(I) > [\![P]\!](I) - 1/n$. Since the lower bound is defined as a sum over $\mathcal{T}_n'$, there is a finite subset $\mathcal{T}_n$ such that $\mathrm{lowerBd}_P^{\mathcal{T}_n}(I) > [\![P]\!](I) - 2/n$. Since $\mathcal{T}_n'$ is still compatible, the soundness result yields $\mathrm{lowerBd}_P^{\mathcal{T}_n}(I) \leq [\![P]\!](I)$, implying the claim. □

Finitely many interval traces are not enough for complete upper bounds, if the weight function is unbounded. This issue arises even if we can compute the tightest possible bounds on the weight function, as the following program illustrates.

Example C.3. Consider the following probabilistic program expressed in pseudocode.

threshold := 1
while (sample ≤ *threshold*) **do**
  threshold := $\frac{\textit{threshold}}{2}$
  score(2)

The program only requires addition and scalar multiplication. It can even be implemented using call-by-name (CbN) semantics (which allows each sampled value to be used at most once). For example in SPCF we can write

$$P \equiv \left(\mu_s^{\varphi}.\, \mathsf{if}(\mathsf{sample} - s, \mathsf{score}(2); \varphi(s/2), 1)\right) 1$$

The program P has the weight function

$$\mathrm{wt}_P(\langle t_0, \dots, t_n\rangle) = \begin{cases} 2^n & \text{if } t_n > 2^{-n} \wedge \\ & \quad \forall i \in \{0, \dots, n-1\} : t_i \le 2^{-i} \\ 0 & \text{otherwise.} \end{cases}$$

P is integrable because the normalizing constant is

$$Z = \int_{\mathbb{T}} \mathrm{wt}_P(\boldsymbol{s})\, \mathrm{d}\boldsymbol{s} = \sum_{n=1}^{\infty} 2^n \times (1 - 2^{-n}) \prod_{i=0}^{n-1} 2^{-i}$$
$$= \sum_{n=1}^{\infty} 2^n (1 - 2^{-n}) 2^{-n(n-1)/2} < \infty.$$

We claim that P requires infinitely many interval traces for the upper bound to converge to the true denotation. Define the sets of traces T_n for $n \ge 1$ by

$$T_n = [0, 2^0] \times [0, 2^{-1}] \times \cdots \times [0, 2^{-n+1}] \times (2^{-n}, 1].$$

Suppose we are given an arbitrary finite exhaustive set of interval traces. This set needs to cover all of the T_n's, so one interval trace, say $\boldsymbol{t}$, must cover infinitely many T_n's. Since $\mathrm{wt}_P(\boldsymbol{s}) = 2^n$ for $\boldsymbol{s} \in T_n$, the weight function on $\boldsymbol{t}$ is unbounded. Therefore, the only possible upper bound for $\boldsymbol{t}$ is ∞, *even if our semantics could compute the set* $\{\mathrm{wt}_P(\boldsymbol{s}) \mid \boldsymbol{s} \in (\!|\boldsymbol{t}|\!)\}$ *exactly*. Hence any finite exhaustive interval trace has upper bound ∞, while the true denotation is finite. As we have seen, this is not because of imprecision of interval analysis, but an inherent problem if the weight function is unbounded. So we cannot hope for complete upper bounds with finitely many interval traces.

D Supplementary Material for Section 5

We provide additional proofs and material for Section 5. To have access to named rules, we give the type system in Fig. 9 which agrees with the one in Fig. 4 in everything but the labels.

D.1 Soundness

To show soundness (Theorem 5.1), we establish a (weight-aware) subject reduction property for our type system as follows. For an interval $[a, b] \in \mathbb{I}$ and $r \in \mathbb{R}_{\ge 0}$, we define $r \cdot [a, b] := [r \cdot a, r \cdot b]$. To simplify notation, we use a modified transition relation that omits the concrete trace (which is irrelevant in Theorem 5.1). We write $P \to_w P'$ if $(P, \boldsymbol{s}, 1) \to (P', \boldsymbol{s}', w)$ for some $w \in \mathbb{R}$ and $\boldsymbol{s}, \boldsymbol{s}' \in \mathbb{T}$. Note that we could define $\to_w$ as a dedicated reduction system by adapting the rules from $\to$ in Fig. 2.

Lemma D.1 (Substitution). *If* $\Gamma; \{x_i : \sigma_i\}_{i=1}^n \vdash P : \mathcal{A}$ *for distinct variables* x_i *and if* $\Gamma \vdash M_i : \left\{\begin{smallmatrix}\sigma_i \\ \mathbf{1}\end{smallmatrix}\right\}$ *for all* $i \in \{1, \dots, n\}$ *then* $\Gamma \vdash P[M_i/x_i]_{i=1}^n : \mathcal{A}$.

Proof. By a standard induction on M. □

Lemma D.2 (Weighted Subject Reduction). *Let* P *be any program such that* $\vdash P : \left\{\begin{smallmatrix}\sigma \\ J\end{smallmatrix}\right\}$ *and* $P \to_w P'$ *for some* $w > 0$. *Then* $\vdash P' : \left\{\begin{smallmatrix}\sigma \\ \frac{1}{w} \cdot J\end{smallmatrix}\right\}$.

Proof. We prove this by induction on the structure of P.

Case $P = \mathsf{sample}$**:** then $P \to_1 \underline{r}$ for some $r \in [0, 1]$. As multiple consecutive application of (Sub) can be replaced by a single one since ($\sqsubseteq_{\mathcal{A}}$ is transitive), we can assume w.l.o.g. that the last step in $\vdash P : \left\{\begin{smallmatrix}\sigma \\ J\end{smallmatrix}\right\}$ was:

$$\dfrac{\dfrac{}{\vdash \mathsf{sample} : \left\{\begin{smallmatrix}[0,1] \\ \mathbf{1}\end{smallmatrix}\right\}}\,(\textsc{Sample})}{\vdash \mathsf{sample} : \left\{\begin{smallmatrix}I \\ J\end{smallmatrix}\right\}}\,(\textsc{Sub})$$

By subtyping we have $[0, 1] \sqsubseteq I$ and $\mathbf{1} \sqsubseteq J$. So $[r, r] \sqsubseteq I$ and we can type $\vdash \underline{r} : \left\{\begin{smallmatrix}I \\ J\end{smallmatrix}\right\}$ using (Lit) and (Sub) as required.

Case $P = f(\underline{r_1}, \dots, \underline{r_{|f|}})$**:** then $P \to_1 \underline{f(r_1, \dots, r_{|f|})}$. W.l.o.g., we can assume that the last step in $\vdash P : \left\{\begin{smallmatrix}\sigma \\ J\end{smallmatrix}\right\}$ was

$$\dfrac{\dfrac{\dfrac{}{\vdash \underline{r_1} : \left\{\begin{smallmatrix}[r_1,r_1] \\ \mathbf{1}\end{smallmatrix}\right\}}\,(\textsc{Lit})}{\vdash \underline{r_1} : \left\{\begin{smallmatrix}I_1 \\ J_1\end{smallmatrix}\right\}}\,(\textsc{Sub}) \quad \cdots \quad \dfrac{\dfrac{}{\vdash \underline{r_{|f|}} : \left\{\begin{smallmatrix}[r_{|f|},r_{|f|}] \\ \mathbf{1}\end{smallmatrix}\right\}}\,(\textsc{Lit})}{\vdash \underline{r_{|f|}} : \left\{\begin{smallmatrix}I_{|f|} \\ J_{|f|}\end{smallmatrix}\right\}}\,(\textsc{Sub})}{\vdash f(\underline{r_1}, \dots, \underline{r_{|f|}}) : \left\{\begin{smallmatrix}f^{\mathbb{I}}(I_1, \dots, I_{|f|}) \\ (\times^{\mathbb{I}})_{i=1}^{|f|} J_i\end{smallmatrix}\right\}}\,(\textsc{Prim})}{\vdash f(\underline{r_1}, \dots, \underline{r_{|f|}}) : \left\{\begin{smallmatrix}I \\ J\end{smallmatrix}\right\}}\,(\textsc{Sub})$$

So by subtyping $r_i \in I_i$ and $1 \in J_i$ for all i. By definition of $f^{\mathbb{I}}$ we thus have $f(r_1, \dots, r_{|f|}) \in f^{\mathbb{I}}(I_1, \dots, I_{|f|})$ and (again by subtyping) we have $f(r_1, \dots, r_{|f|}) \in I$. Similarly, by definition of $\times^{\mathbb{I}}$ we have $1 \in (\times^{\mathbb{I}})_{i=1}^{|f|} J_i$ and thus $1 \in J$. We can type $\vdash \underline{f(r_1, \dots, r_{|f|})} : \left\{\begin{smallmatrix}I \\ J\end{smallmatrix}\right\}$ using (Lit) and (Sub) as required.

Case $P = \mathsf{if}(\underline{r}, M, N)$ **and** $r \le 0$**:** then $P \to_1 M$. W.l.o.g., we can assume that the last step in $\vdash P : \left\{\begin{smallmatrix}\sigma \\ J\end{smallmatrix}\right\}$ is

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \left\{\begin{matrix}\sigma\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Var}) \qquad \frac{\Gamma \vdash M : \mathcal{A} \quad \mathcal{A} \sqsubseteq_{\mathcal{A}} \mathcal{B}}{\Gamma \vdash M : \mathcal{B}}\ (\textsc{Sub}) \qquad \frac{\Gamma; x : \sigma \vdash M : \mathcal{A}}{\Gamma \vdash \lambda x.M : \left\{\begin{matrix}\sigma \to \mathcal{A}\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Abs}) \qquad \frac{\Gamma; \varphi : \sigma \to \mathcal{A}; x : \sigma \vdash M : \mathcal{A}}{\Gamma \vdash \mu_x^{\varphi}.M : \left\{\begin{matrix}\sigma \to \mathcal{A}\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Fix})$$

$$\frac{}{\Gamma \vdash \mathsf{sample} : \left\{\begin{matrix}[0,1]\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Sample}) \qquad \frac{\Gamma \vdash M : \left\{\begin{matrix}\sigma_1 \to \left\{\begin{matrix}\sigma_2\\ [e,f]\end{matrix}\right\}\\ [a,b]\end{matrix}\right\} \quad \Gamma \vdash N : \left\{\begin{matrix}\sigma_1\\ [c,d]\end{matrix}\right\}}{\Gamma \vdash MN : \left\{\begin{matrix}\sigma_2\\ [a,b] \times^{\mathbb{I}} [c,d] \times^{\mathbb{I}} [e,f]\end{matrix}\right\}}\ (\textsc{App})$$

$$\frac{}{\Gamma \vdash \underline{r} : \left\{\begin{matrix}[r,r]\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Lit}) \qquad \frac{\Gamma \vdash M : \left\{\begin{matrix}[\_,\_]\\ [a,b]\end{matrix}\right\} \quad \Gamma \vdash N : \left\{\begin{matrix}\sigma\\ [c,d]\end{matrix}\right\} \quad \Gamma \vdash P : \left\{\begin{matrix}\sigma\\ [c,d]\end{matrix}\right\}}{\Gamma \vdash \mathsf{if}(M,N,P) : \left\{\begin{matrix}\sigma\\ [a,b] \times^{\mathbb{I}} [c,d]\end{matrix}\right\}}\ (\textsc{If})$$

$$\frac{\Gamma \vdash M : \left\{\begin{matrix}[a,b]\\ [c,d]\end{matrix}\right\}}{\Gamma \vdash \mathsf{score}(M) : \left\{\begin{matrix}[a,b] \sqcap [0,\infty]\\ [c,d] \times^{\mathbb{I}} ([a,b] \sqcap [0,\infty])\end{matrix}\right\}}\ (\textsc{Score}) \qquad \frac{\Gamma \vdash M_1 : \left\{\begin{matrix}[a_1,b_1]\\ [c_1,d_1]\end{matrix}\right\} \quad \cdots \quad \Gamma \vdash M_{|f|} : \left\{\begin{matrix}[a_{|f|},b_{|f|}]\\ [c_{|f|},d_{|f|}]\end{matrix}\right\}}{\Gamma \vdash f(M_1,\dots,M_{|f|}) : \left\{\begin{matrix}f^{\mathbb{I}}([a_1,b_1],\dots,[a_{|f|},b_{|f|}])\\ (\times^{\mathbb{I}})_{i=1}^{|f|}[c_i,d_i]\end{matrix}\right\}}\ (\textsc{Prim})$$

Figure 9. Weight-aware interval type system for SPCF with typing rule names. The rules agree with those in Fig. 4.

$$\dfrac{\dfrac{\dfrac{}{\vdash \underline{r} : \left\{\begin{matrix}[r,r]\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Lit})}{\vdash \underline{r} : \left\{\begin{matrix}I\\ J''\end{matrix}\right\}}\ (\textsc{Sub}) \qquad \vdash M : \left\{\begin{matrix}\sigma'\\ J'\end{matrix}\right\} \qquad \vdash N : \left\{\begin{matrix}\sigma'\\ J'\end{matrix}\right\}}{\vdash \mathsf{if}(\underline{r},M,N) : \left\{\begin{matrix}\sigma'\\ J' \times^{\mathbb{I}} J''\end{matrix}\right\}}\ (\textsc{If})}{\vdash \mathsf{if}(\underline{r},M,N) : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}}\ (\textsc{Sub})$$

and we have $1 \in J''$, $\sigma' \sqsubseteq_\sigma \sigma$ and $J' \times^{\mathbb{I}} J'' \sqsubseteq J$ by subtyping. As $1 \in J''$, we get $J' \sqsubseteq J' \times^{\mathbb{I}} J''$. Thus we obtain $J' \sqsubseteq J$ and $\sigma' \sqsubseteq_\sigma \sigma$, and we can type $\vdash M : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}$ using (Sub).

Case $P = \mathsf{if}(\underline{r}, M, N)$ **and** $r > 0$**:** then $P \to_1 N$. Analogous to the previous case.

Case $P = \mathsf{score}(\underline{r})$ **and** $r \geq 0$**:** then $P \to_r \underline{r}$. W.l.o.g., we can assume that the last step in $\vdash P : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \underline{r} : \left\{\begin{matrix}[r,r]\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Lit})}{\vdash \underline{r} : \left\{\begin{matrix}I'\\ J'\end{matrix}\right\}}\ (\textsc{Sub})}{\vdash \mathsf{score}(\underline{r}) : \left\{\begin{matrix}I' \sqcap [0,\infty]\\ J' \times^{\mathbb{I}} (I' \sqcap [0,\infty])\end{matrix}\right\}}\ (\textsc{Score})}{\vdash \mathsf{score}(\underline{r}) : \left\{\begin{matrix}I\\ J\end{matrix}\right\}}\ (\textsc{Sub})$$

By subtyping we have $r \in I'$ and even $r \in I' \sqcap [0,\infty]$ because $r \geq 0$. Thus $r \in I$, again by subtyping. Similarly, $1 \in J'$ and by definition of $\times^{\mathbb{I}}$, we have $r \in J' \times^{\mathbb{I}} (I' \sqcap [0,\infty])$. Thus $r \in J$ by subtyping. This already implies $1 \in \frac{1}{r} \cdot J$ and we can thus type $\vdash \underline{r} : \left\{\begin{matrix}I\\ \frac{1}{r} \cdot J\end{matrix}\right\}$ by using (Lit) and (Sub).

Case $P = (\lambda x.M)V$**:** then $P \to_1 M[V/x]$. W.l.o.g., the last step in $\vdash P : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\{x : \tilde{\sigma}'\} \vdash M : \left\{\begin{matrix}\tilde{\sigma}''\\ \tilde{J}''\end{matrix}\right\}}{\vdash \lambda x.M : \left\{\begin{matrix}\tilde{\sigma}' \to \left\{\begin{matrix}\tilde{\sigma}''\\ \tilde{J}''\end{matrix}\right\}\\ \mathbf{1}\end{matrix}\right\}}\ (\textsc{Abs})}{\vdash \lambda x.M : \left\{\begin{matrix}\sigma' \to \left\{\begin{matrix}\sigma''\\ J''\end{matrix}\right\}\\ J'\end{matrix}\right\}}\ (\textsc{Sub}) \qquad \vdash V : \left\{\begin{matrix}\sigma'\\ J'''\end{matrix}\right\}}{\vdash (\lambda x.M)V : \left\{\begin{matrix}\sigma''\\ J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J'''\end{matrix}\right\}}\ (\textsc{App})}{\vdash (\lambda x.M)V : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}}\ (\textsc{Sub})$$

By subtyping we get that $\sigma' \sqsubseteq_\sigma \tilde{\sigma}'$. It is easy to see that we can also type $\vdash V : \left\{\begin{matrix}\sigma'\\ \mathbf{1}\end{matrix}\right\}$ because V is a value and we get $\vdash V : \left\{\begin{matrix}\tilde{\sigma}'\\ \mathbf{1}\end{matrix}\right\}$ by (Sub). Using Lemma D.1, we can thus type $\vdash M[V/x] : \left\{\begin{matrix}\tilde{\sigma}''\\ \tilde{J}''\end{matrix}\right\}$.

We have $1 \in J'$ by subyptying and as V is a value, it is easy to see that $1 \in J'''$. Hence $J'' \sqsubseteq J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J'''$. Also by subtyping, we find $\tilde{\sigma}'' \sqsubseteq_\sigma \sigma'' \sqsubseteq_\sigma \sigma$, $\tilde{J}'' \sqsubseteq J''$, and $J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J''' \sqsubseteq J$. This implies $\tilde{\sigma}'' \sqsubseteq_\sigma \sigma$ and $\tilde{J}'' \sqsubseteq J$. By subtyping $\vdash M[V/x] : \left\{\begin{matrix}\sigma\\ J\end{matrix}\right\}$, as required.

Case $P = (\mu_x^{\varphi}.M)V$**:** then $P \to_1 M[V/x, (\mu_x^{\varphi}.M)/\varphi]$. Analogous to the previous case for abstractions.

Case $P = E[P']$ for an evaluation context $E \neq [\cdot]$: then $P' \to_r P''$ and $P \to_r E[P'']$. All such cases follow easily by case analysis on E. As an example, consider the context $E = [\cdot]N$. In this situation, we have $P = E[P'] = P'N$ with $P' \to_r P''$, so $P \to_r P''N$. W.l.o.g., the last step in $\vdash P : \left\{\begin{smallmatrix}\sigma\\ J\end{smallmatrix}\right\}$ is

$$\dfrac{\dfrac{\vdash P' : \left\{\begin{smallmatrix}\sigma' \to \left\{\begin{smallmatrix}\sigma''\\ J''\end{smallmatrix}\right\}\\ J'\end{smallmatrix}\right\} \quad \vdash N : \left\{\begin{smallmatrix}\sigma'\\ J'''\end{smallmatrix}\right\}}{\vdash P'N : \left\{\begin{smallmatrix}\sigma''\\ J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J'''\end{smallmatrix}\right\}}\,(\textsc{App})}{\vdash P'N : \left\{\begin{smallmatrix}\sigma\\ J\end{smallmatrix}\right\}}\,(\textsc{Sub})$$

By the inductive assumption for $P' \to_r P''$, we get $\vdash P'' : \left\{\begin{smallmatrix}\sigma' \to \left\{\begin{smallmatrix}\sigma''\\ J''\end{smallmatrix}\right\}\\ \frac{1}{r} \cdot J'\end{smallmatrix}\right\}$ and can then type

$$\dfrac{\dfrac{\vdash P'' : \left\{\begin{smallmatrix}\sigma' \to \left\{\begin{smallmatrix}\sigma''\\ J''\end{smallmatrix}\right\}\\ \frac{1}{r} \cdot J'\end{smallmatrix}\right\} \quad \vdash N : \left\{\begin{smallmatrix}\sigma'\\ J'''\end{smallmatrix}\right\}}{\vdash P'N : \left\{\begin{smallmatrix}\sigma''\\ \frac{1}{r} \cdot J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J'''\end{smallmatrix}\right\}}\,(\textsc{App})}{\vdash P'N : \left\{\begin{smallmatrix}\sigma\\ \frac{1}{r} \cdot J\end{smallmatrix}\right\}}\,(\textsc{Sub})$$

because if $J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J''' \sqsubseteq J$ then $\frac{1}{r} \cdot J' \times^{\mathbb{I}} J'' \times^{\mathbb{I}} J''' \sqsubseteq \frac{1}{r} \cdot J$.

The proof for the other evaluation contexts, i.e., where $P = \mathsf{if}(P', M, N)$, $P = VP'$, $P = \mathsf{score}(P')$, or $P = \underline{f}(r_1, \ldots, r_{i-1}, P', N_{i+1}, \ldots, N_{|f|})$ for some $P' \to_r P''$, are all analogous to the above. □

Lemma D.3 (Zero-Weighted Subject Reduction). *Let P be any program such that $\vdash P : \left\{\begin{smallmatrix}\sigma\\ J\end{smallmatrix}\right\}$ and $P \to_0 P'$. Then*

1. *$\vdash P' : \left\{\begin{smallmatrix}\sigma\\ J'\end{smallmatrix}\right\}$ for some J', and*
2. *$0 \in J$*

Proof. The proof that $\vdash P' : \left\{\begin{smallmatrix}\sigma\\ J'\end{smallmatrix}\right\}$ for some J' is analogous to the proof of Lemma D.2 with fewer restrictions on the weight. The claim $0 \in J$ follows by observing that $P \to_0 P'$ is only possible if the redex in P is $\mathsf{score}(\underline{0})$. In case $P = \mathsf{score}(\underline{0})$, the claim follows directly from (Score). If $P = E[\mathsf{score}(\underline{0})]$, it is a simple induction on the structure of the evaluation context E. □

Theorem 5.1. *Let $\vdash P : \boldsymbol{R}$ be a simply-typed program. If $\vdash P : \left\{\begin{smallmatrix}[a,b]\\ [c,d]\end{smallmatrix}\right\}$ and $(P, \boldsymbol{s}, 1) \to^* (\underline{r}, \langle\rangle, w)$ for some $\boldsymbol{s} \in \mathbb{T}$ and $r, w \in \mathbb{R}$, then $r \in [a,b]$ and $w \in [c,d]$.*

Proof. Let

$$(P, \boldsymbol{s}, 1) = (P_0, \boldsymbol{s}_0, w_0) \to \cdots \to (P_n, \boldsymbol{s}_n, w_n) = (\underline{r}, \langle\rangle, w)$$

be the reduction sequence of $(P, \boldsymbol{s}, 1)$. By definition of $\to_w$ it is easy to see that we get

$$P = P_0 \to_{\tilde{w}_1} P_1 \to_{\tilde{w}_2} \cdots \to_{\tilde{w}_n} P_n = \underline{r}$$

for unique $\tilde{w}_1, \ldots, \tilde{w}_n$. Note that $w_i = \prod_{j=1}^{i} \tilde{w}_j$.

We first assume that $w \neq 0$ (so $\tilde{w}_i \neq 0$ for all i). We claim that for each $0 \leq i \leq n$, we can type

$$\vdash P_i : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w_i} \cdot [c,d]\end{smallmatrix}\right\} \tag{7}$$

Equation (7) follows by simple induction: the base case $i = 0$ holds by the assumption $\vdash P_0 : \left\{\begin{smallmatrix}[a,b]\\ [c,d]\end{smallmatrix}\right\}$ and because $w_0 = 1$. For the inductive case, we assume that $\vdash P_i : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w_i} \cdot [c,d]\end{smallmatrix}\right\}$. We apply Lemma D.2 to $P_i \to_{\tilde{w}_i} P_{i+1}$ and finish the induction step using $w_{i+1} = w_i \tilde{w}_{i+1}$:

$$\vdash P_{i+1} : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{\tilde{w}_{i+1}} \cdot \frac{1}{w_i} \cdot [c,d]\end{smallmatrix}\right\} = \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w_{i+1}} \cdot [c,d]\end{smallmatrix}\right\}$$

Equation (7) thus implies $\vdash \underline{r} : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w} \cdot [c,d]\end{smallmatrix}\right\}$. W.l.o.g., we can assume that this type judgment has the form

$$\dfrac{\dfrac{}{\vdash \underline{r} : \left\{\begin{smallmatrix}[r,r]\\ \mathbf{1}\end{smallmatrix}\right\}}\,(\textsc{Const})}{\vdash \underline{r} : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w} \cdot [c,d]\end{smallmatrix}\right\}}\,(\textsc{Sub})$$

This implies $r \in [a,b]$ and $w \in [c,d]$ (because $1 \in \frac{1}{w} \cdot [c,d]$), as required.

Now consider the case $w = 0$, so at least one $\tilde{w}_i = 0$. The claim that $r \in [a,b]$ follows easily as above (now using Item 1 of Lemma D.3 to handle the weight 0 reduction steps). Let i^* be the smallest index such that $\tilde{w}_{i^*} = 0$. Using the same argument as above on the (possibly empty) prefix up to index $i^* - 1$ (where all $\tilde{w}_j$ are non-zero) we find

$$\vdash P_{i^*-1} : \left\{\begin{smallmatrix}[a,b]\\ \frac{1}{w_{i^*-1}} \cdot [c,d]\end{smallmatrix}\right\}$$

Note that this is well defined because $\tilde{w}_j > 0$ for $j < i^*$. Item 2 of Lemma D.3 shows $0 \in \frac{1}{w_{i^*-1}} \cdot [c,d]$, which already implies $w = 0 \in [c,d]$, as required. □

D.2 Weak Completeness

Proposition 5.2. *Let $\vdash P : \alpha$ be a simply-typed program. There exists a weighted interval type $\mathcal{A}$ such that $\vdash P : \mathcal{A}$.*

Proof. For every *simple* type α, we define a weighted type $\mathcal{A}_\alpha$ and weightless type σ_α by mutual recursion as follows.

$$\sigma_{\mathbf{R}} := [-\infty, \infty]$$
$$\sigma_{\alpha \to \beta} := \sigma_\alpha \to \mathcal{A}_\beta$$
$$\mathcal{A}_\alpha := \left\{\begin{smallmatrix}\sigma_\alpha\\ [0, \infty]\end{smallmatrix}\right\}$$

That is, we insert $[-\infty, \infty]$ for values and $[0, \infty]$ for weights in all locations. We claim that if $\vdash P : \alpha$ in the simple type system, then $\vdash P : \mathcal{A}_\alpha$ in the weight-aware interval type system.

For the proof, we strengthen the induction hypothesis by claiming: if $\Gamma \vdash P : \alpha$ in the simple type system then $\Gamma^{\uparrow\sigma} \vdash P : \mathcal{A}_\alpha$ where

$$\Gamma^{\uparrow\sigma} := \{x : \sigma_\alpha \mid x : \alpha \in \Gamma\}.$$

The claim can be proved by simple induction on the derivation of $\Gamma \vdash P : \alpha$ using the respective typing rule for the interval type system possibly followed by (Sub) for typing rules that yield proper subtypes of $\mathcal{A}_\alpha$.

Note that this typing derivation does not contain any useful information to improve the precision of GuBPI. □

D.3 Constraint-based Type Inference

In this section we formalize the constraint-based type inference algorithm and sketch our constraints-solving method based on worklist and widening. The overarching idea is to substitute intervals with variables ν_i, called interval variables, and to encode typability as a constraint system. As we work in the restricted interval domain (as opposed to e.g. full first-order refinements), the resulting constraints can be solved very efficiently, which is crucial to the practicality of our tool.

Symbolic types. Symbolic types are defined by the following grammar:

$$\kappa := \nu_i \mid \kappa \to \mathscr{A} \qquad \mathscr{A} := \begin{Bmatrix} \kappa \\ \nu_j \end{Bmatrix}$$

where ν_i, ν_j are interval variables. Symbolic types are identical to interval types but use interval variables instead of intervals as first-order types and in the weight bound.

Constraints. Constraints on interval variables come in three forms:

$$c := \ \nu_n \equiv [a,b] \ \Big| \ \kappa_1 \sqsubseteq \kappa_2 \ \Big| \ \nu_n \equiv \underline{f}(\nu_{n_1}, \dots, \nu_{n_{|f|}})$$

where f is a primitive function and $[a,b]$ an interval. That is, a constraint can either equate an interval variable to a particular interval, require a symbolic type κ_1 to be a subtype of a type κ_2, or equate an interval variable to the result of a function applied to interval variables. Note that due to the compositional nature of our subtype relation $\sqsubseteq$ (which extends to symbolic types) we can restrict ourself to constraints of the form $\nu_1 \sqsubseteq \nu_2$ because each constraint of the form $\kappa_1 \sqsubseteq \kappa_2$ or $\mathscr{A}_1 \sqsubseteq \mathscr{A}_2$ (with identical base types) can be reduced to an equivalent set of constraints on interval variables by the definition of the subtype relation.

Symbolic type system. In the presentation of our symbolic type inference system, we aim to stay as close as possible to the implementation. Thus we describe it as an impure type system, meaning that our typing rules have side effects. In our case, typing rules can generate fresh interval variables.

For a simple type α, we write $\mathit{fresh}(\alpha)$ for the symbolic weightless type obtained by replacing every base type $\mathbf{R}$ with a fresh interval variable ν_n (and adding weights given by fresh interval variables where needed). We write $\mathit{fresh}()$ for $\mathit{fresh}(\mathbf{R})$. For a symbolic type κ we write $\mathit{base}(\kappa)$ for the underlying simple type (defined in the obvious way).

Our constraint generation system is given in Fig. 10. Judgments have the form $\Gamma \vdash M : \mathscr{A}, C$ where Γ maps variables to weightless symbolic types, $\mathscr{A}$ is a weighted symbolic type and C a list of constraints on the interval variables. The rules follow the structure of the system in Fig. 4 but replace all operations on intervals with interval variables and constraints. The term structure directly determines the symbolic typing derivation; there are no choices to be made, contrary to Fig. 4, which requires "cleverness", for example to find a suitable interval for an argument in the fixpoint rule. Note that in our system, we assume that the *simple* types of arguments of abstractions and fixpoints are given. These types can be determined by a simple prior run of any standard type inference algorithm.

From symbolic to concrete types. An assignment A is a mapping from interval variables to concrete intervals. Given a symbolic type κ, we define the interval type κ^A by replacing every interval variable in κ with the concrete interval assigned to it in A. For a weighted symbolic type $\mathscr{A}$, we define $\mathscr{A}^A$ in the same way. Given a set of constraints C, we say that A satisfies C, written $A \models C$ if all constraints in C are satisfied (defined in the obvious way).

Theorem D.4. *If* $\vdash M : \mathscr{A}, C$ *and* $A \models C$ *then* $\vdash M : \mathscr{A}^A$.

Proof. This can be shown by induction on the structure of the term, which also determines the symbolic typing derivation. From this, we obtain a valid interval typing derivation by replacing interval variables in the derivation with the concrete intervals assigned to them in A, and by applying the (Sub) rule in places where subtyping constraints are introduced. □

This theorem states that solutions to our constraints directly give us valid judgments in our interval type system, which allows us to invoke Theorem 5.1.

Solving Constraints. To solve the resulting constraints, we employ known techniques from abstract interpretation [14]. Again, note that due to the simplicity of our constraints, our approach avoids expensive calls to a theorem prover. When solving a set of constraints C, we are interested in the smallest solution, i.e. an assignment A with $A \models C$ where the intervals in $\mathcal{A}$ are smallest possible w.r.t. $\sqsubseteq$.

Naïve algorithm. A naïve attempt to find a satisfying assignment for a set of constraints would be to iterate over the constraints and to extend the current assignment (initially chosen to map all elements to the bottom element $\bot$ in the interval domain, i.e. an empty interval) whenever needed. For example, if $\nu_i \sqsubseteq \nu_j$ is not satisfied by assignment A, we can update A by mapping ν_j to $A(\nu_j) \sqcup A(\nu_i)$. As is well known from abstract interpretation, this naïve

$$\frac{x:\kappa\in\Gamma \qquad \nu=\mathit{fresh}()}{\Gamma\vdash x:\begin{Bmatrix}\kappa\\ \nu\end{Bmatrix},\{\nu\equiv\mathbf{1}\}} \qquad \frac{\kappa=\mathit{fresh}(\alpha) \qquad \nu=\mathit{fresh}() \qquad \Gamma;x:\kappa\vdash M:\mathscr{A},C}{\Gamma\vdash\lambda x^{\alpha}.M:\begin{Bmatrix}\kappa\to\mathscr{A}\\ \nu\end{Bmatrix},C\cup\{\nu\equiv\mathbf{1}\}}$$

$$\frac{\Gamma\vdash M:\begin{Bmatrix}\nu_1\\ \nu_2\end{Bmatrix},C \qquad \nu=\mathit{fresh}()}{\Gamma\vdash \mathsf{score}(M):\begin{Bmatrix}\nu_1\\ \nu\end{Bmatrix},C\cup\{\nu\equiv\nu_1\times\nu_2\}}$$

$$\frac{\nu_1=\mathit{fresh}() \qquad \nu_2=\mathit{fresh}()}{\Gamma\vdash \underline{r}:\begin{Bmatrix}\nu_1\\ \nu_2\end{Bmatrix},\{\nu_1\equiv[r,r],\nu_2\equiv\mathbf{1}\}}$$

$$\frac{\kappa=\mathit{fresh}(\alpha) \qquad \kappa_1=\mathit{fresh}(\beta) \qquad \nu,\nu_1=\mathit{fresh}() \qquad \Gamma;\varphi:\kappa\to\begin{Bmatrix}\kappa_1\\ \nu_1\end{Bmatrix};x:\kappa\vdash M:\begin{Bmatrix}\kappa_2\\ \nu_2\end{Bmatrix},C}{\Gamma\vdash \mu_x^{\varphi:\alpha\to\beta}.M:\begin{Bmatrix}\kappa\to\begin{Bmatrix}\kappa_2\\ \nu_2\end{Bmatrix}\\ \nu\end{Bmatrix},C\cup\{\nu\equiv\mathbf{1},\kappa_2\sqsubseteq\kappa_1,\nu_2\sqsubseteq\nu_1\}}$$

$$\frac{\nu,\nu'=\mathit{fresh}()}{\Gamma\vdash \mathsf{sample}:\begin{Bmatrix}\nu\\ \nu'\end{Bmatrix},\{\nu\equiv[0,1],\nu'\equiv\mathbf{1}\}}$$

$$\frac{\Gamma\vdash M:\begin{Bmatrix}\kappa_1\to\begin{Bmatrix}\kappa_2\\ \nu_2\end{Bmatrix}\\ \nu_1\end{Bmatrix},C_1 \qquad \Gamma\vdash N:\begin{Bmatrix}\kappa_3\\ \nu_3\end{Bmatrix},C_2 \qquad \nu=\mathit{fresh}()}{\Gamma\vdash MN:\begin{Bmatrix}\kappa_2\\ \nu\end{Bmatrix},C_1\cup C_2\cup\{\kappa_3\sqsubseteq\kappa_1,\nu\equiv\nu_1\times\nu_2\times\nu_3\}}$$

$$\frac{\Gamma\vdash M:\begin{Bmatrix}\nu_1\\ \nu_2\end{Bmatrix},C_M \qquad \Gamma\vdash N:\begin{Bmatrix}\kappa_1\\ \nu_3\end{Bmatrix},C_N \qquad \Gamma\vdash P:\begin{Bmatrix}\kappa_2\\ \nu_4\end{Bmatrix},C_P \qquad \kappa=\mathit{fresh}(\mathit{base}(\kappa_1)) \qquad \nu,\nu'=\mathit{fresh}()}{\Gamma\vdash \mathsf{if}(M,N,P):\begin{Bmatrix}\kappa\\ \nu\end{Bmatrix},C_M\cup C_N\cup C_P\cup\{\kappa_1\sqsubseteq\kappa,\kappa_2\sqsubseteq\kappa,\nu\equiv\nu_2\times\nu',\nu_3\sqsubseteq\nu',\nu_4\sqsubseteq\nu'\}}$$

$$\frac{\Gamma\vdash M_1:\begin{Bmatrix}\nu_1\\ \nu_1'\end{Bmatrix},C_1 \qquad \cdots \qquad \Gamma\vdash M_{|f|}:\begin{Bmatrix}\nu_{|f|}\\ \nu_{|f|}'\end{Bmatrix},C_{|f|} \qquad \nu,\nu'=\mathit{fresh}()}{\Gamma\vdash f(M_1,\ldots,M_{|f|}):\begin{Bmatrix}\nu\\ \nu'\end{Bmatrix},\bigcup_{i=1}^{|f|}C_i\cup\{\nu\equiv\underline{f}(\nu_1,\ldots,\nu_{|f|}),\nu'\equiv\prod_{i=1}^{|f|}\nu_i'\}}$$

Figure 10. Symbolic weight-aware type system.

approach may not terminate because the interval domain is not chain-complete. For instance, consider the constraints $C=\{\nu_1=[0,0],\nu_2=[1,1],\nu_1\sqsubseteq\nu_3,\nu_3\equiv\nu_3+\nu_2\}$. The minimal solution is $\{\nu_1\mapsto[0,0],\nu_2\mapsto[1,1],\nu_3\mapsto[0,\infty]\}$, but the algorithm never terminates and instead assigns the ascending chain $[0,0],[0,1],[0,2],\ldots$ to ν_3.

Widening. To remedy the above problem, we use ***widening***, a standard approach to ensure termination of abstract interpretation on domains with infinite chains [14]. Let ∇ be a *widening operator* for intervals. This means that $I_1\sqcup I_2\sqsubseteq I_1\nabla I_2$ for all intervals I_1,I_2 and for every chain $I_0\sqsubseteq I_1\sqsubseteq I_2\sqsubseteq\cdots$, the chain $(I_i^{\nabla})_{i\in\mathbb{N}}$ defined by $I_0^{\nabla}:=I_0$ and $I_i^{\nabla}:=I_{i-1}^{\nabla}\nabla I_i$ for $i\geq 1$ stabilises eventually. A trivial widening operator is given by only allowing intervals to extend to infinity, defined as follows:

$$\begin{aligned}
\bot\nabla I &:= I\nabla\bot := I\\
[a,b]\nabla[c,d] &:= [a,b] && \text{if } a\leq c\wedge d\leq b\\
[a,b]\nabla[c,d] &:= [a,\infty] && \text{if } a\leq c\wedge d>b\\
[a,b]\nabla[c,d] &:= [-\infty,b] && \text{if } d\leq b\wedge c<a\\
[a,b]\nabla[c,d] &:= [-\infty,\infty] && \text{if } c<a\wedge d>b
\end{aligned}$$

As soon as the upper bound increases or lower bound decreases, the bound is directly set to $=\infty$ or $-\infty$ respectively. By using the widening operator in each update step of our naïve algorithm, we break infinite increasing chains and the resulting algorithm is guaranteed to converge to a satisfying assignment (if one exists).

GuBPI solves constraints by using a standard worklist algorithm [15, 23], combined with the previous widening operator.

E Supplementary Material for Section 6

E.1 Extensions to Linear Splitting

In this section we give additional information about how our linear optimisation of the interval-trace semantics can be extended to non-uniform samples and non-linear scoring values.

Beyond uniform samples. To allow for non-uniform samples, we can combine the standard interval trace semantics with the linear optimisation. That is, in addition to bounding linear score functions, we also split and bound each non-uniform sample (as in the standard interval trace semantics). Suppose that α_i is sampled from some continuous distribution D. We then split the real line into chunks (the size and number of which is selected by a heuristic depending on D). For each such chunk $[a,b]$, we compute the volume and multiply by the lower and upper bounds of the pdf of D on $[a,b]$. In this way, we can even approximate integrals where

not all variables are sampled from a uniform distribution (without needing to resort to inverse cumulative distribution functions). Our experiments show, that as long as the guards are still linear, this approach is advantageous compared to the naive interval-based semantics.

Beyond linear functions. While guards on conditionals are often linear, this is rarely the case for score expressions (as one usually observes values from some non-uniform distribution with a non-linear pdf). Consider the pedestrian example again. While all guards are linear, the score expression has the form $\mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}(\mathcal{V})$. We can handle such *non-linear* score values by applying linear optimisation to the linear subexpressions and interval arithmetic for the nonlinear parts. Formally, we assume that each $\mathcal{W}_i \in \Xi$ (each symbolic value we score with) has the form $\mathcal{W}_i = f_i(\mathcal{Z}_i^1, \dots, \mathcal{Z}_i^{m_i})$ where $\mathcal{Z}_i^j$ for $1 \leq j \leq m_i$ denote *linear* functions of the sample variables and $f : \mathbb{R}^{m_i} \to \mathbb{R}$ is a possibly non-linear function. Every score expression can be written in this way. For instance, in the pedestrian example, we have $f = \mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}$.

Let $\Xi' = \{\mathcal{Z}_i^j \mid i \in \{1, \dots, n\}, j \in \{1, \dots, m_i\}\}$ be the set of all such linear functions. We bound each linear function in Ξ' using linear optimisation as before. We obtain a box b (which now has dimension $|\Xi'|$ instead of $|\Xi|$) and define the weight $\mathit{weight}(b)$ by applying the interval liftings $f_i^{\mathbb{I}}$ of the non-linear functions f_i to the bounds for each argument. Formally, $\mathit{weight}(b) = \prod_{i=1}^{k} f_i^{\mathbb{I}}(b_i^1, \dots, b_i^{m_i})$ where b_i^j is the interval bound on $\mathcal{Z}_i^j$. Note that this strictly generalizes the approach outlined before since we can choose f_i as the identity if $\mathcal{W}_i$ is already linear. The definition of $\mathit{approx}(b)$ with the new weight definition still satisfies Proposition 6.4. This way, we can even approximate integrals over non-linear functions by means of simple volume computations. As our experiments (e.g. on the pedestrian example) show, this approximation is precise enough to obtain useful bounds on the posterior. It is important to note that while we can deal with non-linear score values, we cannot handle non-linear guards and instead use the standard semantics for such cases.

F Supplementary Material for Section 7

Our experiments were performed on a server running Ubuntu 18.04 with an 8core Intel(R) Xeon(R) CPU E3-1271 v3 @ 3.60GHz CPU with 32Gbp of RAM. The current version of GuBPI is not parallelised and makes no use of the additional cores. The running times on a Macbook Pro with Apple M1 were comparable, and sometimes even faster.

F.1 Pyro's HMC samples for the pedestrian example

The HMC samples plotted in Figs. 1 and 7 were generated with the probabilistic programming system Pyro [5]. Since the original pedestrian program has infinite expected running time, we introduced a stopping condition in the random walk: if the distance traveled exceeded 10, the loop was exited. (This has a negligible effect on the posterior distribution because the weight of such a trace is at most $\mathrm{pdf}_{\mathrm{Normal}(1.1,0.1)}(10) < 10^{-1700}$.)

We used Pyro's HMC sampler to compute 10 Markov chains with 1000 samples each for this program. We set the hyperparamaters to 0.1 for the step size and 50 for the number of steps. We also tried the NUTS sampler, which aims to automatically estimate good values for the hyperparameters, but it performed worse than the manually chosen values. The running time for the chains varied significantly: some took around one minute, others almost an hour. This depended on whether the Markov chain got "stuck" in a long trace. (The length of the traces varied between 2 and about 200.)

We discarded chains with very low acceptance rates (under 1%), aggregated the remaining chains, which had acceptance rates of over 50%, and used their histogram in Figs. 1 and 7.

F.2 Details on Probability Estimation

In Table 1 (results of our tool for the probability estimation benchmark), we omitted the query for space reasons. Complete information including the query can be found in Table 4.

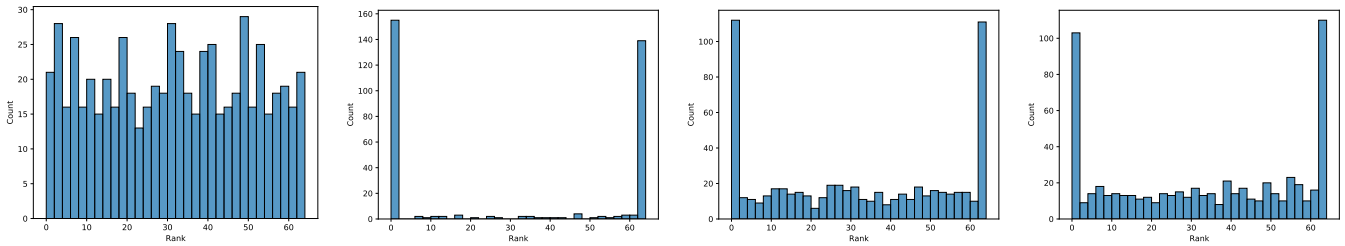
F.3 Simulation-based Calibration

We implemented SBC for both likelihood-weighted importance sampling and Pyro's HMC. As hyperparameters for SBC, we picked $L = 63$ samples per simulation (following the suggestion in [60] to take one less than a power of two) and N simulations with $N = 10L$ (also following the paper's suggestion). Note that the number of samples is much less than the 10000 samples used for Figs. 1 and 7 (10 chains with 1000 samples each). But setting $L = 1000$ would be at least 100 times slower because N has to increase proportionally to L. Also note that for Pyro, we ran HMC with L warmup steps before generating L samples. Both importance samples and HMC samples exhibited significant autocorrelation. As suggested in [60], we applied thinning to reduce its effect, choosing a thinning factor of around $\frac{L}{L_{\mathit{eff}}}$ where L_{eff} is the effective sample size.

Pedestrian example. For importance sampling, the rank histogram looks fairly uniform (Fig. 11a), which means that SBC does not detect an issue with the inference and thus increases the confidence in the validity of the importance samples. For Pyro's HMC, simulation-based calibration is very slow: the rank histogram (Fig. 11b) took 32 hours (!) to produce. (Note that only 332 simulations could be used, the rest were discarded because the acceptance rate was too low.) The spikes at the boundary indicate that the samples have high autocorrelation and in fact, the effective sample size L_{eff} was at most $\frac{L}{10}$, often much lower (depending on the chain). The suggestion in [60] is thus to apply thinning, with a factor of $\frac{L}{L_{\mathit{eff}}}$, which is at least 10 in our case. This would increase

Table 4. Evaluation on selected benchmarks from [56]. We give the times (in seconds) and bounds computed by [56] and GuBPI. The table agrees with Table 1 but spells out the full problem name and the exact query.

			Tool from [56]		GuBPI	
Program	**Q**	**Query**	t	**Result**	t	**Result**
tug-of-war	Q1	$total\_a\_b < total\_t\_s$	1.29	[0.6126, 0.6227]	0.72	[0.6134, 0.6135]
tug-of-war	Q2	$total\_a\_s < total\_b\_t$	1.09	[0.5973, 0.6266]	0.79	[0.6134, 0.6135]
beauquier-etal-3	Q1	$count < 1$	1.15	[0.5000, 0.5261]	22.5	[0.4999, 0.5001]
example-book-simple	Q1	$count \geq 2$	8.48	[0.6633, 0.7234]	6.52	[0.7417, 0.7418]
example-book-simple	Q2*	$count \geq 4$	10.3	[0.3365, 0.3848]	8.01	[0.4137, 0.4138]
example-cart	Q1	$count \geq 1$	2.41	[0.8980, 1.1573]	67.3	[0.9999, 1.0001]
example-cart	Q2	$count \geq 2$	2.40	[0.8897, 1.1573]	68.5	[0.9999, 1.0001]
example-cart	Q3	$count \geq 4$	0.15	[0.0000, 0.1150]	67.4	[0.0000, 0.0001]
example-ckd-epi-simple	Q1*	$f_1 \leq 4.4 \wedge f \geq 4.6$	0.15	[0.5515, 0.5632]	0.86	[0.0003, 0.0004]
example-ckd-epi-simple	Q2*	$f_1 \geq 4.6 \wedge f \leq 4.4$	0.08	[0.3019, 0.3149]	0.84	[0.0003, 0.0004]
example-fig6	Q1	$c \leq 1$	1.31	[0.1619, 0.7956]	21.2	[0.1899, 0.1903]
example-fig6	Q2	$c \leq 2$	1.80	[0.2916, 1.0571]	21.4	[0.3705, 0.3720]
example-fig6	Q3	$c \leq 5$	1.51	[0.4314, 2.0155]	24.7	[0.7438, 0.7668]
example-fig6	Q4	$c \leq 8$	3.96	[0.4400, 3.0956]	27.4	[0.8682, 0.9666]
example-fig7	Q1	$x \leq 1000$	0.04	[0.9921, 1.0000]	0.18	[0.9980, 0.9981]
example4	Q1	$x + y > 10$	0.02	[0.1910, 0.1966]	0.31	[0.1918, 0.1919]
example5	Q1	$x + y > z + 10$	0.06	[0.4478, 0.4708]	0.27	[0.4540, 0.4541]
herman-3	Q1	$count < 1$	0.47	[0.3750, 0.4091]	124	[0.3749, 0.3751]



(a) Importance sampling for the pedestrian example, thinning factor 100.

(b) HMC for the pedestrian example, no thinning. (Only 332 simulations were used because the rest had too low acceptance rates.)

(c) NUTS for the binary Gaussian mixture model, thinning factor 10.

(d) NUTS for the *two-dimensional* binary Gaussian mixture model, thinning factor 10.

Figure 11. Simulation-based calibration: rank histogram plots (630 simulations with 63 samples each).

the running time of SBC by the same factor, to at least 300 hours, but probably 600 or more. We did not consider it a good use of resources to carry out this experiment.

Binary Gaussian Mixture Model. We also considered the binary GMM (Fig. 5c) and a two-dimensional version of the same model. The spikes at the boundary could again be a sign of high autocorrelation, but in this case, we already applied thinning with a factor of 10 (again based on the effective sample size). Instead, as discussed in [60], this symmetric U-shape indicates that the computed data-averaged posterior is underdispersed relative to the prior distribution. This interpretation is consistent with our knowledge about the model: HMC only finds one mode in the posterior distribution and misses the others. Hence SBC successfully detects the issue, and in the case of the higher-dimensional model, it does so in less time than GuBPI (cf. Table 3). For the other models, including the pedestrian example, GuBPI is faster.

Contents