Static Analysis of Posterior Inference in Bayesian Probabilistic Programming

Peixin Wang[1], Hongfei Fu[2], Tengshun Yang[3,4], Guanyan Li[1], and Luke Ong[1,5]

[1] University of Oxford
[2] Shanghai Jiao Tong University
[3] SKLCS, Institute of Software, Chinese Academy of Sciences
[4] University of Chinese Academy of Sciences
[5] Nanyang Technological University

Abstract. In Bayesian probabilistic programming, a central problem is to estimate the normalised posterior distribution (NPD) of a probabilistic program with score statements. Prominent approximate approaches to address this problem cannot generate guaranteed outcomes within a finite time limit, and previous formal approaches w.r.t. exact inference for NPD are restricted to programs with bounded loops/recursion. A recent work (Beutner *et al.*, PLDI 2022) proposed an automated approach that derives guaranteed bounds for NPD over programs with unbounded recursion. However, as this approach requires recursion unrolling, it suffers from the path explosion problem. Moreover, existing approaches do not consider *score-recursive* probabilistic programs that allow score statements inside loops, which is non-trivial and requires careful treatment to ensure the integrability of the normalising constant.

In this work, we propose a novel automated approach to derive bounds for NPD via polynomial templates, fixed-point theorems and Optional Stopping Theorem (OST). Our approach can handle probabilistic programs with unbounded while-loops and continuous distributions with infinite supports. Our novelties are three-fold: First, the use of polynomial templates circumvents the path explosion problem from recursion unrolling; Second, we derive a novel variant of OST that addresses the integrability issues in score-recursive programs; Third, to increase the accuracy of the derived bounds, our approach adopts a novel technique of truncation onto a bounded range of program values. Experiments over a wide range of benchmarks demonstrate that our approach is time-efficient and can derive bounds on NPD that can be tighter than (or comparable with) the recursion-unrolling approach (Beutner *et al.*, PLDI 2022).

1 Introduction

Bayesian statistical probabilistic programming aims at first modelling probabilistic models as probabilistic programs and then analyzing the models through their probabilistic program representations. Compared with traditional approaches [29,30,7,9] that specify an ad-hoc programming language, probabilistic programming languages (PPLs) [31] provide a universal framework to perform Bayesian

inference. Unlike standard programming languages, PPLs have two specific constructs: `sample` and `score` [6].[6] The former construct allows drawing samples from a (prior) distribution, while the latter records the likelihood of observed data in the form of "`score`(weight)".[7] Thanks to its universality, probabilistic programming has nowadays become an active research subject in both machine learning and programming language communities, and there have been an abundance of PPLs, such as Pyro[4], WebPPL[20], Anglican[46], Church[19], etc.

In this work, we consider the central problem of analyzing normalised posterior distributions (NPD) in Bayesian inference over probabilistic programs. The general setting of this problem is that given a prior distribution $p(z)$ over the latent variables $z \in \mathbb{R}^n$ of interest and the distribution $p(x, z)$ of the probabilistic model represented by a probabilistic program, the task is to estimate the NPD by observing the evidence $x \in \mathbb{R}^m$ with the likelihood $p(x|z)$. Note that the problem can be generally solved by the Bayes' rule $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$, but the main difficulty to apply Bayes' rule is that the normalising constant $p(x) = \int p(x|z)p(z)\mathrm{d}z$ is usually intractable to compute.

There are two existing classes of approaches to address the NPD problem. The first is the approximate approaches that estimate the NPD by random simulations, while the second is the formal approaches that aim at deriving guaranteed upper and lower bounds for NPD. In approximate approaches, two dominant methods are Markov chain Monte Carlo (MCMC) [16] and variational inference (VI) [5]. Although such approaches can produce approximate results efficiently, they cannot provide formal guarantees within a finite time limit. In formal approaches, there is a large amount of previous work such as (λ)PSI [17,18], AQUA [23], Hakaru [32] and SPPL [40], aiming to make exact inference for NPD. However, these methods are restricted to specific kinds of programs, e.g., programs with closed-form solutions to NPD or without continuous distributions, and none of them can handle probabilistic programs with unbounded while-loops/recursion. Recently, Beutner et al. [3] proposed an approach that infers guaranteed upper and lower bounds for NPD and allows unbounded recursion. The main techniques in this approach are (i) the unrolling of every recursion in a probabilistic program to eliminate the appearance of recursion, (ii) the widening operator of abstract interpretation to eliminate the non-termination case in the unrolling and (iii) the interval semantics to handle continuous distributions.

Challenges and gaps. In this work, we focus on developing formal approaches to derive upper and lower bounds on NPD over probabilistic programs. We allow programs to have unbounded while-loops and infinite-support distributions, which makes it a challenging task. The most relevant work is Beutner et al. [3], but it has the following drawbacks. The first is that this approach is based on recursion unrolling, and hence may cause path explosion. The second is that this approach cannot handle the situation where `score` statements with weight greater than 1 appear inside a loop. In the sequel, we call such programs *score-recursive* programs, and we show that `score` inside a loop may cause unbounded weights and integrability issues and thus requires careful treatment. Staton et

[6] Sometimes `observe` is used instead of `score` [21], which has the same implicit effect.
[7] The argument "weight" corresponds to the likelihood each time the data is observed.

al. [44] also noted that for a non-recursive λ-calculus with `score`, unbounded weights may introduce the possibility of "infinite model evidence errors". To circumvent the second drawback, previous results (e.g., Borgström et al. [6]) allow only 1-bounded weights, and no existing approaches can handle score-recursive programs whose weights of `score` can be greater than 1.

Contributions. Our contributions are three-fold: First, to circumvent the path explosion from loop unrolling (that corresponds to recursion unrolling in functional programs), we propose a novel approach that adopts the template paradigm [11,10,51], fixed-point theorems [45] and Optional Stopping Theorem (OST) [52] to synthesize polynomial bounds for NPD. Second, to address the integrability issue from score-recursive programs, we present a novel variant of the classical OST to validate the finiteness of the normalising constant $p(x)$ in NPD (i.e., $0 < p(x) < \infty$). Third, to increase the accuracy of the derived bounds, we further propose a novel truncation operation that truncates the probabilistic program of concern onto a bounded range of program values, which allows our algorithm to explore high-degree polynomials to improve the accuracy. Experimental results show that our approach can handle a wide range of benchmarks including score-recursive examples from phylogenetics [37], and improve the runtime of the previous approach [3] by up to 6 times while obtaining tighter or comparable bounds.

Limitations. A limitation is that our approach has the combinatorial explosion that when the degree of polynomial templates increases, the time complexity of the template solving increases exponentially. However, from our experimental results, a moderate choice of the degree (e.g., no more than 10) suffices. Another limitation is that for sampling statements our approach currently can only handle continuous distributions with polynomial density functions, but with proper approximation via piecewise polynomials, our approach can directly handle any continuous distributions with continuous density functions.

2 Preliminaries

We first review some basic concepts from probability theory (see standard textbooks such as [34,52] for a detailed treatment), and then present the Bayesian probabilistic programming language and the normalised posterior distribution (NPD) problem. We denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the sets of all natural numbers (including zero), integers, and real numbers, respectively.

2.1 Basics of Probability Theory

A *measurable space* is a pair (U, Σ_U), where U is a nonempty set and Σ_U is a σ-algebra on U, i.e., a family of subsets of U such that $\Sigma_U \subseteq \mathcal{P}(U)$ contains $\emptyset$ and is closed under complementation and countable union. Elements of Σ_U are called *measurable* sets. A function f from a measurable space (U_1, Σ_{U_1}) to another measurable space (U_2, Σ_{U_2}) is *measurable* if $f^{-1}(A) \in \Sigma_{U_1}$ for all $A \in \Sigma_{U_2}$.

A *measure* μ on a measurable space (U, Σ_U) is a mapping from Σ_U to $[0, \infty]$ such that (i) $\mu(\emptyset) = 0$ and (ii) μ is countably additive: for every pairwise-disjoint set sequence $\{A_n\}_{n\in\mathbb{N}}$ in Σ_U, it holds that $\mu(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$. We call

the triple (U, Σ_U, μ) a *measure space*. If $\mu(U) = 1$, we call μ a *probability measure*, and (U, Σ_U, μ) a *probability space*. The Lebesgue measure λ is the unique measure on $(\mathbb{R}, \Sigma_{\mathbb{R}})$ satisfying $\lambda([a, b)) = b - a$ for all valid intervals $[a, b)$ in $\Sigma_{\mathbb{R}}$. For each $n \in \mathbb{N}$, we have a measurable space $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$ and a unique product measure λ_n on $\mathbb{R}^n$ satisfying $\lambda_n(\prod_{i=1}^n A_i) = \prod_{i=1}^n \lambda(A_i)$ for all $A_i \in \Sigma_{\mathbb{R}}$.

The *Lebesgue* integral operator $\int$ is a partial operator that maps a measure μ on (U, Σ_U) and a real-valued function f on the same space (U, Σ_U) to a real number or infinity, which is denoted by $\int f \mathrm{d}\mu$ or $\int f(x)\mu(\mathrm{d}x)$. The detailed definition of Lebesgue integral is somewhat technical, see [36,39] for more details. Given a measurable set $A \in \Sigma_U$, the integral of f over A is defined by $\int_A f(x)\mu(\mathrm{d}x) := \int f(x) \cdot [x \in A]\mu(\mathrm{d}x)$ where $[-]$ is the Iverson bracket such that $[\phi] = 1$ if ϕ is true, and 0 otherwise. If μ is a probability measure, then we call the integral as the *expectation* of f, denoted by $\mathbb{E}_{x\sim\mu;A}[f]$, or $\mathbb{E}[f]$ when the scope is clear from the context.

For a measure v on (U, Σ_U), a measurable function $f : U \to \mathbb{R}_{\geq 0}$ is the *density* of v with respect to μ if $v(A) = \int f(x) \cdot [x \in A]\mu(\mathrm{d}x)$ for all measurable $A \in \Sigma_U$, and μ is called the *reference measure* (most often μ is the Lebesgue measure). Common families of probability distributions on the reals, e.g., uniform, normal distributions, are measures on $(\mathbb{R}, \Sigma_{\mathbb{R}})$. Most often these are defined in terms of probability density functions with respect to the Lebesgue measure. That is, for each μ_D there is a measurable function $\mathrm{pdf}_D : \mathbb{R} \to \mathbb{R}_{\geq 0}$ that determines it: $\mu_D(A) := \int_A \mathrm{pdf}_D(\mathrm{d}\lambda)$. As we will see, density functions such as pdf_D play an important role in Bayesian inference.

Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a *random variable* is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R} \cup \{+\infty, -\infty\}$. The expectation of a random variable X, denoted by $\mathbb{E}(X)$, is the Lebesgue integral of X w.r.t. $\mathbb{P}$, i.e., $\int X \, \mathrm{d}\mathbb{P}$. A *filtration* of $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence $\{\mathcal{F}_n\}_{n=0}^{\infty}$ such that for every $n \geq 0$, the triple $(\Omega, \mathcal{F}_n, \mathbb{P})$ is a probability space and $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$. A *stopping time* w.r.t. $\{\mathcal{F}_n\}_{n=0}^{\infty}$ is a random variable $T : \Omega \to \mathbb{N} \cup \{0, \infty\}$ such that for every $n \geq 0$, the event $\{T \leq n\}$ is in $\mathcal{F}_n$.

A *discrete-time stochastic process* is a sequence $\Gamma = \{X_n\}_{n=0}^{\infty}$ of random variables in $(\Omega, \mathcal{F}, \mathbb{P})$. The process Γ is *adapted* to a filtration $\{\mathcal{F}_n\}_{n=0}^{\infty}$, if for all $n \geq 0$, X_n is a random variable in $(\Omega, \mathcal{F}_n, \mathbb{P})$. A discrete-time stochastic process $\Gamma = \{X_n\}_{n=0}^{\infty}$ adapted to a filtration $\{\mathcal{F}_n\}_{n=0}^{\infty}$ is a *martingale* (resp. *supermartingale*, *submartingale*) if for all $n \geq 0$, $\mathbb{E}(|X_n|) < \infty$ and it holds almost surely (i.e., with probability 1) that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] = X_n$ (resp. $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \leq X_n$, $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \geq X_n$). See [52] for details. Applying martingales to qualitative and quantitative analysis of probabilistic programs is a well-studied technique [7,9,13].

2.2 Bayesian Probabilistic Programming Language

The syntax of our probabilistic programming language (PPL) is given in Fig. 1, where the metavariables S, B and E stand for statements, boolean expressions and arithmetic expressions, respectively. Our PPL is imperative with the usual conditional and loop structures (i.e., **if** and **while**), as well as the following new structures: (a) sample constructs of the form "**sample** D" that sample a value from a prescribed distribution D over $\mathbb{R}$ and then assign this value

to a sampling variable r; (b) score statements of the form "$\mathbf{score}(EW)$" that weight the current execution with a value expressed by EW (note that $\mathit{pdf}(D, x)$ means the value of a probability density function w.r.t. D at x); (c) probabilistic branching statements of the form "$\mathbf{if}\ \mathbf{prob}(p) \ldots$" that lead to the then part with probability $p \in (0, 1]$ and to the else part with probability $1 - p$. We also have sequential compositions (i.e., ";") and support return statements (i.e., **return**) that returns the value of the program variable of interest. Note that $c, c_1, c_2 \in \mathbb{R}$ are constants, and our language supports any distributions with continuous density functions and infinite supports, including but not limited to uniform and normal distributions.

$$
\begin{aligned}
S &::= \mathbf{skip} \mid x := ES \mid \mathbf{score}(EW) \mid \mathbf{return}\ x \mid S_1; S_2 \\
&\quad \mid \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od} \mid \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi} \mid \mathbf{if}\ \mathbf{prob}(p)\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi} \\
B &::= \mathbf{true} \mid \mathbf{false} \mid \neg B \mid B_1\ \mathbf{and}\ B_2 \mid B_1\ \mathbf{or}\ B_2 \mid E_1 \leq E_2 \mid E_1 \geq E_2 \\
E &::= x \mid c \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \quad ES ::= E \mid \mathbf{sample}\ D \\
EW &::= E \mid \mathit{pdf}(D, x) \quad D ::= \mathbf{normal}(c_1, c_2) \mid \mathbf{uniform}(c_1, c_2) \mid \cdots
\end{aligned}
$$

Fig. 1. Syntax of Our Probabilistic Programming Language

Given a probabilistic program in our language, we distinguish two disjoint sets of variables in the program: (i) the set V_p of *program variables* whose values are determined by assignments in the program (i.e., the expressions at the LHS of ":="); (ii) the set V_r of *sampling variables* whose values are independently sampled from prescribed probability distributions each time they are accessed (i.e., each "**sample** D" is associated with a sampling variable r). We relegate detailed program examples to Section 3.

To relate variables with their values, we introduce the notion of valuations. Let V be a finite set of variables with an implicit linear order over its elements. A *valuation* on V is a function $\mathbf{v} : V \to \mathbb{R}$ that assigns a real value to each variable in V. We denote the set of all valuations on V by Val_V. For each $1 \leq i \leq |V|$, we denote the value of the i-th variable (in the implicit linear order) in $\mathbf{v}$ by $\mathbf{v}[i]$, so that we can view each valuation as a real vector on V. A *program* (resp. *sampling*) valuation is a valuation on V_p (resp. V_r), respectively. For the sake of convenience, we fix the notations in the following way, i.e., we always use $\mathbf{v} \in \mathit{Val}_{V_\mathrm{p}}$ to denote a program valuation, and $\mathbf{r} \in \mathit{Val}_{V_\mathrm{r}}$ to denote a sampling valuation; we also write $\mathbf{v}[\mathit{ret}]$ for the value of the return variable in $\mathbf{v}$.

Below we present the semantics for our language. Existing semantics in the literature are either measure-[44,27] or sampling-based [28,3]. To facilitate the development of our algorithm, we migrate the *transition-based* semantics [8,11] to our language and treat each probabilistic program as a *weighted probabilistic transition system* (WPTS). A WPTS extends a PTS in [8,11] with weights and an initial probability distribution. The transformation from a probabilistic program into its WPTS can be done in a straightforward way (see e.g. [12,8]).

Definition 1 (WPTS). *A* weighted probabilistic transition system *(WPTS) is a tuple*

$$\Pi = (V_\mathrm{p}, V_\mathrm{r}, L, \ell_\mathrm{init}, \ell_\mathrm{out}, \mu_\mathrm{init}, \mathcal{D}, \mathfrak{T}) \qquad (\dagger)$$

for which:

- V_{p} *and* V_{r} *are finite disjoint sets of* program *and resp.* sampling *variables.*
- L *is a finite set of* locations *with special locations* $\ell_{\mathrm{init}}, \ell_{\mathrm{out}} \in L$*. Informally,* ℓ_{init} *is the initial location and* ℓ_{out} *represents program termination.*
- μ_{init} *is the* initial probability distribution *over* $\mathbb{R}^{V_{\mathrm{p}}}$*, while* $\mathcal{D}$ *is a function that assigns a probability distribution* $\mathcal{D}(r)$ *to each* $r \in V_{\mathrm{r}}$*. We call each* $\mathbf{v} \in \mathit{supp}(\mu_{\mathrm{init}})$ *an* initial program valuation*, and abuse the notation so that* $\mathcal{D}$ *also denotes the independent joint distribution of all* $\mathcal{D}(r)$*'s (*$r \in V_{\mathrm{r}}$*).*
- $\mathfrak{T}$ *is a finite set of* transitions *where each transition* $\tau \in \mathfrak{T}$ *is a tuple* $\langle \ell, \phi, F_1, \ldots, F_k \rangle$ *such that (i)* $\ell \in L$ *is the* source location *of the transition, (ii)* ϕ *is the* guard condition *which is a predicate over variables* V_{p}*, and (iii) each* $F_j := \langle \ell'_j, p_j, \mathrm{upd}_j, \mathrm{wt}_j \rangle$ *is called a* weighted fork *for which (a)* $\ell'_j \in L$ *is the* destination location *of the fork, (b)* $p_j \in (0, 1]$ *is the probability of this fork, (c)* $\mathrm{upd}_j : \mathbb{R}^{|V_{\mathrm{p}}|} \times \mathbb{R}^{|V_{\mathrm{r}}|} \to \mathbb{R}^{|V_{\mathrm{p}}|}$ *is an* update function *that takes as inputs the current program and sampling valuations and returns an updated program valuation in the next step, and (d)* $\mathrm{wt}_j : \mathbb{R}^{|V_{\mathrm{p}}|} \times \mathbb{R}^{|V_{\mathrm{r}}|} \to [0, \infty)$ *is a* score function *that gives the likelihood weight of this fork depending on the current program and sampling valuations.*

In a WPTS, we use update and score functions to model the update on the program variables and the likelihood weight when running a basic block in a program, respectively. If there is no score statement in the block, then the score function is constantly 1. We always assume that a WPTS Π is *deterministic* and *total*, i.e., (i) there is no program valuation that simultaneously satisfies the guard conditions of two distinct transitions from the same source location, and (ii) the disjunction of the guard conditions of all the transitions from any source location is a tautology.

We say that a WPTS is *non-score-recursive* if for all transitions $\tau = \langle \ell, \phi, F_1, F_2, \ldots, F_k \rangle$ in the WPTS with each fork $F_j = \langle \ell'_j, p_j, \mathrm{upd}_j, \mathrm{wt}_j \rangle$ $(1 \leq j \leq k)$, we have that each score function wt_j is constantly 1 (i.e., the multiplicative weight does not change) if $\ell'_j \neq \ell_{\mathrm{out}}$. Otherwise, the WPTS is *score-recursive*. Informally, a non-score-recursive WPTS has non-trivial score functions only on the transitions to the termination of a program, while a score-recursive WPTS has `score` in the execution of the program (e.g., in a while loop).

Below we present the semantics of a WPTS. Fix a WPTS Π in the form of (†). Given a program valuation $\mathbf{v}$ and a predicate ϕ over variables V_{p}, we say that $\mathbf{v}$ *satisfies* ϕ (written as $\mathbf{v} \models \phi$) if ϕ holds when the variables in ϕ are substituted by their values in $\mathbf{v}$. Moreover, we have that a *state* is a pair $\Xi = (\ell, \mathbf{v})$ where $\ell \in L$ (resp. $\mathbf{v} \in \mathbb{R}^{|V_{\mathrm{p}}|}$) represents the current location (resp. program valuation), respectively, while a *weighted state* is a triple $\Theta = (\ell, \mathbf{v}, w)$ where $(\ell, \mathbf{v})$ is a state and $w \in [0, \infty)$ represents the multiplicative likelihood weight so far. We denote by Λ the set of all states, and by Δ the set of all weighted states.

The semantics of the WPTS Π is formalized by the infinite sequence $\Gamma = \{\widehat{\Theta}_n = (\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)\}_{n \geq 0}$ of *random weighted states* where each $(\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)$ is the random weighted state at the nth execution step of the WPTS such that $\widehat{\ell}_n$ (resp. $\widehat{\mathbf{v}}_n$, $\widehat{w}_n$) is the random variable for the location (resp. the random vector for the program valuation, the random variable for the multiplicative likelihood weight)

at the nth step, respectively. The sequence Γ starts with the initial random weighted state $\widehat{\Theta}_0 = (\widehat{\ell}_0, \widehat{\mathbf{v}}_0, \widehat{w}_0)$ such that $\widehat{\ell}_0$ is constantly ℓ_{init}, $\widehat{\mathbf{v}}_0 \in \mathsf{supp}\,(\mu_{\text{init}})$ is sampled from the initial distribution μ_{init} and the initial weight $\widehat{w}_0$ is constantly set to 1[8]. Then, at each nth step, given the current random weighted state $\widehat{\Theta}_n = (\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)$, the next random weighted state $\widehat{\Theta}_{n+1} = (\widehat{\ell}_{n+1}, \widehat{\mathbf{v}}_{n+1}, \widehat{w}_{n+1})$ is determined by: (a) If $\widehat{\ell}_n = \ell_{\text{out}}$, then $(\widehat{\ell}_{n+1}, \widehat{\mathbf{v}}_{n+1}, \widehat{w}_{n+1})$ takes the same weighted state as $(\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)$ (i.e., the next weighted state stays at the termination location ℓ_{out}); (b) Otherwise, $\widehat{\Theta}_{n+1}$ is determined by the following procedure:

- First, since the WPTS Π is deterministic and total, we take the unique transition $\tau = \langle \hat{\ell}_n, \phi, F_1, \ldots, F_k \rangle$ such that $\hat{\mathbf{v}}_n \models \phi$.
- Second, we choose a fork $F_j = \langle \ell_j, p_j, upd_j, wt_j \rangle$ with probability p_j.
- Third, we obtain a sampling valuation $\mathbf{r} \in \mathsf{supp}\,(\mathcal{D})$ by sampling each $r \in V_{\text{r}}$ independently w.r.t $\mathcal{D}(r)$.
- Finally, the value of the next random weighted state $(\widehat{\ell}_{n+1}, \widehat{\mathbf{v}}_{n+1}, \widehat{w}_{n+1})$ is determined as that of $(\ell'_j, upd_j(\hat{\mathbf{v}}_n, \mathbf{r}), \widehat{w}_n \cdot wt_j(\hat{\mathbf{v}}_n, \mathbf{r}))$.

Given the semantics, a *program run* of the WPTS Π is a concrete instance of Γ, i.e., an infinite sequence $\omega = \{\Theta_n\}_{n \geq 0}$ of weighted states where each $\Theta_n = (\ell_n, \mathbf{v}_n, w_n)$ is the concrete weighted state at the nth step in this program run with location ℓ_n, program valuation $\mathbf{v}_n$ and multiplicative likelihood weight w_n. A state $(\ell, \mathbf{v})$ is called *reachable* if there exists a program run $\omega = \{\Theta_n\}_{n \geq 0}$ such that $\Theta_n = (\ell, \mathbf{v}, w_n)$ for some n.

We introduce more technical concepts. The *termination time* random variable T of a WPTS Π is given by $T(\omega) := \min\{n \in \mathbb{N} \mid \ell_n = \ell_{\text{out}}\}$ for every program run $\omega = \{(\ell_n, \mathbf{v}_n, w_n)\}_{n \geq 0}$ where $\min \emptyset := \infty$. That is, $T(\omega)$ is the number of steps a program run ω takes to reach the termination location ℓ_{out}. A WPTS Π is *almost-surely terminating* (AST) if $\mathbb{P}(T < \infty) = 1$. A *return function* of a WPTS Π is the function $ret : \mathsf{supp}\,(\mu_{\text{init}}) \to \Sigma_{\mathbb{R}^{|V_{\text{p}}|}}$ such that for any initial program valuation $\mathbf{v}$, we have $ret_{\Pi}(\mathbf{v})$ is the set of all program valuations $\widehat{\mathbf{v}}_T(\omega)$ such that ω is a program run whose initial program valuation equals $\mathbf{v}$.

Below we define the normalised posterior distribution (NPD) problem.

Definition 2 (Normalised Posterior Distribution). *Given a WPTS Π in the form of (†), a designated initial program valuation $\mathbf{v}_{\text{init}}$ and a measurable subset $\mathcal{U} \in \Sigma_{\mathbb{R}^{|V_{\text{p}}|}}$, the* expected weight $[\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}_{\text{init}})$ *is defined as* $[\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}_{\text{init}}) := \mathbb{E}_{\mathbf{v}_{\text{init}}}[[\widehat{\mathbf{v}}_T \in \mathcal{U}] \cdot \widehat{w}_T]$. *(Recall that $\widehat{\mathbf{v}}_T, \widehat{w}_T$ are the random (vector) variables of the program valuation and the likelihood weight at termination, respectively. Thus, $[\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}_{\text{init}})$ is the expectation of $\widehat{w}_T$ over all runs that start from $(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$ and end with $\widehat{\mathbf{v}}_T \in \mathcal{U}$. We write $[\![\Pi]\!](\mathbf{v}_{\text{init}})$ iff $\mathcal{U} = \mathbb{R}^{|V_{\text{p}}|}$.) Then the* normalised posterior distribution *(NPD) $\mathsf{posterior}^{\Pi}$ of Π is defined by:*

$$\mathsf{posterior}^{\Pi}_{\mathcal{U}}(\mathcal{V}) := [\![\Pi]\!]_{\mathcal{U}}(\mathcal{V})/Z_{\Pi} \text{ for all measurable subsets } \mathcal{U}, \mathcal{V} \in \Sigma_{\mathbb{R}^{|V_{\text{p}}|}}$$

where $[\![\Pi]\!]_{\mathcal{U}}(\mathcal{V}) := \int_{\mathcal{V}} [\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}) \cdot \mu_{\text{init}}(\mathrm{d}\mathbf{v})$, *and* $Z_{\Pi} := [\![\Pi]\!](\mathbb{R}^{|V_{\text{p}}|})$ *is the* normalising constant. *The WPTS Π is called* integrable *if we have* $0 < Z_{\Pi} < \infty$.

[8] This follows the traditional setting in e.g. [3].

We consider to address the automated bound analysis for the NPD of a WPTS. To be more precise, we consider *interval bound* analysis that aims at deriving an interval $[l, u] \subseteq [0, \infty]$ for an integrable WPTS Π and measurable sets $\mathcal{U}, \mathcal{V} \in \Sigma_{\mathbb{R}^{|V_P|}}$ as tight as possible such that $l \leq [\![\Pi]\!]_{\mathcal{U}}(\mathcal{V}) \leq u$.

3 Motivating Examples

In this section, we have an overview of our novelties via two motivating examples.

Example 1 (Pedestrian Random Walk). Consider the pedestrian random walk example [28] written in our language in Fig. 2. In this example, a pedestrian is lost on a road, and she only knows that she is away from her house at most 3 km. Thus, she starts to repeatedly walk a uniformly random distance of at most 1 km in either direction, until reaching her house. Upon she arrives, an odometer tells that she has walked 1.1 km totally. However, this odometer was once broken and the measured distance is normally distributed around the true distance with standard deviation 0.1 km. In the program, the variable *pos* represents the current position of the pedestrian, the variable *step* holds the distance she walks in the next step, and the probabilistic branch in the loop body specifies that the pedestrian walks either forward or backward, both with probability 0.5. The variable *dis* records the total distance the pedestrian travelled so far.

This example is a non-score-recursive program and was previously handled in [3] by exhaustive recursion unrolling that has the path-explosion problem. In this work, we propose a novel fixed-point and template-based approach that synthesizes polynomial bounds for the NPD via a novel fixed-point theorem, a truncation operation that allows the polynomial synthesis to be over a bounded range. Our approach can obtain comparable bounds to [3] while our runtime is two-thirds of the time of [3] (see Section 6). □

Example 2 (Phylogenetic Birth Model). Consider a simplified version of the phylogenetic birth model [37]. A species arises with a birth-rate *lambda*, and it propagates with a constant likelihood of 1.1 at some time interval.[9] This example is modelled as a probabilistic program in Fig. 3. Assume that *lambda* is associated with a prior distribution, we want to infer its posterior distribution given the species evolvement described by the while loop. This example cannot be handled by previous approaches (such as [3,18]) since it is a score-recursive program with an unbounded while-loop and its scoring weight is greater than 1.

To see why such a score-recursive program is nontrivial to tackle, consider a simple loop example where in each loop iteration, the loop terminates with probability $\frac{1}{2}$, and continues to the next loop iteration with the same probability. At the end of each loop iteration, a score command "`score`(3)" is executed. Then the normalising constant is equal to $\sum_{n=1}^{\infty} \mathbb{P}(T = n) \cdot 3^n = \sum_{n=1}^{\infty} (\frac{3}{2})^n = \infty$, so that the infinity makes the posterior distribution invalid. One can observe that in this example the main problem lies at the fact that the scaling speed of the likelihood weight (i.e., 3) is higher than that for program termination (i.e., $\frac{1}{2}$).

[9] For simplicity, we assume constant weights that can be viewed as over-approximation for a continuous density function.

To address the phylogenetic birth model, our approach synthesizes polynomial bounds via a novel variant of Optional Stopping Theorem (OST) and again the truncation operation. Our experimental result shows that the derived bounds match the simulation result with 10^6 samples (see Section 6). □

```
start := sample uniform(0, 3);
pos := start; dist := 0;
while pos ≥ 0 do
  step := sample uniform(0, 1);
  if prob(0.5) then
    pos := pos − step
  else
    pos := pos + step
  fi;
  dist := dist + step
od;
score(normal(1.1, 0.1), dist);
return start
```

Fig. 2. A Pedestrian Random Walk

```
lambda := sample uniform(0, 2);
time := 10; amount := 0;
while time ≥ 0 do
  wait := sample uniform(0, 0.5);
  time := time − wait;
  if prob(0.5 · lambda) then
    birth := sample uniform(0, 0.01);
    amount := amount + birth;
    score(1.1)
  fi;
od;
return lambda
```

Fig. 3. A Phylogenetic Birth Model

4 Theoretical Approaches

In this section, we present our theoretical approaches, namely the fixed-point approach, the OST approach and the truncation operation.

4.1 The Fixed-Point Approach

We assume familiarity with basic concepts in fixed-point theory (refer to Appendix B.1 for details of these concepts). In this work, we apply the following well-known fixed-point theorem.

Theorem 1 (*Tarski* [45]). *Let $(K, \sqsubseteq)$ be a complete lattice and $f : K \to K$ a monotone function. Then, both* lfp f *and* gfp f *exist. Moreover, we have*

$$\text{lfp } f = \bigsqcap \{x \mid f(x) \sqsubseteq x\} \text{ and } \text{gfp } f = \bigsqcup \{x \mid x \sqsubseteq f(x)\}.$$

Based on Theorem 1, we then present our fixed-point approach. Our fixed-point approach works for non-score-recursive WPTS's. Below we fix a non-score-recursive WPTS Π in the form of (†). Given a maximum finite value $M \in [0, \infty)$, we say that a *state function* is a measurable function $h : \Lambda \to [-M, M]$ such that for all $\mathbf{v} \in \mathbb{R}^{|V_p|}$, we have that $h(\ell_{\text{out}}, \mathbf{v}) \in [0, M]$. We denote the set of all state functions with maximum value M by $\mathcal{K}_M$. We use the usual partial order $\leq$ on $\mathcal{K}_M$ defined in the pointwise fashion, i.e., for any $h_1, h_2 \in \mathcal{K}_M$, $h_1 \leq h_2$ iff $h_1(\Xi) \leq h_2(\Xi)$ for all $\Xi \in \Lambda$. Since (i) measurable functions are closed under both supremum and infimum and (ii) the top element $\top$ (resp. the bottom

element $\bot$) in the set $\mathcal{K}_M$ is the constant function that maps every state Ξ to M (resp. $-M$), one can straightforwardly verify that $(\mathcal{K}_M, \leq)$ is a complete lattice. To connect state functions with expected weights (Definition 2), we define the *expected-weight function* ew_{Π} by $ew_{\Pi}(\ell_{\text{init}}, \mathbf{v}) := [\![\Pi]\!](\mathbf{v})$, and omit the subscript Π if it is clear from the context. In this work, we concern the following monotone function over the complete lattice $(\mathcal{K}_M, \leq)$.

Definition 3 (Expected-Weight Transformers). *Given a maximum finite value $M \in [0, \infty)$, the* expected-weight transformer $ewt_{\Pi} : \mathcal{K}_M \to \mathcal{K}_M$ *is the higher-order function such that for each state function $h \in \mathcal{K}_M$ and state $(\ell, \mathbf{v})$, if $\tau = \langle \ell, \phi, F_1, \ldots, F_k \rangle$ is the unique transition that satisfies $\mathbf{v} \models \phi$ and $F_j = \langle \ell'_j, p_j, upd_j, wt_j \rangle$ for each $1 \leq j \leq k$, then we have that*

$$ewt_{\Pi}(h)(\ell, \mathbf{v}) = \begin{cases} \sum_{j=1}^{k} p_j \cdot \mathbb{E}_{\mathbf{r}} \left[wt_j(\mathbf{v}, \mathbf{r}) \cdot h(\ell'_j, upd_j(\mathbf{v}, \mathbf{r})) \right] & \text{if } \ell \neq \ell_{\text{out}} \\ 1 & \text{otherwise} \end{cases} . \tag{1}$$

Here the expectation $\mathbb{E}_{\mathbf{r}}[-]$ is taken over a sampling valuation $\mathbf{r}$ that observes the joint probability distributions of all the sampling variables $r \in V_{\text{r}}$.

We omit the subscript Π in ewt_{Π} if it is clear from the context. Informally, given a state function h, the expected-weight transformer ewt computes the expected weight $ewt(h)$ after one step of the WPTS execution. From the monotonicity of the Lebesgue integral, we have that ewt is monotone. Moreover, since for a non-score-recursive WPTS $wt_j(\mathbf{v}, \mathbf{r})$ is constantly 1 when $\ell_j^d \neq \ell_{\text{out}}$, we have that $ewt : \mathcal{K}_M \to \mathcal{K}_M$ is monotone when $M \geq 1$. The following theorem lays the backbone of our fixed-point approach.

Theorem 2. *Let Π be a non-score-recursive WPTS whose weights are bounded in $[-M, M]$ for a finite $M \geq 1$. Then the expected-weight function* ew *is the least fixed point of the expected-weight transformer* ewt *in the complete lattice $(\mathcal{K}_M, \leq)$. Furthermore, if the WPTS Π is AST, then the function* ew *is the unique fixed point of the higher-order function* ewt *in $(\mathcal{K}_M, \leq)$ when $M \geq 1$.*

By combining Theorem 2 and Theorem 1, it suffices to derive a prefixed point of ewt to obtain an upper bound for ew, and a postfixed point to obtain a lower bound in the case of almost-sure termination. For the space limitation, we relegate the detailed proof for Theorem 2 to Appendix B.2.

4.2 The OST Approach

For a score-recursive WPTS, the expected-weight transformer ewt may no longer be a function for the complete lattice $(\mathcal{K}_M, \leq)$. Hence, the fixed-point approach is not suitable for score-recursive WPTS's. To address this point, we propose a novel approach via a novel variant of OST. We first present the OST variant.

Theorem 3 (OST Variant). *Let $\{X_n\}_{n=0}^{\infty}$ be a martingale (resp. supermartingale) adapted to a filtration $\{\mathcal{F}_n\}_{n=0}^{\infty}$, and κ be a stopping time w.r.t. $\{\mathcal{F}_n\}_{n=0}^{\infty}$. Then the following condition is sufficient to ensure that $\mathbb{E}(|X_\kappa|) < \infty$ and $\mathbb{E}(X_\kappa) = \mathbb{E}(X_0)$ (resp. $\mathbb{E}(X_\kappa) \leq \mathbb{E}(X_0)$):*

($\mho$) *There exist integers* $b_1, b_2 > 0$ *and real numbers* $c_1 > 0, c_2 > c_3 > 0$ *such that (i)* $\mathbb{P}(\kappa > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ *for sufficiently large* $n \in \mathbb{N}$*, and (ii) for all* $n \in \mathbb{N}$*,* $|X_{n+1} - X_n| \leq b_1 \cdot n^{b_2} \cdot e^{c_3 \cdot n}$ *almost surely.*

Our OST variant extends the classical OST with the relaxation that we allow the magnitude of the next random variable X_{n+1} to be bounded by that of X_n by a multiplicative factor e^{c_3}. To cancel the effect of the multiplicative factor, we require in extra the exponential decrease in $\mathbb{P}(\kappa > n) \leq c_1 \cdot e^{-c_2 \cdot n}$. The proof resembles that of [51, Theorem 5.2] and is relegated to Appendix B.3.

Below we show how one applies our OST variant to score-recursive WPTS's. We fix a WPTS Π in the form of (†). The key concepts in the application of the OST variant are *potential weight functions* over the WPTS Π as follows. In the following definition, we use the expected-weight transformer *ewt* in the context of a score-recursive WPTS (i.e., wt_j may not be 1).

Definition 4 (Potential Weight Functions). *A* potential upper weight function *(PUWF) is a function* $h : L \times Val_{V_p} \to \mathbb{R}$ *that has the properties below:*

(C1) *for all reachable states* $(\ell, \mathbf{v})$ *with* $\ell \neq \ell_{\text{out}}$*, we have* $\text{ewt}(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$*;*
(C2) *for all states* $(\ell, \mathbf{v})$ *such that* $\ell = \ell_{\text{out}}$*, we have* $h(\ell, \mathbf{v}) = 1$*.*

Analogously, a potential lower weight function *(PLWF) is a function* $h : L \times Val_{V_p} \to \mathbb{R}$ *that satisfies the conditions (C1') and (C2), for which the condition (C1') is almost the same as (C1) except for that "*$\text{ewt}(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$*" is replaced with "*$\text{ewt}(h)(\ell, \mathbf{v}) \geq h(\ell, \mathbf{v})$*".*

Informally, a PUWF is a state function that satisfies the prefixed point condition of *ewt* at non-terminating locations, and equals one at the termination location. A PLWF is defined similarly, with the difference that we use the post-fixed point condition instead.

A WPTS Π has the *bounded update* property over its program variables, if there exists a constant $\varkappa > 0$ such that for every reachable state $(\ell, \mathbf{v})$ and its unique transition $\tau = \langle \ell, \phi, F_1, \ldots, F_k \rangle$ with each fork $F_j = \langle \ell'_j, p_j, upd_j, \text{wt}_j \rangle$, we have $\forall \mathbf{r} \in \mathsf{supp}(\mathcal{D}) \;\; \forall x \in V_{\text{p}}, \;\; |upd_j(\mathbf{v}, \mathbf{r})(x) - \mathbf{v}(x)| \leq \varkappa$.

By applying our OST variant (Theorem 3) in a way similar to [51, Theorem 6.10, Theorem 6.12], we obtain the main theorem for our OST approach. The proof is relegated to Appendix B.4.

Theorem 4. *Let* Π *be a score-recursive WPTS with bounded update. Suppose that there exist real numbers* $c_1 > 0$ *and* $c_2 > c_3 > 0$ *such that (i)* $\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ *for sufficiently large* $n \in \mathbb{N}$ *and (ii) for each score function* wt *in* Π *we have* $|\text{wt}| \leq e^{c_3}$*. Then for any PUWF (resp. PLWF)* h *over* Π*, we have that* $[\![\Pi]\!](\mathbf{v}_{\text{init}}) \leq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$ *(resp.* $[\![\Pi]\!](\mathbf{v}_{\text{init}}) \geq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$*) for any initial state* $(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$*.*

4.3 Truncation over WPTS's

Informally, the truncation of a WPTS restricts the value of every program variable in the WPTS to a prescribed bounded range $[-R, R]$ $(R > 0)$ (or

$[0, R], [-R, 0]$ if the value of a program variable is always non-negative or resp. non-positive). Note that every program variable can have its own bounded range.

Below we formally present the truncation over a WPTS. We say that a *truncating function* $\mathcal{B}$ is a function that maps every program variable $x \in V_{\mathrm{p}}$ to a bounded interval $\mathcal{B}(x)$ in $\mathbb{R}$ that specifies the bounded range of the variable x. We denote by $\Phi_{\mathcal{B}}$ the formula $\bigwedge_{x \in V_{\mathrm{p}}} x \in \mathcal{B}(x)$. We also consider a non-negative bound function $\mathcal{M} : \mathbb{R}^{|V_{\mathrm{p}}|} \to (0, \infty)$ that acts as an upper/lower bound for the expected weight $[\![\Pi]\!](\mathbf{v})$ when $\mathbf{v} \not\models \Phi_{\mathcal{B}}$.

Definition 5 (Truncation over WPTS's). *Let Π be a WPTS in the form of (†). Given a truncating function $\mathcal{B}$ and a bound function $\mathcal{M} > 0$, the* truncated *WPTS w.r.t. $\mathcal{B}$ and $\mathcal{M}$ is defined as $\Pi_{\mathcal{B},\mathcal{M}} := (V_{\mathrm{p}}, V_{\mathrm{r}}, L \cup \{\#\}, \ell_{\mathrm{init}}, \ell_{\mathrm{out}}, \mu_{\mathrm{init}}, \mathcal{D}, \mathfrak{T}_{\mathcal{B},\mathcal{M}})$ where $\#$ is a fresh deadlock location and the transition relation $\mathfrak{T}_{\mathcal{B},\mathcal{M}}$ is given by*

$$\begin{aligned}\mathfrak{T}_{\mathcal{B},\mathcal{M}} := {} & \{\langle \ell, \phi \wedge \Phi_{\mathcal{B}}, F_1, \ldots, F_k\rangle \mid \langle \ell, \phi, F_1, \ldots, F_k\rangle \in \mathfrak{T} \text{ and } \ell \neq \ell_{\mathrm{out}}\} \\ & \cup \{\langle \ell, \phi \wedge (\neg\Phi_{\mathcal{B}}), F_1^{\mathcal{M},\sharp}, \ldots, F_k^{\mathcal{M},\sharp}\rangle \mid \langle \ell, \phi, F_1, \ldots, F_k\rangle \in \mathfrak{T} \text{ and } \ell \neq \ell_{\mathrm{out}}\} \quad (\ddagger) \\ & \cup \{\langle \ell_{\mathrm{out}}, \mathbf{true}, F_{\ell_{\mathrm{out}}}\rangle, \langle \sharp, \mathbf{true}, F_\sharp\rangle\}\end{aligned}$$

for which $F_\ell := \langle \ell, 1, id, \overline{1}\rangle$ for $\ell \in \{\ell_{\mathrm{out}}, \sharp\}$ with id the identity function and $\overline{1}$ the constant function that always takes the value 1, and for a fork $F = \langle \ell', p, \mathrm{upd}, \mathrm{wt}\rangle$ we have $F^{\mathcal{M},\sharp} := F$ if $\ell' = \ell_{\mathrm{out}}$ and $F^{\mathcal{M},\sharp} := \langle \sharp, p, \mathrm{upd}, \mathcal{M}\rangle$ otherwise.

Thus, the truncated WPTS is obtained from the original one by first restraining each transition to the bounded range $\Phi_{\mathcal{B}}$ and then redirecting to the deadlock location $\sharp$ all the situations jumping out of the bounded range and not going to the termination location. To make the truncated WPTS deterministic and total, we add in extra the self-loop $\langle \sharp, \mathbf{true}, F_\sharp\rangle$. Our main theorem below shows that by choosing an appropriate bound function $\mathcal{M}$ in the truncation, one can obtain upper/lower approximation of the original WPTS (proof through Kleene iteration in Appendix C). Below we fix a WPTS Π and its truncated WPTS $\Pi_{\mathcal{B},\mathcal{M}}$ with a truncating function $\mathcal{B}$ and a bound function $\mathcal{M}$.

Theorem 5. *Suppose that (∗) for each truncation fork $F^{M,\sharp} = \langle \sharp, p, \mathrm{upd}, \mathcal{M}\rangle$ derived from some $F = \langle \ell', p, \mathrm{upd}, \mathrm{wt}\rangle$ with its source location ℓ (see (‡)), we have $[\![\Pi]\!](\mathbf{v}) \leq \mathcal{M}(\mathbf{v})$ for all $\mathbf{v}$ such that the state $(\ell, \mathbf{v})$ is reachable. Then $[\![\Pi]\!](\mathbf{v}_{\mathrm{init}}) \leq [\![\Pi_{\mathcal{B},\mathcal{M}}]\!](\mathbf{v}_{\mathrm{init}})$ for all initial program valuation $\mathbf{v}_{\mathrm{init}}$. Analogously, if it holds the condition (⋆) which is almost the same as (∗) except for that "$[\![\Pi]\!](\mathbf{v}) \leq \mathcal{M}(\mathbf{v})$" is replaced with "$[\![\Pi]\!](\mathbf{v}) \geq \mathcal{M}(\mathbf{v})$", then we have $[\![\Pi]\!](\mathbf{v}_{\mathrm{init}}) \geq [\![\Pi_{\mathcal{B},\mathcal{M}}]\!](\mathbf{v}_{\mathrm{init}})$ for all initial program valuation $\mathbf{v}_{\mathrm{init}}$.*

5 An Algorithmic Approach

In the following, we present an algorithmic implementation for our theoretical approaches to address the NPD problem. We assume polynomial density functions in every sampling statements. As the NPD is defined through expected weights (Definition 2), our algorithm first derives polynomial bounds for

expected weights, and then obtains bounds of the NPD. Moreover, our algorithm is template-based and has the following paradigm: (a) First obtain an upper bound function $\mathcal{M}_{\rm up}$ and a lower bound function $\mathcal{M}_{\rm low}$ on the expected weight outside the bounded range and truncate the input WPTS into a bounded range with bounds $\mathcal{M}_{\rm up}$ and $\mathcal{M}_{\rm low}$, whose correctness follows from Theorem 5; (b) Then synthesize polynomial bounds on the truncated WPTS via either our fixed-point approach (for a non-score-recursive WPTS) or OST approach (for a score-recursive WPTS), whose correctness follows from Theorem 2 or Theorem 4.

In our algorithm, we use the classical notion of invariants (see e.g. [7,43]) to over-approximate reachable states. We follow [14,43] to consider *affine invariants*. An affine invariant for a WPTS is a map I that assigns to each location ℓ a system $I(\ell)$ of affine inequalities such that for all reachable states $(\ell, \mathbf{v})$, the affine inequalities $I(\ell)$ holds under the values for program variables given in $\mathbf{v}$.

Below we fix an input WPTS Π in the form of (†) with an affine invariant I. In the case that Π is non-score-recursive, we define its *exit range* $\mathit{exit}(\Pi)$ as the set of all program valuations $\mathbf{v}$ such that the state $(\ell_{\rm out}, \mathbf{v})$ is reachable. Besides the input Π, our algorithm requires the following extra inputs:

- a truncating function $\mathcal{B}$ and two bound functions $\mathcal{M}_{\rm up}$ and $\mathcal{M}_{\rm low}$ such that they fulfill the conditions $(*)$ and $(\star)$ in the statement of Theorem 5;
- a positive integer d for the degree of the polynomial templates;
- in the case that Π is non-score-recursive, an error bound $\epsilon' > 0$ and a polynomial approximation g' for each non-polynomial score function g such that $|g(\mathbf{v}) - g'(\mathbf{v})| \leq \epsilon'$ for all program valuations $\mathbf{v} \in \mathit{exit}(\Pi)$ (see Theorem 8 in Appendix C);
- in the case that Π is score-recursive, positive integers and real numbers b_i, c_j $(i \in \{1,2\}, 1 \leq j \leq 3)$ such that $(\mho)$ in Theorem 3 holds.

We empirically specify a large enough bounded range of $\mathcal{B}$ over program variables that captures the major behaviour of the program. The bound functions $\mathcal{M}_{\rm up}, \mathcal{M}_{\rm low}$ could be either obtained by heuristics from the properties of the score functions in non-score-recursive programs, or derived using our OST variant (Theorem 4) by directly migrating existing template-based approaches [8,51,12,9] to our case. Polynomial approximations for continuous functions can be obtained by Matlab, while concentration bounds can be derived by the automated approaches in [12,48].

The first part of our algorithm synthesizes polynomial bounds for expected weights $[\![\Pi]\!](\mathbf{v}_{\rm init})$ when the initial probability distribution is Dirac at an initial program valuation $\mathbf{v}_{\rm init}$ (i.e., the initial program valuation is fixed to be $\mathbf{v}_{\rm init}$). Its algorithmic procedure (**Step A1** – **A5**) is as follows.

Step A1. The algorithm performs the truncation over the WPTS Π with the truncating function $\mathcal{B}$ and bound functions $\mathcal{M}_{\rm up}$ and $\mathcal{M}_{\rm low}$ to obtain the truncated WPTS's $\Pi_{\mathcal{B},\mathcal{M}_{\rm up}}$ and $\Pi_{\mathcal{B},\mathcal{M}_{\rm low}}$. We denote $B := \{\mathbf{v} \mid \mathbf{v}(x) \in \mathcal{B}(x) \text{ for all } x \in V_{\rm p}\}$.

Step A2. From the bounded range B, the algorithm computes an extended bounded range B' such that any program valuation from a one-step execution of the WPTS from some program valuation in B falls in B'. Formally, the extended bounded range B' should satisfy that for every location ℓ, program valuation

$\mathbf{v} \in B$, transition $\tau = \langle \ell, \phi, F_1, F_2, \ldots, F_k \rangle$, fork $F_j = \langle \ell'_j, p_j, upd_j, wt_j \rangle$ in τ and sampling valuation $\mathbf{r} \in \mathsf{supp}(\mathcal{D})$, we have that $upd_j(\mathbf{v}, \mathbf{r}) \in B'$. Our algorithm determines the extended bounded range B' by considering all possible executions of Π starting from the bounded range B [25,42]. The purpose to have a superset B' is to reduce the runtime in the solving of the template, see **Step A4** below.

Step A3. The algorithm sets up for each location $\ell \notin \{\ell_{\text{out}}, \sharp\}$ a d-degree polynomial template h_ℓ over the program variables V_{p} with unknown coefficients as parameters to be resolved. For $\ell \in \{\ell_{\text{out}}, \sharp\}$, our algorithms assumes $h_\ell \equiv 1$. Each template h_ℓ represents the desired bound function and is the summation of all monomials over V_{p} of degree no more than d for which each monomial has a standalone unknown coefficient. The unknown coefficients in different h_ℓ's are disjoint and all these unknown coefficients are to be resolved to obtain concrete bound functions.

Step A4. The algorithm establishes constraints for the templates h_ℓ's from (C1), (C1') and (C2) in Definition 4. Since both the fixed-point approach and the OST approach has the same constraints from the prefixed point and the postfixed point conditions, we consider unified constraints for the two approaches. For upper bounds on expected weights, the algorithm has the following constraints to synthesize a PUWF over $\Pi_{\mathcal{B},M_{up}}$:

(D1) For every location $\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$ and program valuation $\mathbf{v} \in I(\ell) \cap B$, we have that $ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$.

(D2) For every location $\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$ and program valuation $\mathbf{v} \in I(\ell) \cap (B' \setminus B)$, we have that $\mathcal{M}_{\text{up}}(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$.

For lower bounds, the algorithm has (D1') and resp. (D2') which are obtained from (D1) and resp. (D2) by replacing "$ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$" with "$ewt(h)(\ell, \mathbf{v}) \geq h(\ell, \mathbf{v})$" in (D1) and resp. "$\mathcal{M}_{\text{up}}(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$" with "$\mathcal{M}_{\text{low}}(\ell, \mathbf{v}) \geq h(\ell, \mathbf{v})$" in (D2), respectively. Note that (D1) and (D2) ensure (C1) and (C2) since $\mathcal{M}_{\text{up}} \leq h(\ell, \mathbf{v})$ implies that $ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$ for every location $\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$ and program valuation $\mathbf{v} \in I(\ell) \cap B$. The same holds for (D1') and (D2'). Note that in (D1), the calculation of the value $ewt(h)(\ell, \mathbf{v})$ has the piecewise nature that different sampling $\mathbf{r}$ may cause the next program valuation to be within or outside the bounded range, and to satisfy or violate the guards in the WPTS. In our algorithm, we have a refined treatment that enumerates all possible valid situations w.r.t different guards of the WPTS in the calculation of $ewt(h)(\ell, \mathbf{v})$ for a sampling valuation $\mathbf{r}$, and use (D2) to over-approximate the situations of whether the next program valuation lies in the bounded range or not in order to circumvent the piecewise difficulty.

Step A5. Our algorithm solves the unknown coefficients in the templates h_ℓ ($\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$ via the well-established methods of Putina[35] and Handelman's Positivstellensatz [22]. To be more detailed, the objective of this step is to solve the constraints from the previous step which is a conjunction of formulas of the form $\forall \mathbf{v} \in P.(\mathbf{g}(\mathbf{v}) \geq 0)$ where P is a polyhedron and $\mathbf{g}$ is a polynomial over V_{p} whose coefficients are affine expressions in the program variables, and such formulas can be guaranteed by the sound forms of Putinar's and Handelman's Positivstellensatz. The application of Putinar's Positivstellensatz results in semidefinite constraints and can be solved by semidefinite programming

(SDP), while the application of Handelman's theorem leads to linear constraints and can be solved by linear programming (LP). We refer to Appendix D for the details on the application of Putinar's and Handelman's theorem.

Given a measurable set $\mathcal{U} \in \Sigma_{\mathbb{R}^{|V_\mathrm{p}|}}$, the expected weight $[\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}_\mathrm{init})$ can be derived in the same manner above as one can always find an equivalent WPTS $\Pi_{\mathcal{U}}$ such that $[\![\Pi_{\mathcal{U}}]\!](\mathbf{v}_\mathrm{init}) = [\![\Pi]\!]_{\mathcal{U}}(\mathbf{v}_\mathrm{init})$ for any initial program valuation $\mathbf{v}_\mathrm{init}$.[10] The second part of our algorithm calls the first part of the algorithm to compute the bounds for the NPD. For the sake of simplicity, we only present the one-dimensional case, i.e., the WPTS has only one return variable $x \in V_\mathrm{p}$ of interest and $\mathbf{v}_\mathrm{init}[x] \sim \mu_\mathrm{init}$, the other elements in $\mathbf{v}_\mathrm{init}$ are fixed and constant. It can be straightforwardly extended to general cases.

Step B1. Given an input WPTS Π with $\mathbf{v}_\mathrm{init}[x] \sim \mu_\mathrm{init}$ and a measurable set $\mathcal{V} = \mathit{supp}(\mu_\mathrm{init}) \in \Sigma_\mathbb{R}$, our algorithm splits $\mathcal{V}$ uniformly into n intervals $I_1 = [a_1, b_1], \ldots, I_n = [a_n, b_n]$ and generates n different initial program valuations $\mathbf{v}^i_\mathrm{init}, \ldots, \mathbf{v}^n_\mathrm{init}$ such that each $\mathbf{v}^j_\mathrm{init}[x] := \frac{a_j + b_j}{2}$ ($j \in [1, n]$, $n > 0$ is an integer).

Step B2. Our algorithm calls the first part of the algorithm, and outputs a set $\mathit{Upper} = \{Up_1, \ldots, Up_n\}$ (resp. $\mathit{Lower} = \{Lw_1, \ldots, Lw_n\}$) of concrete polynomial upper bound functions (resp. lower bound functions) for expected weights $\{[\![\Pi]\!](\mathbf{v}^1_\mathrm{init}), \ldots, [\![\Pi]\!](\mathbf{v}^n_\mathrm{init})\}$. Then our algorithm integrates each bound function over its corresponding interval and summate the results to obtain the interval bound of the normalising constant $[\![\Pi]\!](\mathcal{V})$, i.e., $[\![\Pi]\!](\mathcal{V}) \in [l, u]$ where $u := \sum_{j=1}^{n} \int_{I_j} Up_j(\ell_\mathrm{init}, \mathbf{v}^j_\mathrm{init}) \mu_\mathrm{init}(\mathrm{d}\mathbf{v}^j_\mathrm{init}[x])$ and $l := \sum_{j=1}^{n} \int_{I_j} Lw_j(\ell_\mathrm{init}, \mathbf{v}^j_\mathrm{init}) \mu_\mathrm{init}(\mathrm{d}\mathbf{v}^j_\mathrm{init}[x])$.

Address the NPD problem. Given a measurable set $\mathcal{U} \in \Sigma_\mathbb{R}$, the interval bound of $[\![\Pi]\!]_{\mathcal{U}}(\mathcal{V})$ can be obtained similarly as one can always find an equivalent $\Pi_{\mathcal{U}}$ such that $[\![\Pi_{\mathcal{U}}]\!](\mathcal{V}) = [\![\Pi]\!]_{\mathcal{U}}(\mathcal{V})$. Then assume the interval bound of $[\![\Pi]\!]_{\mathcal{U}}(\mathcal{V})$ is $[l_{\mathcal{U}}, u_{\mathcal{U}}]$, its NPD $\mathsf{posterior}^{\Pi}_{\mathcal{U}}(\mathcal{V})$ is bounded by $[\frac{l_{\mathcal{U}}}{u}, \frac{u_{\mathcal{U}}}{l}]$ with $0 < l \leq u$.

6 Experimental Results

In this section, we present the experimental results of our approach over a variety of programs. First, we show that our approach can handle novel examples that cannot be addressed by other existing tools w.r.t. NPD. Then we compare our approach with the state-of-the-art tool GuBPI [3] w.r.t. NPD. Finally, even though the problem of path probability estimations is not the focus of our work, we demonstrate that our approach can work well for this problem, and we also compare the performance of our approach with GuBPI.

We implemented a parser from probabilistic programs to WPTS's in F#, our algorithms in Matlab, and used Mosek [1] for solving semidefinite programming. All results were obtained on an Intel Core i7-10875H (2.3 GHz) machine with 16 GB of memory, running MS Windows 10. Polynomial approximations for continuous functions were obtained by Matlab, while concentration bounds were derived by the automated approaches in [12,48].

[10] For instance, construct a $\Pi_{\mathcal{U}}$ by adding a conditional branch of the form "**if** $\mathbf{v}_T \in \mathcal{U}$ **then** `score`(0) **fi**" immediately before the termination of Π

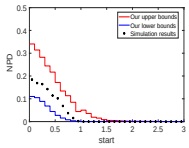
6.1 NPD - Novel Examples. We consider 10 novel examples adapted from the literature, where all 7 examples with prefix "Pd" or "RdWalk" are from [3], the two "RACE" examples are from [48], and the last example is from statistical phylogenetics [37] (see also Section 3). Concretely, the "RACE(V2)" and "BIRTH" examples are both score-recursive probabilistic programs with weights greater than 1, and thus their integrability condition should be verified by the existence of suitable concentration bounds (see Theorem 3); other examples are non-score-recursive probabilistic while loops with unsupported types of scoring by previous tools (e.g., polynomial scoring). Therefore, no existing static-analysis tools w.r.t. NPD can tackle these novel examples. The results are reported in Table 1, where the first column is the name of each example, the second column contains the parameter of each example used in our approach (i.e., the degree of the polynomial template and the bounded range of program variables), the third column is the used solver, and the fourth and fifth columns correspond to the runtime of upper and lower bounds computed by our approach, respectively. Our runtime is reasonable, that is, most examples can obtain tight bounds within 100 seconds, and the simulation results by Pyro [4] (10^6 samples per case) match our derived bounds. Due to the page limit, below we display part of the comparison in Fig. 4, see Appendix E for other figures.

Table 1. Results for Novel Examples

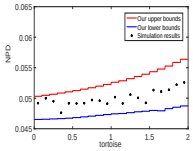
Benchmark	**Parameters**	**Solver**	**Upper**	**Lower**
			Time (s)	**Time (s)**
Pd(v1)	$d = 6$, $pos, dis \in [0, 5]$	SDP	54.65	52.39
Race(v1)	$d = 6$, $h, t \in [0, 5]$	LP	87.43	86.27
Race(v2)*	$d = 6$, $h, t \in [0, 5]$	LP	81.19	81.18
RdWalk(v1)	$d = 6$, $x, y \in [0, 5]$	LP	46.65	47.72
RdWalk(v2)	$d = 6$, $x, y \in [0, 5]$	LP	97.45	103.65
RdWalk(v3)	$d = 6$, $x, y \in [0, 5]$	LP	250.70	252.49
RdWalk(v4)	$d = 6$, $x, y \in [0, 5]$	LP	98.75	98.21
PdMB(v3)	$d = 4$, $pos, dis \in [0, 5]$	LP	15.26	14.57
PdMB(v4)	$d = 4$, $pos, dis \in [0, 5]$	LP	16.07	16.12
Birth*	$d = 6$, $lambda \in [0, 3], time \in [0, 10]$	LP	14.72	611.66

* It is a score-recursive probabilistic program with weights greater than 1.

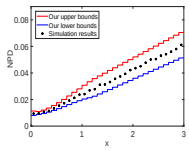
6.2 NPD - Comparision with GuBPI [3]. We consider the Pedestrian example "Pd" from [3] (see also Section 3), and its variants. More concretely, we enlarged the standard deviation of the observed normal distribution to be 5 for all other 6 examples whose prefix name are "Pd"; for the four "PdBeta" examples, we also add different beta distributions in the loop bodies. The last example is from [17]. We report the results in Table 2 whose layout is similar to Table 1 except that the column "#" displays whether or not the bounds are trivial, i.e., $[0, \infty]$. We also compare our results with GuBPI's and simulation results (10^6 samples per case), and show part of the comparison in Fig. 5, see Appendix E for other figures. Our runtime is up to 6 times faster than GuBPI while we can still obtain tighter or comparable bounds for all examples. Specifically, for the first example "Pd", our upper bounds are a bit higher than GuBPI's when the value of *start* falls into $[0, 0.7]$ (which is not suprising as the deviation of the



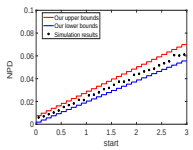
(a) Pd(v1)



(b) Race(v2)

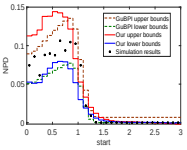


(c) RdWalk(v1)

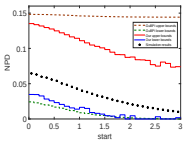


(d) PdMB(v3)

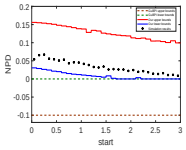
Fig. 4. NPD Bounds of Novel Examples



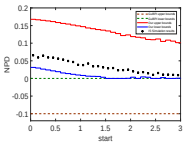
(a) Pd



(b) PdLD



(c) PdBeta(v1)



(d) PdBeta(v2)

Fig. 5. NPD Bounds of Comparison

The red lines and the blue lines mark the upper and lower bounds of our results; the black bold stars mark the simulation results; the brown dotted lines and the green dotted lines mark the upper and lower bounds by GuBPI (we denote by -0.1 the infinite bounds).

normal distribution in this example is quite small, i.e., 0.1, and our approach constructs over-approximation constraints while GuPBI uses recursion unrolling to search for the feasible space exhaustively), but our lower bounds are greater than GuBPI's, and our NPD bounds are tighter in the following.[11] For all 6 variants of "Pd" where the deviation of the normal distrbution is enlarged, our NPD bounds are tighter than GuBPI's, in particular, our upper bounds are much lower than GuBPI's. For the four "PdBeta" examples, we also found that GuBPI produced zero-valued unnormalized lower bounds, thus its results w.r.t. NPD are trivial, i.e., $[0, \infty]$. However, we can still produce non-trivial results and our runtime is at least 2 times faster than GuBPI.

6.3 Path Probability Estimation. We consider five recursive examples in [3], which were also cited from the PSI repository [17]. Since all five examples are non-paramteric and with unbounded numbers of loop iterations, PSI cannot handle them correctly as mentioned in [3]. We estimated the path probability of certain events, i.e., queries over program variables, and report the results in Table 3. For the first three examples, we obtained tighter lower bounds than GuBPI and same upper bounds, while our runtime is at least 2 times faster than GuBPI. Moreover, we found a potential error of GuBPI. That is, the fourth example "cav-ex-5" in Table 3 is an AST program with no scores, which means its normalizing constant should be exactly one. However, the upper bound of the normailising constant obtained by GuBPI is smaller than 1 (i.e., 0.6981). A stochastic simulation using 10^6 samples yielded the results that fall within our bounds but violate those computed by GuBPI. Thus, GuBPI possibly omitted

[11] When the value of *start* approaches 3, our NPD bounds is close to zero, but the upper bounds may be lower than zero, which is caused by numerical issues of semi-definite programming. The problem of numerical issues is orthogonal to our work and remains to be addressed in both academic and industrial fields.

Table 2. Comparison with GuBPI

Benchmark	Our Tool				GuBPI	
	Parameters	Solver	Time (s)	#	Time (s)	#
Pd	$d = 10$, $pos, dis \in [0, 5]$	SDP	3176.685	•	5266.063	•
PdLD	$d = 6$, $pos, dis \in [0, 5]$	LP	41.99	•	648.151	•
PdBeta(v1)	$d = 6$, $pos, dis \in [0, 5]$	LP	99.86	•	645.055	○
PdBeta(v2)	$d = 6$, $pos, dis \in [0, 5]$	LP	228.43	•	653.237	○
PdBeta(v3)	$d = 6$, $pos, dis \in [0, 5]$	LP	101.36	•	657.645	○
PdBeta(v4)	$d = 6$, $pos, dis \in [0, 5]$	LP	208.86	•	686.207	○
PdMB(v5)	$d = 6$, $pos, dis \in [0, 5]$	LP	88.41	•	391.772	•
Para-recur	$d = 8$, $p \in [0, 1]$	LP	36.61	•	253.728	•

* ○ marks the trivial bound $[0, \infty]$, while • marks the non-trivial ones.

some valid program runs of this example and produces wrong results. All our results match the simulation results (10^6 samples per case).

Table 3. Results for Path Probability Estimation

Benchmark	Query	Our Tool			GuBPI		Simul
		Parameters	Time (s)	Bounds	Time (s)	Bounds	
cav-ex-7	Q1	$6, [0, 30], [0, 4]$	15.062	$[0.9698, 1.0000]$	38.834	$[0.7381, 1.0000]$	0.9938
	Q2	$6, [0, 40], [0, 4]$	16.321	$[0.9985, 1.0000]$	37.651	$[0.7381, 1.0000]$	0.9993
AddUni(L)	Q1	$6, [0, 10], [0, 1]$	8.85	$[0.9940, 1.0000]$	21.064	$[0.9375, 1.0000]$	0.9991
	Q2	$6, [0, 15], [0, 1]$	8.80	$[0.9995, 1.0000]$	14.941	$[0.9375, 1.0000]$	0.9999
RdBox	Q1	$4, [-0.8, 0.8], [0, 10]$	25.87	$[0.9801, 1.0000]$	173.535	$[0.9462, 1.0000]$	0.9999
cav-ex-5 *	Q1	$6, [20, \infty], [0, 10]$	33.17	$[0.8123, 0.9707]$	229.623	$[0.5768, 0.6374]$	0.9098
	Q2	$6, [20, \infty], [0, 20]$	54.373	$[0.8970, 1.0000]$	224.504	$[0.5768, 0.6375]$	0.9645
GWalk **	Q1	$8, [1, \infty], [0, 0.1]$	7.255	$[0.0023, 0.0023]$	33.246	$[0.0023, 0.0024]$	0.0023
	Q2	$8, [1, \infty], [0, 0.2]$	8.197	$[0.0025, 0.0025]$	31.728	$[0.0025, 0.0025]$	0.0025

* GuBPI's result contradicts ours, and we found GuBPI produces wrong results for this example.
** As we care about path probabilities, we compared bounds of unnormalized distributions for this example (the NPD can be derived in the same manner above).

7 Related Works

Below we compare our results with the most related work in the literature.

Static analysis in Bayesian probabilistic programming. There are a lot of works on NPD inference for probabilistic programs, such as (λ)PSI [17,18], AQUA [23], Hakaru [32] and SPPL [40]. However, these methods are restricted to specific kinds of programs, e.g., programs with closed-form solutions to NPD or without continuous distributions, and none of them can handle probabilistic programs with unbounded while-loops/recursion. As far as we know, the most revelant work on static analysis of posterior distribution over unbounded loops/recursion is the approach [3] that infers the bounds for posterior distributions by recursion unrolling and bounding the non-termination case via the widening operator of abstract interpretation. By unrolling recursion to arbitrary depth, this approach can achieve high precision on the derive bounds. However, a major drawback of this approach is that the recursion unrolling may cause path explosion. Our approach circumvents the path explosion problem by constraint solving. Another

major drawback is that this approach cannot handle score-recursive programs as simply applying the approach to score-recursive programs leads to the trivial bound $[0, \infty]$, and we address this issue by a novel OST variant.

MCMC and variational inference. As mentioned previously, statistical approaches such as MCMC [38,16] and variational inference [5] cannot provide formal guarantee on the bounds for posterior distributions in a finite time limit. In contrast, our approach has formal guarantee on the derived bounds.

Static analysis of probabilistic programs. In recent years, there have been an abundance of works on static analysis of probabilistic programs. Most of them address fundamental aspects such as termination [8,11,15], sensitivity [2,50], expectation [33,51], tail bounds [26,47,49], assertion probability [42,48], etc. Compared with these results, we have:

- Our work focuses on normalized posterior distribution in Bayesian probabilistic programming, and hence is an orthogonal objective.
- Our algorithm follows the previous works on the synthesis of polynomial templates [8,51,12,9], but we have a truncation operation to increase the accuracy which to our best knowledge is novel.
- Our approach extends the classical OST as the previous works [51,47] do, but we consider a multiplicative variant, while the work [51] considers only an additive variant, and the work [47] considers a general extension through the uniform integrability condition and an implementation via polynomial functions, but does not have a detailed treatment for a multiplicative variant.

References

1. ApS, M.: The MOSEK optimization toolbox for MATLAB manual. Version 10.0. (2022), http://docs.mosek.com/10.0/toolbox/index.html
2. Barthe, G., Espitau, T., Grégoire, B., Hsu, J., Strub, P.: Proving expected sensitivity of probabilistic programs. Proc. ACM Program. Lang. **2**(POPL), 57:1–57:29 (2018). https://doi.org/10.1145/3158145
3. Beutner, R., Ong, C.L., Zaiser, F.: Guaranteed bounds for posterior inference in universal probabilistic programming. In: Jhala, R., Dillig, I. (eds.) PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022. pp. 536–551. ACM (2022). https://doi.org/10.1145/3519939.3523721
4. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P.A., Horsfall, P., Goodman, N.D.: Pyro: Deep universal probabilistic programming. J. Mach. Learn. Res. **20**, 28:1–28:6 (2019), http://jmlr.org/papers/v20/18-403.html
5. Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: A review for statisticians. Journal of the American statistical Association **112**(518), 859–877 (2017)
6. Borgström, J., Lago, U.D., Gordon, A.D., Szymczak, M.: A lambda-calculus foundation for universal probabilistic programming. In: Garrigue, J., Keller, G., Sumii, E. (eds.) Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016. pp. 33–46. ACM (2016). https://doi.org/10.1145/2951913.2951942
7. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: CAV 2013. pp. 511–526 (2013)
8. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 511–526. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_34
9. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through positivstellensatz's. In: CAV 2016. pp. 3–22 (2016)
10. Chatterjee, K., Fu, H., Goharshady, A.K., Goharshady, E.K.: Polynomial invariant generation for non-deterministic recursive programs. In: Donaldson, A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020. pp. 672–687. ACM (2020). https://doi.org/10.1145/3385412.3385969
11. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In: Bodík, R., Majumdar, R. (eds.) Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016. pp. 327–342. ACM (2016). https://doi.org/10.1145/2837614.2837639
12. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. ACM Trans. Program. Lang. Syst. **40**(2), 7:1–7:45 (2018). https://doi.org/10.1145/3174800
13. Chatterjee, K., Novotný, P., Žikelić, Đ.: Stochastic invariants for probabilistic termination. In: POPL 2017. pp. 145–160 (2017)

14. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: Jr., W.A.H., Somenzi, F. (eds.) Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2725, pp. 420–432. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_39
15. Fu, H., Chatterjee, K.: Termination of nondeterministic probabilistic programs. In: Enea, C., Piskac, R. (eds.) Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11388, pp. 468–490. Springer (2019). https://doi.org/10.1007/978-3-030-11245-5_22
16. Gamerman, D., Lopes, H.F.: Markov chain Monte Carlo: stochastic simulation for Bayesian inference. CRC press (2006)
17. Gehr, T., Misailovic, S., Vechev, M.T.: PSI: exact symbolic inference for probabilistic programs. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9779, pp. 62–83. Springer (2016). https://doi.org/10.1007/978-3-319-41528-4_4
18. Gehr, T., Steffen, S., Vechev, M.T.: λpsi: exact inference for higher-order probabilistic programs. In: Donaldson, A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020. pp. 883–897. ACM (2020). https://doi.org/10.1145/3385412.3386006
19. Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K.A., Tenenbaum, J.B.: Church: a language for generative models. In: McAllester, D.A., Myllymäki, P. (eds.) UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008. pp. 220–229. AUAI Press (2008)
20. Goodman, N.D., Stuhlmüller, A.: The Design and Implementation of Probabilistic Programming Languages. http://dippl.org (2014)
21. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Future of Software Engineering Proceedings, pp. 167–181 (2014)
22. Handelman, D.: Representing polynomials by positive linear functions on compact convex polyhedra. Pacific Journal of Mathematics **132**(1), 35–62 (1988)
23. Huang, Z., Dutta, S., Misailovic, S.: AQUA: automated quantized inference for probabilistic programs. In: Hou, Z., Ganesh, V. (eds.) Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12971, pp. 229–246. Springer (2021). https://doi.org/10.1007/978-3-030-88885-5_16
24. Jeffreys, H.: "weierstrass's theorem on approximation by polynomials" and "extension of weierstrass's approximation theory". Methods of Mathematical Physics pp. 446–448 (1988)
25. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252
26. Kura, S., Urabe, N., Hasuo, I.: Tail probabilities for randomized program runtimes via martingales for higher moments. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 135–153. Springer (2019)
27. Lee, W., Yu, H., Rival, X., Yang, H.: Towards verified stochastic variational inference for probabilistic programs. Proc. ACM Program. Lang. **4**(POPL), 16:1–16:33 (2020). https://doi.org/10.1145/3371084
28. Mak, C., Ong, C.L., Paquet, H., Wagner, D.: Densities of almost surely terminating probabilistic programs are differentiable almost everywhere. In: Yoshida, N. (ed.)

Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12648, pp. 432–461. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_16
29. McIver, A., Morgan, C.: Developing and reasoning about probabilistic programs in *pGCL*. In: Cavalcanti, A., Sampaio, A., Woodcock, J. (eds.) Refinement Techniques in Software Engineering, First Pernambuco Summer School on Software Engineering, PSSE 2004, Recife, Brazil, November 23-December 5, 2004, Revised Lectures. Lecture Notes in Computer Science, vol. 3167, pp. 123–155. Springer (2004). https://doi.org/10.1007/11889229_4
30. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science, Springer (2005). https://doi.org/10.1007/b138392
31. van de Meent, J., Paige, B., Yang, H., Wood, F.: An introduction to probabilistic programming. CoRR **abs/1809.10756** (2018)
32. Narayanan, P., Carette, J., Romano, W., Shan, C., Zinkov, R.: Probabilistic inference by program transformation in hakaru (system description). In: Kiselyov, O., King, A. (eds.) Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9613, pp. 62–79. Springer (2016). https://doi.org/10.1007/978-3-319-29604-3_5
33. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: resource analysis for probabilistic programs. In: Foster, J.S., Grossman, D. (eds.) Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018. pp. 496–512. ACM (2018). https://doi.org/10.1145/3192366.3192394
34. Pollard, D.: A user's guide to measure theoretic probability. No. 8, Cambridge University Press (2002)
35. Putinar, M.: Positive polynomials on compact semi-algebraic sets. Indiana University Mathematics Journal **42**(3), 969–984 (1993), http://www.jstor.org/stable/24897130
36. Rankin, R.: Real and complex analysis. by w. rudin. pp. 412. 84s. 1966.(mcgraw-hill, new york.). The Mathematical Gazette **52**(382), 412–412 (1968)
37. Ronquist, F., Kudlicka, J., Senderov, V., Borgström, J., Lartillot, N., Lundén, D., Murray, L., Schön, T.B., Broman, D.: Universal probabilistic programming offers a powerful approach to statistical phylogenetics. Communications biology **4**(1), 1–10 (2021)
38. Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo method, vol. 10. John Wiley & Sons (2016)
39. Rudin, W., et al.: Principles of mathematical analysis, vol. 3. McGraw-hill New York (1976)
40. Saad, F.A., Rinard, M.C., Mansinghka, V.K.: SPPL: probabilistic programming with fast exact symbolic inference. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021. pp. 804–819. ACM (2021). https://doi.org/10.1145/3453483.3454078
41. Sangiorgi, D.: Introduction to bisimulation and coinduction. Cambridge University Press (2011)
42. Sankaranarayanan, S., Chakarov, A., Gulwani, S.: Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In: Boehm, H., Flanagan, C. (eds.) ACM SIGPLAN Conference on Programming Language

Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013. pp. 447–458. ACM (2013). https://doi.org/10.1145/2491956.2462179
43. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constraint-based linear-relations analysis. In: Giacobazzi, R. (ed.) Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3148, pp. 53–68. Springer (2004). https://doi.org/10.1007/978-3-540-27864-1_7
44. Staton, S., Yang, H., Wood, F.D., Heunen, C., Kammar, O.: Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 525–534. ACM (2016). https://doi.org/10.1145/2933575.2935313
45. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific journal of Mathematics **5**(2), 285–309 (1955)
46. Tolpin, D., van de Meent, J., Wood, F.D.: Probabilistic programming in anglican. In: Bifet, A., May, M., Zadrozny, B., Gavaldà, R., Pedreschi, D., Bonchi, F., Cardoso, J.S., Spiliopoulou, M. (eds.) Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III. Lecture Notes in Computer Science, vol. 9286, pp. 308–311. Springer (2015). https://doi.org/10.1007/978-3-319-23461-8_36
47. Wang, D., Hoffmann, J., Reps, T.W.: Central moment analysis for cost accumulators in probabilistic programs. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021. pp. 559–573. ACM (2021). https://doi.org/10.1145/3453483.3454062
48. Wang, J., Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Quantitative analysis of assertion violations in probabilistic programs. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021. pp. 1171–1186. ACM (2021). https://doi.org/10.1145/3453483.3454102
49. Wang, P.: Tail-bound cost analysis over nondeterministic probabilistic programs. Journal of Shanghai Jiaotong University (Science) pp. 1–11 (2022)
50. Wang, P., Fu, H., Chatterjee, K., Deng, Y., Xu, M.: Proving expected sensitivity of probabilistic programs with randomized variable-dependent termination time. Proc. ACM Program. Lang. **4**(POPL), 25:1–25:30 (2020). https://doi.org/10.1145/3371093
51. Wang, P., Fu, H., Goharshady, A.K., Chatterjee, K., Qin, X., Shi, W.: Cost analysis of nondeterministic probabilistic programs. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 204–220. PLDI 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3314221.3314581
52. Williams, D.: Probability with martingales. Cambridge university press (1991)

A Supplementary Material for Section 2

A.1 Basics of Probability Theory

A *measurable space* is a pair (U, Σ_U), where U is a nonempty set and Σ_U is a σ-algebra on U, i.e., a family of subsets of U such that $\Sigma_U \subseteq \mathcal{P}(U)$ contains $\emptyset$ and is closed under complementation and countable union. Elements of Σ_U are called *measurable* sets. A function f from a measurable space (U_1, Σ_{U_1}) to another measurable space (U_2, Σ_{U_2}) is *measurable* if $f^{-1}(A) \in \Sigma_{U_1}$ for all $A \in \Sigma_{U_2}$.

A *measure* μ on a measurable space (U, Σ_U) is a mapping from Σ_U to $[0, \infty]$ such that (i) $\mu(\emptyset) = 0$ and (ii) μ is countably additive: for every pairwise-disjoint set sequence $\{A_n\}_{n\in\mathbb{N}}$ in Σ_U, it holds that $\mu(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$. We call the triple (U, Σ_U, μ) a *measure space*. If $\mu(U) = 1$, we call μ a *probability measure*, and (U, Σ_U, μ) a *probability space*. The Lebesgue measure λ is the unique measure on $(\mathbb{R}, \Sigma_{\mathbb{R}})$ satisfying $\lambda([a, b)) = b - a$ for all valid intervals $[a, b)$ in $\Sigma_{\mathbb{R}}$. For each $n \in \mathbb{N}$, we have a measurable space $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$ and a unique product measure λ_n on $\mathbb{R}^n$ satisfying $\lambda_n(\prod_{i=1}^n A_i) = \prod_{i=1}^n \lambda(A_i)$ for all $A_i \in \Sigma_{\mathbb{R}}$.

The *Lebesgue* integral operator $\int$ is a partial operator that maps a measure μ on (U, Σ_U) and a real-valued function f on the same space (U, Σ_U) to a real number or infinity, which is denoted by $\int f \mathrm{d}\mu$ or $\int f(x)\mu(\mathrm{d}x)$. The detailed definition of Lebesgue integral is somewhat technical, see [36,39] for more details. Given a measurable set $A \in \Sigma_U$, the integral of f over A is defined by $\int_A f(x)\mu(\mathrm{d}x) := \int f(x) \cdot [x \in A]\mu(\mathrm{d}x)$ where $[-]$ is the Iverson bracket such that $[\phi] = 1$ if ϕ is true, and 0 otherwise. If μ is a probability measure, then we call the integral as the *expectation* of f, denoted by $\mathbb{E}_{x\sim\mu;A}[f]$, or $\mathbb{E}[f]$ when the scope is clear from the context.

For a measure v on (U, Σ_U), a measurable function $f : U \to \mathbb{R}_{\geq 0}$ is the *density* of v with respect to μ if $v(A) = \int f(x) \cdot [x \in A]\mu(\mathrm{d}x)$ for all measurable $A \in \Sigma_U$, and μ is called the *reference measure* (most often μ is the Lebesgue measure). Common families of probability distributions on the reals, e.g., uniform, normal distributions, are measures on $(\mathbb{R}, \Sigma_{\mathbb{R}})$. Most often these are defined in terms of probability density functions with respect to the Lebesgue measure. That is, for each μ_D there is a measurable function $\mathrm{pdf}_D : \mathbb{R} \to \mathbb{R}_{\geq 0}$ that determines it: $\mu_D(A) := \int_A \mathrm{pdf}_D(\mathrm{d}\lambda)$. As we will see, density functions such as pdf_D play an important role in Bayesian inference.

Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a *random variable* is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R} \cup \{+\infty, -\infty\}$. The expectation of a random variable X, denoted by $\mathbb{E}(X)$, is the Lebesgue integral of X w.r.t. $\mathbb{P}$, i.e., $\int X \, \mathrm{d}\mathbb{P}$. A *filtration* of $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence $\{\mathcal{F}_n\}_{n=0}^{\infty}$ such that for every $n \geq 0$, the triple $(\Omega, \mathcal{F}_n, \mathbb{P})$ is a probability space and $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$. A *stopping time* w.r.t. $\{\mathcal{F}_n\}_{n=0}^{\infty}$ is a random variable $T : \Omega \to \mathbb{N} \cup \{0, \infty\}$ such that for every $n \geq 0$, the event $\{T \leq n\}$ is in $\mathcal{F}_n$.

A *discrete-time stochastic process* is a sequence $\Gamma = \{X_n\}_{n=0}^{\infty}$ of random variables in $(\Omega, \mathcal{F}, \mathbb{P})$. The process Γ is *adapted* to a filtration $\{\mathcal{F}_n\}_{n=0}^{\infty}$, if for all $n \geq 0$, X_n is a random variable in $(\Omega, \mathcal{F}_n, \mathbb{P})$. A discrete-time stochastic process $\Gamma = \{X_n\}_{n=0}^{\infty}$ adapted to a filtration $\{\mathcal{F}_n\}_{n=0}^{\infty}$ is a *martingale* (resp. *supermartingale*, *submartingale*) if for all $n \geq 0$, $\mathbb{E}(|X_n|) < \infty$ and it holds almost surely (i.e., with probability 1) that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] = X_n$ (resp.

$\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \leq X_n$, $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \geq X_n$). See [52] for details. Applying martingales to qualitative and quantitative analysis of probabilistic programs is a well-studied technique [7,9,13].

A.2 Details for WPTS Semantics

We denote by $\mathit{\Lambda}$ the set of all states, by $\mathit{\Delta}$ the set of all weighted states, and by $\Sigma_{\mathit{\Delta}}$ the product σ-algebra (on $\mathit{\Delta}$) among the discrete σ-algebra $(L, 2^L)$ for locations, the σ-algebra $\Sigma_{\mathbb{R}^{|V_\mathrm{p}|}}$ for program valuations, and the σ-algebra $\Sigma_{\mathbb{R}}$ for the multiplicative likelihood weight. We define $\Sigma^n_{\mathit{\Delta}}$ (for $n \geq 1$) as the set $\{A_1 \times \cdots \times A_n \mid \forall 1 \leq i \leq n.(A_i \in \Sigma_{\mathit{\Delta}})\}$, and $\mathit{\Delta}^\infty$ as the set of all infinite sequences of weighted states.

The probability space for the WPTS $\mathit{\Pi}$ is defined such that its sample space is the set of all program runs, its σ-algebra is generated by the countable union $\bigcup_{n\geq 1}\{B \times \mathit{\Delta}^\infty \mid B \in \Sigma^n_{\mathit{\Delta}}\})$, and its probability measure $\mathbb{P}$ is the unique one such that (i) $\mathbb{P}(A \times \mathit{\Lambda}^\infty) = \mu_{\mathrm{init}}(\{\mathbf{v} \mid (\ell_{\mathrm{init}}, \mathbf{v}, 1) \in A\})$ for all $A \in \Sigma_{\mathit{\Delta}}$, and (ii) $\mathbb{P}(A \times B \times \mathit{\Delta}^\infty)$ (for every $A \in \Sigma_{\mathit{\Delta}}, B \in \Sigma^n_{\mathit{\Delta}}$ $(n \geq 1)$) equals the probability w.r.t the sampling of μ_{init} (for the initial program valuation) and $\mathcal{D}$ (for a sampling valuation in each step until the $(n+1)$-th step) that a program run $\{\mathit{\Theta}_n\}_{n\geq 0}$ is subject to $\mathit{\Theta}_0 \in A$ and $(\mathit{\Theta}_1, \ldots, \mathit{\Theta}_{n+1}) \in B$. For each program valuation $\mathbf{v}$, we denote by $\mathbb{P}_{\mathbf{v}}$ the probability measure of $\mathit{\Pi}$ when the initial distribution is changed to the Dirac distribution at $\mathbf{v}$.

A.3 Sampling-based Semantics

We recall one prominent semantics in the literature, i.e., the sampling-based semantics [6,44]. We show that the transition-based semantics in our work is equivalent to the widely-used sampling-based semantics in Bayesian statistical probabilistic programming.

The sampling-based semantics by Borgström et al. [6] interprets a probabilistic program as a deterministic program parameterized by a sequence of random draws sampled during the execution of the program.

A *sampling trace* is a finite sequence $\boldsymbol{t} = \langle r_1, \ldots, r_n \rangle$ of real numbers, and we define $\mathcal{T} := \bigcup_{n\in\mathbb{N}} \mathbb{R}^n$ as the set of all sampling traces. Given a probabilistic program P, a *configuration* σ under the semantics is a tuple $\langle \mathbf{v}, S, w, \boldsymbol{t} \rangle$ where $\mathbf{v} \in \mathit{Val}_{V_\mathrm{p}}$, S is the statement to be executed, $w \in [0, \infty)$ is the global weight variable whose value expresses how well the current computation matches the observations, and $\boldsymbol{t}$ is a sampling trace. We denote by $\mathit{\Sigma}$ the set of all configurations.

The semantics operates on the configurations, where an execution of the program is initialized with $\sigma_0 = \langle \mathbf{v}_0, P, 1, \boldsymbol{t} \rangle$, and the termination configurations have the form of $\langle \_, \mathbf{skip}, \_, [] \rangle$, for which $\_$ is a "wildcard" character that matches everything. Fig. 6 shows the corresponding one-step reduction relation $\rightarrow$ (note that $\Downarrow$ is the usual big-step semantics for deterministic Boolean and arithmetic expressions, so we omit it here).

Let $\rightarrow^*$ be the reflexive transitive closure of the one-step reduction $\rightarrow$ in Fig. 6. Given a probabilistic program P, we call a sampling trace $\boldsymbol{t}$ *terminating*

if $\langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle$ for some valuations $\mathbf{v}, \mathbf{v}' \in \mathit{Val}_{V_\mathrm{p}}$ and weight $w \in \mathbb{R}_{\geq 0}$, i.e., the program P terminates under the samples drawn as in $\boldsymbol{t}$.

$$\frac{}{\langle \mathbf{v}, \mathbf{skip}, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, \mathbf{skip}, w, \boldsymbol{t}\rangle} \qquad \frac{\mathbf{v} \vdash E \Downarrow r}{\langle \mathbf{v}, x := E, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}[x \mapsto r], \mathbf{skip}, w, \boldsymbol{t}\rangle}$$

$$\frac{\mathbf{v} \vdash B \Downarrow b,\ b = \mathbf{true}}{\langle \mathbf{v}, \mathbf{if}\, B\, \mathbf{then}\, S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, S_1, w, \boldsymbol{t}\rangle}$$

$$\frac{\mathbf{v} \vdash B \Downarrow b,\ b = \mathbf{false}}{\langle \mathbf{v}, \mathbf{if}\, B\, \mathbf{then}\, S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, S_2, w, \boldsymbol{t}\rangle}$$

$$\frac{w' = \mathrm{pdf}_D(r) \geq 0}{\langle \mathbf{v}, x := \mathbf{sample}\ D, w, r :: \boldsymbol{t}\rangle \to \langle \mathbf{v}[x \mapsto r], \mathbf{skip}, w \cdot w', \boldsymbol{t}\rangle}$$

$$\frac{\mathbf{v} \vdash B \Downarrow b,\ b = \mathbf{true}}{\langle \mathbf{v}, \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, S; \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, w, \boldsymbol{t}\rangle}$$

$$\frac{\mathbf{v} \vdash B \Downarrow b,\ b = \mathbf{false}}{\langle \mathbf{v}, \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, \mathbf{skip}, w, \boldsymbol{t}\rangle}$$

$$\frac{w' = \mathrm{pdf}_D(x) \geq 0}{\langle \mathbf{v}, \mathbf{observe}(x, D), w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, \mathbf{skip}, w \cdot w', \boldsymbol{t}\rangle}$$

$$\frac{\langle \mathbf{v}, S_1, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}', S_1', w', \boldsymbol{t}'\rangle,\ S_1' \neq \mathbf{skip}}{\langle \mathbf{v}, S_1; S_2, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}', S_1'; S_2, w', \boldsymbol{t}'\rangle}$$

$$\frac{\langle \mathbf{v}, S_1, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}', S_1', w', \boldsymbol{t}'\rangle,\ S_1' = \mathbf{skip}}{\langle \mathbf{v}, S_1; S_2, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}', S_2, w', \boldsymbol{t}'\rangle}$$

$$\frac{}{\langle \mathbf{v}, \mathbf{return}\ x, w, \boldsymbol{t}\rangle \to \langle \mathbf{v}, \mathbf{skip}, w, t\rangle}$$

Fig. 6. One-step reduction for probabilistic programs.

Below we define the notion of posterior distributions by the sampling-based semantics. From the one-step reduction rules (in Fig. 6), we can reason about the global behavior of probabilistic programs in terms of the sampling traces they produce. That is, given a probabilistic program P, and a terminating trace $\boldsymbol{t}$ such that $\langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle$ for valuations $\mathbf{v}, \mathbf{v}' \in \mathit{Val}_{V_\mathrm{p}}$ and weight $w \in \mathbb{R}_{\geq 0}$, we define the *value function* val_P and the *weight function* wt_P as follows:

$$\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}) = \begin{cases} \mathbf{v}' & \text{if } \langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle \\ \text{unspecified} & \text{otherwise,} \end{cases} \tag{2}$$

$$\mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) = \begin{cases} w & \text{if } \langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Moreover, we denote the return variable by $\mathsf{val}_{P,ret}(\mathbf{v}, \boldsymbol{t})$, i.e., $\mathsf{val}_{P,ret}(\mathbf{v}, \boldsymbol{t}) := \mathbf{v}'[ret]$. We also consider the measure space $(\mathcal{T}, \Sigma_{\mathcal{T}}, \mu_{\mathcal{T}})$ where $\mathcal{T} = \bigcup_{n\in\mathbb{N}} \mathbb{R}^n$ (as mentioned previously), $\Sigma_{\mathcal{T}} := \{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \Sigma_{\mathbb{R}^n}\}$ and $\mu_{\mathcal{T}}(U) := \sum_{n\in\mathbb{N}} \lambda_n(U \cap \mathbb{R}^n)$. By definition, the measure space $(\mathcal{T}, \Sigma_{\mathcal{T}}, \mu_{\mathcal{T}})$ specifies the probability values for sets of sampling traces.

Posterior Distributions. Given a probabilistic program P, an initial program valuation $\mathbf{v} \in Val_{V_\mathrm{p}}$ and a measurable set $U \in \Sigma_{\mathbb{R}}$, we define the set of terminating traces where the value of the return variable falls into U as

$$\mathcal{T}_{P,\mathbf{v},U} := \{\boldsymbol{t} \in \mathcal{T} \mid \langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle, \mathbf{v}'[ret] \in U\}$$

and the set of all terminating traces as

$$\mathcal{T}_{P,\mathbf{v}} := \{\boldsymbol{t} \in \mathcal{T} \mid \langle \mathbf{v}, P, 1, \boldsymbol{t}\rangle \to^* \langle \mathbf{v}', \mathrm{skip}, w, []\rangle\}.$$

Note that $\mathcal{T}_{P,\mathbf{v}} = \mathcal{T}_{P,\mathbf{v},\mathbb{R}}$. Therefore, we can define the *unnormalised density* w.r.t $P, \mathbf{v}, U$ as

$$[\![P]\!]_{\mathbf{v}}(U) := \int_{\mathcal{T}_{P,\mathbf{v},U}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}). \tag{4}$$

That is, the integral takes all traces $\boldsymbol{t}$ on which P starts from $\mathbf{v}$ and evaluates to a value in U, weighting each $\boldsymbol{t}$ with the weight $\mathsf{wt}_P(\mathbf{v}, \boldsymbol{t})$ of the corresponding execution. The *normalising constant* is thus defined by

$$Z_{P,\mathbf{v}} := \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}). \tag{5}$$

Therefore, the normalised *posterior distribution* is defined as $\mathsf{posterior}_P(\mathbf{v}, U) := \frac{[\![P]\!]_{\mathbf{v}}(U)}{Z_{P,\mathbf{v}}}$.

We call a program P *integrable* if its normalised constant is finite, i.e, $0 < Z_{P,\mathbf{v}} < \infty$ for any $\mathbf{v} \in Val_{V_\mathrm{p}}$. Given an integrable program, we are interested in deriving lower and upper bounds on the posterior distribution.

Definition 6 (Interval Bounds). *Given an integrable probabilistic program P, a program valuation $\mathbf{v} \in Val_{V_\mathrm{p}}$, and a measurable set $U \in \Sigma_{\mathbb{R}}$, we call $[l, u]$ an* interval bound *of $\mathsf{posterior}_P(\mathbf{v}, U)$ if $l \le \mathsf{posterior}_P(\mathbf{v}, U) \le u$ for two reals $0 \le l \le u \le 1$.*

Equivalence between Posterior Distributions and Expected Weights. [Please summarize the main results of the next subsection here, and put the next subsection into the appendix.]

Below we will introduce the equivalence between posterior distributions and expected weights.

Lemma 1. *For all non-negative bounded measurable function $g : \mathbb{R}^{|V_p|} \to \mathbb{R}$, and a program state $\Xi = (\ell, \mathbf{v})$, and a statement $P := S_1; S_2$, we have that*

$$\begin{aligned}
&\int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{S_1,\mathbf{v}}} \mathsf{wt}_{S_1}(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \Big([\mathsf{val}_{S_1}(\mathbf{v}, \boldsymbol{t}) = \mathbf{v}'] \\
&\quad \cdot \int_{\mathcal{T}_{S_2,\mathbf{v}'}} \mathsf{wt}_{S_2}(\mathbf{v}', \boldsymbol{t}') \cdot g(\mathsf{val}_{S_2}(\mathbf{v}', \boldsymbol{t}'))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}') \Big)
\end{aligned}$$

Proposition 1. *For all non-negative bounded measurable function $g : \mathbb{R}^{|V_p|} \to \mathbb{R}$, a probabilistic program P and an initial program state $\Xi = (\ell, \mathbf{v}) \in \Lambda$, we have that*

$$\mathbb{E}_{\Xi}\left[W_\infty \cdot g(O_\infty)\right] = \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}).$$

See the proof in Appendix A.

Theorem 6. *Given a probabilistic program P, an initial program state $\Xi \in \Lambda$ and a measurable set $U \in \Sigma_{\mathbb{R}}$, it holds that $\mathbb{E}_{\Xi}\left[W_\infty \cdot [O_\infty \in U]\right] = [\![P]\!]_{\mathbf{v}}(U)$. Moreover, the expected weight $\mathbb{E}_{\Xi}\left[W_\infty\right]$ is equivalent to the normalising constant Z_P.*

Proof. We instantiate Proposition 1 with $g(x) = [x \in U]$. For any initial program state $\Xi = (\ell, \mathbf{v})$, we have that

$$\begin{aligned}
\mathbb{E}_{\Xi}\left[W_\infty \cdot [\mathrm{O}_\infty \in U]\right] &= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot [\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}) \in U]\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{P,\mathbf{v},U}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= [\![P]\!]_{\mathbf{v}}(U)
\end{aligned}$$

We prove $\mathbb{E}_{\Xi}\left[W_\infty\right] = Z_P$ by setting $U = \mathbb{R}$.

By Theorem 6, we show the equivalence between posterior distributions under sampling-based semanctis and expected weights under transition-based semanctis. In the following, we will focus on developing approahces to infer interval bounds on expected weights.

A.4 Connections between Two Semantics

Lemma 1 For all non-negative bounded measurable function $g : \mathbb{R}^{|V_{\mathrm{p}}|} \to \mathbb{R}$, and a program state $\Xi = (\ell, \mathbf{v})$, and a statement $P := S_1; S_2$, we have that

$$\begin{aligned}
&\int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{S_1,\mathbf{v}}} \mathsf{wt}_{S_1}(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \Big([\mathsf{val}_{S_1}(\mathbf{v}, \boldsymbol{t}) = \mathbf{v}'] \\
&\quad \cdot \int_{\mathcal{T}_{S_2,\mathbf{v}'}} \mathsf{wt}_{S_2}(\mathbf{v}', \boldsymbol{t}') \cdot g(\mathsf{val}_{S_2}(\mathbf{v}', \boldsymbol{t}'))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}') \Big)
\end{aligned}$$

Proof.

Proposition 1 For all non-negative bounded measurable function $g : \mathbb{R}^{|V_{\mathrm{p}}|} \to \mathbb{R}$, a probabilistic program P and an initial program state $\Xi = (\ell, \mathbf{v}) \in \Lambda$, we have that

$$\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] = \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}).$$

the order in appendix A need to be readjusted and the opinion is mentioned here: $\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \to \mathrm{d}\mu_{\mathcal{T}}(\boldsymbol{t})$, $\mathbb{P}_{\Xi}(\mathrm{d}\omega) \to \mathrm{d}\mathbb{P}_{\Xi}$.

Proof. We prove by induction on the structure of statements.

– Case $P \equiv$ "**skip**".

$$\begin{aligned}
\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty(\omega)) \mathbb{P}_{\Xi}(\mathrm{d}\omega) \\
&= g(\mathbf{v}) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} [\boldsymbol{t} = []] \cdot g(\mathbf{v})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})
\end{aligned}$$

– Case $P \equiv$ "$x := E$".

$$\begin{aligned}
\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty(\omega)) \mathbb{P}_{\Xi}(\mathrm{d}\omega) \\
&= g(\mathbf{v}[x \mapsto [\![E]\!](\mathbf{v})]) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} [\boldsymbol{t} = []] \cdot g(\mathbf{v}[x \mapsto [\![E]\!](\mathbf{v})])\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})
\end{aligned}$$

– Case $P \equiv$ "$x :=$ **sample** D".

$$\begin{aligned}\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty(\omega))\mathbb{P}_\Xi(\mathrm{d}\omega)\\ &= \int g(\mathbf{v}[x \mapsto r])\mu_D(\mathrm{d}r)\\ &= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\end{aligned}$$

– Case $P \equiv$ "**observe**(x, D)".

$$\begin{aligned}\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty(\omega))\mathbb{P}_\Xi(\mathrm{d}\omega)\\ &= W_\ell(\mathbf{v}) \cdot g(\mathbf{v})\\ &= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\end{aligned}$$

– Case $P \equiv$ "**return** x".

$$\begin{aligned}\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty(\omega))\mathbb{P}_\Xi(\mathrm{d}\omega)\\ &= g(\mathbf{v})\\ &= \int_{\mathcal{T}_{P,\mathbf{v}}} [\boldsymbol{t} = []] \cdot g(\mathbf{v})\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\\ &= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\end{aligned}$$

– Case $P \equiv$ "**if** B **then** S_1 **else** S_2 **fi**". Assume the next state corresponding to the **then**-branch (resp. **else**-branch) is $\Xi_1 = (\ell_1, \mathbf{v})$ (resp. $\Xi_2 = (\ell_2, \mathbf{v})$). Then we obtain that

$$\begin{aligned}\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= [[\![B]\!](\mathbf{v}) = \mathit{true}] \cdot \mathbb{E}_{\Xi_1}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] + [[\![B]\!](\mathbf{v}) = \mathit{false}] \cdot \mathbb{E}_{\Xi_2}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right]\\ &= [[\![B]\!](\mathbf{v}) = \mathit{true}] \cdot \int_{\mathcal{T}_{S_1,\mathbf{v}}} \mathsf{wt}_{S_1}(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_{S_1}(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\\ &\quad + [[\![B]\!](\mathbf{v}) = \mathit{false}] \cdot \int_{\mathcal{T}_{S_2,\mathbf{v}}} \mathsf{wt}_{S_2}(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_{S_2}(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\\ &= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v},\boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v},\boldsymbol{t}))\,\mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})\end{aligned}$$

– Case $P \equiv$ "$S_1; S_2$".

$$\begin{aligned}
\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= \int W_\infty(\omega)\mathbb{P}_{\Xi}(\mathrm{d}\omega) \int [\omega_T = \Xi'] \cdot W_\infty(\omega') \cdot g(\mathrm{O}_\infty(\omega'))\mathbb{P}_{\Xi'}(\mathrm{d}\omega') \\
&= \int W_\infty(\omega)\mathbb{P}_{\Xi}(\mathrm{d}\omega) \left([\omega_T = \Xi'] \cdot \int \cdot W_\infty(\omega') \cdot g(\mathrm{O}_\infty(\omega'))\mathbb{P}_{\Xi'}(\mathrm{d}\omega') \right) \\
&= \int W_\infty(\omega)\mathbb{P}_{\Xi}(\mathrm{d}\omega) \left([\omega_T = \Xi'] \cdot \int_{\mathcal{T}_{S_2,\mathbf{v}'}} \mathsf{wt}_{S_2}(\mathbf{v}', \boldsymbol{t}') \cdot g(\mathsf{val}_{S_2}(\mathbf{v}', \boldsymbol{t}'))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}') \right) \\
&= \int_{\mathcal{T}_{S_1,\mathbf{v}}} \mathsf{wt}_{S_1}(\mathbf{v}, \boldsymbol{t})\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \Big([\mathsf{val}_{S_1}(\mathbf{v}, \boldsymbol{t}) = \mathbf{v}'] \\
&\quad \cdot \int_{\mathcal{T}_{S_2,\mathbf{v}'}} \mathsf{wt}_{S_2}(\mathbf{v}', \boldsymbol{t}') \cdot g(\mathsf{val}_{S_2}(\mathbf{v}', \boldsymbol{t}'))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}') \Big) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})
\end{aligned}$$

Here ω_i is the i-th element of the sequence $\omega = \{\Xi_n\}_{n\in\mathbb{N}}$, i.e., $\omega_i := \Xi_i$. ω_T is the last element of ω, and $\Xi' = (\ell', \mathbf{v}')$. The third and fourth equalities follow from the induction hypothesis, and the last equality from Lemma 1.

– Case $P \equiv$ "**while** B **do** S **od**". Assume the next state corresponding to the entry of the loop (resp. the exit of the loop) is $\Xi_1 = (\ell_1, \mathbf{v})$ (resp. $\Xi_2 = (\ell_2, \mathbf{v})$). Then we obtain that

$$\begin{aligned}
\mathbb{E}_{\Xi}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] &= [[\![B]\!](\mathbf{v}) = \mathit{true}] \cdot \mathbb{E}_{\Xi_1}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] + [[\![B]\!](\mathbf{v}) = \mathit{false}] \cdot \mathbb{E}_{\Xi_2}\left[W_\infty \cdot g(\mathrm{O}_\infty)\right] \\
&= [[\![B]\!](\mathbf{v}) = \mathit{true}] \cdot \int W_\infty(\omega) \cdot g(\mathrm{O}_\infty)\mathbb{P}_{\Xi_1}(\mathrm{d}\omega) + [[\![B]\!](\mathbf{v}) = \mathit{false}] \cdot g(\mathbf{v}) \\
&= [[\![B]\!](\mathbf{v}) = \mathit{true}] \cdot \int_{\mathcal{T}_{S;P,\mathbf{v}}} \mathsf{wt}_{S;P}(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_{S;P}(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&\quad + [[\![B]\!](\mathbf{v}) = \mathit{false}] \cdot \int_{\mathcal{T}_{\mathbf{skip},\mathbf{v}}} [\boldsymbol{t} = []] \cdot g(\mathsf{val}_{\mathbf{skip}}(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathcal{T}_{P,\mathbf{v}}} \mathsf{wt}_P(\mathbf{v}, \boldsymbol{t}) \cdot g(\mathsf{val}_P(\mathbf{v}, \boldsymbol{t}))\, \mu_{\mathcal{T}}(\mathrm{d}\boldsymbol{t})
\end{aligned}$$

B Supplementary Material for Section 4

B.1 Basics of Fixed Point Theory

Given a partial order $\sqsubseteq$ on a set K and a subset $K' \subseteq K$, an *upper bound* of K' is an element $u \in K$ that is no smaller than every element of K', i.e., $\forall k' \in K'.\ k' \sqsubseteq u$. Similarly, a *lower bound* for K' is an element l that is no greater than every element of K', i.e. $\forall k' \in K'.\ l \sqsubseteq k'$. The *supremum* of K', denoted by $\bigsqcup K'$, is an element $u^* \in K$ such that u^* is an upper-bound of K' and for every upper bound u of K', we have $u^* \sqsubseteq u$. Similarly, the *infimum* $\bigsqcap K'$ is a

lower bound l^* of K' such that for every lower-bound l of K', we have $l \sqsubseteq l^*$. We define $\bot := \bigsqcap K$ and $\top := \bigsqcup K$. In general, suprema and infima may not exist.

A partial order $(K, \sqsubseteq)$ is called a *complete lattice* if every subset $K' \subseteq K$ has a supremum and an infimum. Given a partial order $(K, \sqsubseteq)$, a function $f : K \to K$ is called *monotone* if for every $k_1 \sqsubseteq k_2$ in K, we have $f(k_1) \sqsubseteq f(k_2)$.

Given a complete lattice $(K, \sqsubseteq)$, a function $f : K \to K$ is called *continuous* if for every increasing chain $k_0 \sqsubseteq k_1 \sqsubseteq \ldots$ in K, we have $f(\bigsqcup\{k_n\}_{n=0}^{\infty}) = \bigsqcup\{f(k_n)\}_{n=0}^{\infty}$, and *cocontinuous* if for every decreasing chain $k_0 \sqsupseteq k_1 \sqsupseteq \ldots$ of elements of K, we have $f(\bigsqcap\{k_n\}_{n=0}^{\infty}) = \bigsqcap\{f(k_n)\}_{n=0}^{\infty}$.

Given a complete lattice $(K, \sqsubseteq)$ and a function $f : K \to K$, an element $k \in K$ is called a *fixed-point* if $f(k) = k$. Moreover, k is a *pre fixed-point* if $f(k) \sqsubseteq k$ and a *post fixed-point* if $k \sqsubseteq f(k)$. The *least fixed-point* of f, denoted by $\mathrm{lfp} f$, is the fixed-point that is no greater than every fixed-point under $\sqsubseteq$. Analogously, the *greatest fixed-point* of f, denoted by $\mathrm{gfp} f$, is the fixed-point that is no smaller than all fixed-points.

Theorem 7 (*Kleene* [41]). *Let $(K, \sqsubseteq)$ be a complete lattice and $f : K \to K$ be an continuous function. Then, we have*

$$\mathrm{lfp}\ f = \bigsqcup_{i \geq 0} \left\{ f^{(i)}(\bot) \right\}.$$

Analogously, if f is cocontinuous, then we have

$$\mathrm{gfp}\ f = \bigsqcap_{i \geq 0} \left\{ f^{(i)}(\top) \right\}.$$

B.2 Proofs for Our Fixed-Point Approach

Theorem 2. The expected-weight function *ew* is the least fixed point of the expected-weight transformer *ewt*.

Proof. Define the step-bounded weight random variable $W^n_{(\ell,\mathbf{v})}$ starting from any program state $\Xi = (\ell, \mathbf{v})$ for a step bound $n \in \mathbb{N}$ by

$$W^n_{(\ell,\mathbf{v})}(\omega) = \begin{cases} W_{(\ell,\mathbf{v})}(\omega) & \text{if } T(\omega) \leq n \\ 0 & \text{otherwise} \end{cases}.$$

Since we always assume that the underlying WPTS is almost-surely terminating, it follows that the sequence of random variables $\{W^n\}_{n\in\mathbb{N}}$ converges non-decreasingly to W.

Given any program state $\Xi = (\ell, \mathbf{v})$ with a unique transition $\tau = \langle \ell, \phi_\tau, f_\tau \rangle$ satisfying $\mathbf{v} \models \phi_\tau$, define the step-bounded expected-weight function ew^n by $ew^n(\ell, \mathbf{v}) = \mathbb{E}_{(\ell,\mathbf{v})}[W]^n_{(\ell,\mathbf{v})}$. Without loss of generality, we assume there is only one fork f_τ in this transition. Assume the next sampling valuation from Ξ is $\mathbf{r}_0$ and the next program state is $\Xi' = (\ell', \mathbf{v}')$, i.e., $\mathbf{v}' = f_\tau(\mathbf{v}, \mathbf{r}_0)$. Following the symbols in Section 5, we denote the probability space of the WPTS Π starting

from $(\ell, \mathbf{v})$, i.e., the program runs starting from $(\ell, \mathbf{v})$ as $(\Omega, \mathcal{F}, \mathbb{P})_{\Xi}$. By Tonelli-Fubini Theorem, we have that for all $n \geq 0$,

$$\begin{aligned}
ew^{n+1}(\ell, \mathbf{v}) &= \int W^{n+1}_{(\ell,\mathbf{v})} \, \mathrm{d}\mathbb{P}_{\Xi} \\
&= \int W^{n+1}_{(\ell,\mathbf{v})} \, \mathrm{d}(\mathcal{D}_{\mathbf{r}_0} \times \mathbb{P}_{\Xi'}) \\
&= \int W(\ell, \mathbf{v}) \cdot W^{n}_{(\ell',\mathbf{v}')}(\omega) \, \mathrm{d}(\mathcal{D}_{\mathbf{r}_0} \times \mathbb{P}_{\Xi'}) \\
&= \int_{\mathbf{r}_0} \int_{\omega_{\Xi'}} W(\ell, \mathbf{v}) \cdot W^{n}_{(\ell',\mathbf{v}')}(\omega) \, \mathrm{d}\mathbb{P}_{\Xi'} \, \mathrm{d}\mathcal{D}_{\mathbf{r}_0} \\
&= \int_{\mathbf{r}_0} W(\ell, \mathbf{v}) \cdot \left(\int_{\omega_{\Xi'}} W^{n}_{(\ell',\mathbf{v}')}(\omega) \, \mathrm{d}\mathbb{P}_{\Xi'} \right) \mathrm{d}\mathcal{D}_{\mathbf{r}_0} \\
&= \int_{\mathbf{r}_0} W(\ell, \mathbf{v}) \cdot ew^{n}(\ell', \mathbf{v}') \, \mathrm{d}\mathcal{D}_{\mathbf{r}_0} \\
&= \mathbb{E}_{\mathbf{r}_0} \left[W(\ell, \mathbf{v}) \cdot ew^{n}(\ell', \mathbf{v}') \right] \\
&= ewt(ew^n)(\ell, \mathbf{v})
\end{aligned}$$

By applying MCT to the both sides of the equality above, we have that

$$ew(\ell, \mathbf{v}) = ewt(ew)(\ell, \mathbf{v}).$$

This shows that ew is a fixed point of ewt. Furthermore, given any fixed point h of ewt, since (i) $ew^0 \leq h$ and (ii) $ew^n \leq h$ implies $ew^{n+1} = ewt(ew^n) \leq ewt(h) = h$, one can prove by a straightforward induction on n that $ew^n \leq h$ for all $n \geq 0$. It follows from $ew = \lim_{n\to\infty} ew^n$ that ew is the least fixed point of ewt.

In order to show the uniqueness of the fixed point, we first prove that ewt is both continuous and cocontinuous.

Proposition 2. *If $M \in [0, \infty)$, then the expected-weight transformer $ewt : \mathcal{K}_M \to \mathcal{K}_M$ is both continuous and cocontinous.*

Proof. We first prove that ewt is well-defined. Given an arbitrary $h \in \mathcal{K}_M$, for any $\Xi = (\ell, \mathbf{v}) \in \Lambda$,

- When $\ell = \ell_{\text{out}}$, $ewt(h)(\ell, \mathbf{v}) = 1$.
- When $\ell \neq \ell_{\text{out}}$, for a unique transition $\tau = \langle \ell, \phi_\tau, f_\tau, \ell' \rangle$ such that $\mathbf{v} \models \phi_\tau$,

$$\begin{aligned}
ewt(h)(\ell, \mathbf{v}) &= \mathbb{E}_{\mathbf{r}} \left[h(\ell', f_\tau(\mathbf{v}, \mathbf{r})) \cdot W(\ell, \mathbf{v}) \right] \\
&\leq M \cdot maxscore \\
&< \infty
\end{aligned}$$

where $maxscore$ is the maximum of W given any state Ξ. As W is a nonnegative function, we can prove that $ewt(h)(\ell, \mathbf{v}) \geq 0$. Thus, ewt is well defined. Next, we prove that ewt is monotone. Given any two functions $h_1, h_2 \in \mathcal{K}_M$ such that $h_1 \leq h_2$, by case analysis on $(\ell, \mathbf{v})$,

- If $\ell = \ell_{\text{out}}$, $ewt(h_1)(\ell, \mathbf{v}) = 1 = ewt(h_2)(\ell, \mathbf{v})$.
- If $\ell \neq \ell_{\text{out}}$, given a unique transition $\tau = \langle \ell, \phi_\tau, f_\tau, \ell' \rangle$ such that $\mathbf{v} \models \phi_\tau$,

$$\begin{aligned} ewt(h_1)(\ell, \mathbf{v}) &= \mathbb{E}_{\mathbf{r}}\left[h_1(\ell', f_\tau(\mathbf{v}, \mathbf{r})) \cdot W(\ell, \mathbf{v})\right] \\ &\leq \mathbb{E}_{\mathbf{r}}\left[h_2(\ell', f_\tau(\mathbf{v}, \mathbf{r})) \cdot W(\ell, \mathbf{v})\right] \\ &= ewt(h_2)(\ell, \mathbf{v}) \end{aligned}$$

Therefore, $ewt(h_1) \leq ewt(h_2)$, hence it is monotone. Then we prove upper continuity of ewt. Choose any increasing chain $h_0 \sqsubseteq h_1 \sqsubseteq h_2 \sqsubseteq \cdots$ and do another case analysis on $(\ell, \mathbf{v})$:

- If $\ell = \ell_{\text{out}}$, then

$$ewt(\bigsqcup_{n\geq 0} \{h_n\})(\ell, \mathbf{v}) = 1 = \bigsqcup_{n\geq 0} \{ewt(h_n)\}(\ell, \mathbf{v}).$$

- Otherwise, for a unique transition $\tau = \langle \ell, \phi_\tau, f_\tau \rangle$ such that $\mathbf{v} \models \phi_\tau$(whether to add the term $wt_j(\mathbf{v}, \mathbf{r})$):

$$\begin{aligned} & ewt(\bigsqcup_{n\geq 0} \{h_n\})(\ell, \mathbf{v}) \\ =& \mathbb{E}_{\mathbf{r}}\left[wt_j(\mathbf{v}, \mathbf{r}) \cdot (\bigsqcup_{n\geq 0} \{h_n\})(\ell', f_\tau(\mathbf{v}, \mathbf{r}))\right] \\ =& \mathbb{E}_{\mathbf{r}}\left[\sup_{n\geq 0} \{h_n(\ell', f_\tau(\mathbf{v}, \mathbf{r}))\}\right] \\ =& \mathbb{E}_{\mathbf{r}}\left[\lim_{n\to\infty} \{h_n(\ell', f_\tau(\mathbf{v}, \mathbf{r}))\}\right] \\ \overset{\text{MCT}}{=}& \lim_{n\to\infty} \mathbb{E}_{\mathbf{r}}\left[h_n(\ell', f_\tau(\mathbf{v}, \mathbf{r}))\right] \\ =& \lim_{n\to\infty} ewt(h_n)(\ell, \mathbf{v}) \\ =& \sup_{n\geq 0} \{ewt(h_n)(\ell, \mathbf{v})\} \\ =& \bigsqcup_{n\geq 0} \{ewt(h_n)\}(\ell, \mathbf{v}) \end{aligned}$$

The "MCT" above denotes the monotone convergence theorem. A similar argument establishes cocontinuity for integrable h_0 and decreasing chains.

Then the uniqueness follows from Theorem 7.

Theorem 2. Let Π be a non-score-recursive WPTS whose weights are bounded in $[-M, M]$ for a finite $M \geq 1$. Then the expected-weight function ew is the least fixed point of the expected-weight transformer ewt in the complete lattice $(\mathcal{K}_M, \leq)$. Furthermore, if the WPTS Π is AST, then the function ew is the unique fixed point of the higher-order function ewt in $(\mathcal{K}_M, \leq)$ when $M \geq 1$.

Proof. The proof follows similar arguments in [48, Theorem 4.4]. By Proposition 2, we have that for every state $\Xi = (\ell, \mathbf{v})$,

– lfp $ewt(\ell, \mathbf{v}) = \lim_{n\to\infty} ewt^n(\bot)(\ell, \mathbf{v})$, and
– gfp $ewt(\ell, \mathbf{v}) = \lim_{n\to\infty} ewt^n(\top)(\ell, \mathbf{v})$.

By the definition of ewt^n_M and Proposition 2, we have that

– $ewt^n(\bot)(\ell, \mathbf{v}) = \mathbb{E}_\Xi [W \cdot [T \leq n]] - M \cdot \mathbb{P}(T > n)$, and
– $ewt^n(\top)(\ell, \mathbf{v}) = \mathbb{E}_\Xi [W \cdot [T \leq n]] + M \cdot \mathbb{P}(T > n)$.

Recall that we assume the underlying PTS to be almost-surely terminating. Hence, $\lim_{n\to\infty} \mathbb{P}(T > n) = \mathbb{P}(T = \infty) = 0$. It follows that lfp $ewt(\ell, \mathbf{v}) =$ gfp $ewt(\ell, \mathbf{v})$, i.e., the fixed point is unique.

B.3 Proof for the OST Variant

Theorem 3 (The OST Variant) Let $\{X_n\}_{n=0}^\infty$ be a martingale (resp. supermartingale) adapted to a filtration $\{\mathcal{F}_n\}_{n=0}^\infty$, and T be a stopping time w.r.t. $\{\mathcal{F}_n\}_{n=0}^\infty$. Then the following condition is sufficient to ensure that $\mathbb{E}(|X_T|) < \infty$ and $\mathbb{E}(X_T) = \mathbb{E}(X_0)$ (resp. $\mathbb{E}(X_T) \leq \mathbb{E}(X_0)$):

– There exist real numbers $\lambda, c_1, c_2, c_3 > 0$ and $c_3 < c_2$ such that (i) for sufficiently large $n \in \mathbb{N}$, it holds that $\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$; (ii) for all $n \in \mathbb{N}$, $|X_{n+1} - X_n| \leq \lambda \cdot n^d \cdot e^{c_3 \cdot n}$ almost surely.

Proof. We only prove the "$\leq$" case, the "$=$" case is similar. For every $n \in \mathbb{N}_0$,

$$\begin{aligned}
|X_{T\wedge n}| &= \left| X_0 + \sum_{k=0}^{T\wedge n-1} (X_{k+1} - X_k) \right| \\
&= \left| X_0 + \sum_{k=0}^{\infty} (X_{k+1} - X_k) \cdot \mathbf{1}_{T>k\wedge n>k} \right| \\
&\leq |X_0| + \sum_{k=0}^{\infty} |(X_{k+1} - X_k) \cdot \mathbf{1}_{T>k\wedge n>k}| \\
&\leq |X_0| + \sum_{k=0}^{\infty} |(X_{k+1} - X_k) \cdot \mathbf{1}_{T>k}| \quad .
\end{aligned}$$

Then

$$
\begin{aligned}
&\mathbb{E}\left(|X_0| + \sum_{k=0}^{\infty} |(X_{k+1} - X_k) \cdot \mathbf{1}_{T>k}|\right) \\
&= \text{(By Monotone Convergence Theorem)} \\
&\quad \mathbb{E}\left(|X_0|\right) + \sum_{k=0}^{\infty} \mathbb{E}\left(|(X_{k+1} - X_k) \cdot \mathbf{1}_{T>k}|\right) \\
&= \mathbb{E}\left(|X_0|\right) + \sum_{k=0}^{\infty} \mathbb{E}\left(|X_{k+1} - X_k| \cdot \mathbf{1}_{T>k}\right) \\
&\leq \mathbb{E}\left(|X_0|\right) + \sum_{k=0}^{\infty} \mathbb{E}\left(\lambda \cdot k^d \cdot e^{c_3 \cdot k} \cdot \mathbf{1}_{T>k}\right) \\
&= \mathbb{E}\left(|X_0|\right) + \sum_{k=0}^{\infty} \lambda \cdot k^d \cdot e^{c_3 \cdot k} \cdot \mathbb{P}\left(T > k\right) \\
&\leq \mathbb{E}\left(|X_0|\right) + \sum_{k=0}^{\infty} \lambda \cdot k^d \cdot e^{c_3 \cdot k} \cdot c_1 \cdot e^{-c_2 \cdot k} \\
&= \mathbb{E}\left(|X_0|\right) + \lambda \cdot c_1 \cdot \sum_{k=0}^{\infty} k^d \cdot e^{-(c_2 - c_3) \cdot k} \\
&< \infty \ .
\end{aligned}
$$

where the first inequality is obtained by Condition (ii), and the second inequality is derived from Condition (i).

Therefore, by Dominated Convergence Theorem and the fact that $X_T = \lim_{n\to\infty} X_{T\wedge n}$ a.s.,

$$
\mathbb{E}\left(X_T\right) = \mathbb{E}\left(\lim_{n\to\infty} X_{T\wedge n}\right) = \lim_{n\to\infty} \mathbb{E}\left(X_{T\wedge n}\right) \ .
$$

Finally, the result follows from properties for the stopped process $\{X_{T\wedge n}\}_{n\in\mathbb{N}_0}$ that

$$
\mathbb{E}\left(X_T\right) \leq \mathbb{E}\left(X_0\right) \ .
$$

B.4 Proofs for Our OST-Based Approach

Theorem 4. [Upper Bounds on Expected Weights] Consider a WPTS Π with a d-degree polynomial PUWF h and an affine invariant I, if Π has (1) the concentration property, i.e., $\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ for $c_1, c_2 > 0$, (2) the bounded-update property, and (3) the weight of each step is bounded by e^{c_3} for $0 < c_3 < c_2$, then $\mathbb{E}_{\mathbf{v}_{\text{init}}}\left[w_T\right] \leq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$ for any initial state $(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$.

Proof. Define the stochastic process $\{X_n\}_{n=0}^{\infty}$ as $X_n := h(\ell_n, \mathbf{v}_n)$ where $(\ell_n, \mathbf{v}_n)$ is the program state at the $n-$th step of a program run. Then construct a stochastic process $\{Y_n\}_{n=0}^{\infty}$ such that $Y_n := X_n \cdot \prod_{i=0}^{n-1} W_i$ where W_i is the

weight at the $i-$th step of the program run. According to Condition (C1), we have that $\mathbb{E}\left[X_{n+1} \cdot W_n | \mathcal{F}_n\right] \leq X_n$. Therefore, by the "take out what is known" property of conditional expectation (see [52]), it follows that

$$\begin{aligned} &\mathbb{E}\left[X_{n+1} \cdot \prod_{i=0}^{n} W_i | \mathcal{F}_n\right] \leq X_n \cdot \prod_{i=0}^{n-1} W_i \\ \Leftrightarrow\ &\mathbb{E}\left[Y_{n+1} | \mathcal{F}_n\right] \leq \mathbb{E}\left[Y_n\right], \end{aligned}$$

which means that $\mathbb{E}\left[Y_{n+1}\right] \leq \mathbb{E}\left[Y_n\right]$ from the basic property of conditional expectation. By an easy induction on n, we have that $\mathbb{E}\left[Y_n\right] \leq \mathbb{E}\left[Y_0\right] < \infty$ for all $n \geq 0$, thus the conditional expectation is also taken in the normal sense as each Y_n is indeed integrable. Hence, $\{Y_n\}_{n=0}^{\infty}$ is a supermartingale. Moreover, we have from the bounded-update property that $|X_{n+1}| \leq \zeta \cdot (n+1)^d$ for a real number $\zeta > 0$. By definition, we obtain that for sufficiently large n,

$$\begin{aligned} |Y_{n+1} - Y_n| &= \left| X_{n+1} \cdot \prod_{i=0}^{n} W_i - X_n \cdot \prod_{i=0}^{n-1} W_i \right| \\ &\leq \left| X_{n+1} \cdot \prod_{i=0}^{n} W_i \right| + \left| X_n \cdot \prod_{i=0}^{n-1} W_i \right| \\ &< e^{c_3 \cdot n} \cdot (|X_{n+1}| + |X_n|) \\ &\leq e^{c_3 \cdot n} \cdot [\zeta \cdot (n+1)^d + \zeta \cdot n^d] \\ &\leq \lambda \cdot n^d \cdot e^{c_3 \cdot n} \end{aligned}$$

where the first inequality is induced by the triangle inequality, and the second inequality is derived from the bounded stepwise weight condition such that each $W_i \in [0, e^{c_3}]$ and the fact $W_0 = 1$. By applying the OST variant (Theorem 3), we obtain that $\mathbb{E}\left[Y_T\right] \leq \mathbb{E}\left[Y_0\right]$. By definition and Condition (C2),

$$\begin{aligned} Y_T &= h(\ell_T, \mathbf{v}_T) \cdot \prod_{i=0}^{T-1} W_i \\ &= h(\ell_{\text{out}}, \mathbf{v}_T) \cdot \prod_{i=0}^{T-1} W_i \\ &= \prod_{i=0}^{T-1} W_i \end{aligned}$$

Finally, we have that $\mathbb{E}_{\Xi}\left[W_\infty\right] = \mathbb{E}\left[\prod_{i=0}^{T-1} W_i\right] \leq \mathbb{E}\left[Y_0\right] = h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$.

Theorem 4. [Lower Bounds on Expected Weights] Consider a WPTS Π with a d-degree polynomial PLWF h and an affine invariant I, if Π has (1) the concentration property, i.e., $\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ for $c_1, c_2 > 0$, (2) the bounded-update property, and (3) the weight of each step is bounded by e^{c_3} for $0 < c_3 < c_2$, then $\mathbb{E}_{\mathbf{v}_{\text{init}}}\left[w_\infty\right] \geq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$ for any initial state $(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$.

Proof. Define the stochastic process $\{X_n\}_{n=0}^{\infty}$ as $X_n := h(\ell_n, \mathbf{v}_n)$ where $(\ell_n, \mathbf{v}_n)$ is the program state at the $n-$th step of a program run. Then construct a stochastic process $\{-Y_n\}_{n=0}^{\infty}$ such that $-Y_n := -X_n \cdot \prod_{i=0}^{n-1} W_i$ where W_i is the weight at the $i-$th step of the program run. According to Condition (C1'), we have that $\mathbb{E}\left[-X_{n+1} \cdot W_n | \mathcal{F}_n\right] \leq -X_n$. Therefore, by the "take out what is known" property of conditional expectation (see [52]), it follows that

$$\begin{aligned} &\mathbb{E}\left[-X_{n+1} \cdot \prod_{i=0}^{n} W_i | \mathcal{F}_n\right] \leq -X_n \cdot \prod_{i=0}^{n-1} W_i \\ \Leftrightarrow\ &\mathbb{E}\left[-Y_{n+1} | \mathcal{F}_n\right] \leq \mathbb{E}\left[-Y_n\right], \end{aligned}$$

which means that $\mathbb{E}\left[-Y_{n+1}\right] \leq \mathbb{E}\left[-Y_n\right]$ from the basic property of conditional expectation. By an easy induction on n, we have that $\mathbb{E}\left[-Y_n\right] \leq \mathbb{E}\left[-Y_0\right] < \infty$ for all $n \geq 0$, thus the conditional expectation is also taken in the normal sense as each Y_n is indeed integrable. Hence, $\{-Y_n\}_{n=0}^{\infty}$ is a supermartingale. Moreover, we have from the bounded-update property that $|X_{n+1}| \leq \zeta \cdot (n+1)^d$ for a real number $\zeta > 0$. By definition, we obtain that for sufficiently large n,

$$\begin{aligned} |-Y_{n+1} - (-Y_n)| &= \left| X_{n+1} \cdot \prod_{i=0}^{n} W_i - X_n \cdot \prod_{i=0}^{n-1} W_i \right| \\ &\leq \left| X_{n+1} \cdot \prod_{i=0}^{n} W_i \right| + \left| X_n \cdot \prod_{i=0}^{n-1} W_i \right| \\ &< e^{c_3 \cdot n} \cdot (|X_{n+1}| + |X_n|) \\ &\leq e^{c_3 \cdot n} \cdot [\zeta \cdot (n+1)^d + \zeta \cdot n^d] \\ &\leq \lambda \cdot n^d \cdot e^{c_3 \cdot n} \end{aligned}$$

where the first inequality is induced by the triangle inequality, and the second inequality is derived from the bounded stepwise weight condition such that each $W_i \in [0, e^{c_3}]$ and the fact $W_0 = 1$. By applying the variant of Optional Stopping Theorem (Theorem 3), we obtain that $\mathbb{E}\left[-Y_T\right] \leq \mathbb{E}\left[-Y_0\right]$, so $\mathbb{E}\left[Y_T\right] \geq \mathbb{E}\left[Y_0\right]$. By definition and Condition (C2'),

$$\begin{aligned} -Y_T &= -h(\ell_T, \mathbf{v}_T) \cdot \prod_{i=0}^{T-1} W_i \\ &= -h(\ell_{\text{out}}, \mathbf{v}_T) \cdot \prod_{i=0}^{T-1} W_i \\ &= -\prod_{i=0}^{T-1} W_i \end{aligned}$$

Finally, we have that $\mathbb{E}_{\Xi}\left[W_\infty\right] = \mathbb{E}\left[\prod_{i=0}^{T-1} W_i\right] \geq \mathbb{E}\left[Y_0\right] = h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$.

B.5 Correctness of Truncation

Theorem 5. Suppose that $(*)$ for each fork $F^{M,\sharp} = \langle \sharp, p, upd, M\rangle$ derived from some $F = \langle \ell', p, upd, wt\rangle$ in a transition with source location ℓ (see $(\ddagger)$) from the construction of $\Pi_{\mathcal{B},M}$, we have that $ewt(\ell', upd(\mathbf{v})) \leq M(\mathbf{v})$ for all $\mathbf{v}$ in some reachable state $(\ell, \mathbf{v})$. Then $ew_{\Pi}(\mathbf{v}) \leq ew_{\Pi_{\mathcal{B},M}}(\mathbf{v})$ for all initial program valuation $\mathbf{v}$. Analogously, if it holds the condition $(\star)$ which is almost the same as $(*)$ except for that "$ewt(\ell', upd(\mathbf{v})) \leq M(\mathbf{v})$" is replaced with "$ewt(\ell', upd(\mathbf{v})) \geq M(\mathbf{v})$", then we have $ew_{\Pi}(\mathbf{v}) \geq ew_{\Pi_{\mathcal{B},M}}(\mathbf{v})$ for all initial program valuation $\mathbf{v}$.

Proof. We first prove that when every score function $\mathcal{M}$ in a $F^{\mathcal{M},\sharp}$ derived from a transition with source location ℓ is equal to the function $ewt(\ell, -)$, we have that $\Pi_{\mathcal{B},\mathcal{M}}$ is equal to Π. By Theorem 2, the expected weight functions ew_{Π}, $ew_{\Pi_{\mathcal{B},\mathcal{M}}}$ are the least fixed point of the higher-order operator ewt defined in Definition 3. We prove that both $ew_{\Pi} \leq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$ and $ew_{\Pi} \geq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$ holds. Note that since we choose the scoring function to be the exact expected weight function of Π, it holds that $ew_{\Pi}(-, \mathbf{v}) = ew_{\Pi_{\mathcal{B},\mathcal{M}}}(-, \mathbf{v})$ for all program valuations outside B. Thus, the nontrivial part is to consider program valuations inside the truncated range.

- $ew_{\Pi} \leq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$: To show that $ew_{\Pi} \leq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$, it suffices to observe that $ew_{\Pi_{\mathcal{B},\mathcal{M}}}$ satisfies $ewt_{\Pi}(ew_{\Pi_{\mathcal{B},\mathcal{M}}}) = ew_{\Pi_{\mathcal{B},\mathcal{M}}}$. Since ew_{Π} is the least fixed point of the higher order equation, we directly obtain that $ew_{\Pi} \leq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$.
- $ew_{\Pi} \geq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$: To show that $ew_{\Pi} \geq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$, it suffices to observe that ew_{Π} (extended with the $\sharp$ location whose score function is 1) satisfies the higher-order equation of $\Pi_{\mathcal{B},\mathcal{M}}$. Thus, we directly have that $ew_{\Pi} \geq ew_{\Pi_{\mathcal{B},\mathcal{M}}}$.

Then we prove the theorem. We only prove the upper-bound case, since the lower-bound case can be proved similarly. The proof follows from Theorem 2. Denote $\bot$ as the bottom element of the complete lattice $(\mathcal{K}_M, \leq)$. Then by Theorem 2, we have that $\lim_{n\to\infty} ewt^n_{\Pi}(\bot) = ew_{\Pi}$ and $\lim_{n\to\infty} ewt^n_{\Pi_{\mathcal{B},\mathcal{M}}}(\bot) = ew_{\Pi_{\mathcal{B},\mathcal{M}}}$. Since $ew_{\Pi}(\mathbf{v}) \leq \mathcal{M}(\mathbf{v})$ for all $\mathbf{v} \in exit(\Pi)$, one can perform a straightforward induction on n that $ew^{n,ew_{\Pi}}_{\Pi_B} \leq ew^{n,f}_{\Pi_B}$ for all n. □

C Supplementary Material for Section 5

Theorem 8. *Let Π be a non-score-recursive WPTS with score functions $f_1, \ldots, f_k$ on the transitions to the termination location ℓ_{out}. Suppose we have a non-negative real number ϵ and polynomials $f'_1, \ldots, f'_k$ such that for all $x \in exit(\Pi)$ and $1 \leq j \leq k$, $|f'_j(x) - f_j(x)| \leq \epsilon$. Then we have that $|ew_{\Pi}(\mathbf{v}) - ew_{\Pi'}(\mathbf{v})| \leq \epsilon$ for all initial program valuation $\mathbf{v}$, where Π' is obtained from Π by replacing each f_j $(1 \leq j \leq k)$ with f'_j.*

Proof. By Theorem 2, we have that $\lim_{n\to\infty} ewt^n_{\Pi}(\bot) = ew_{\Pi}$ and $\lim_{n\to\infty} ewt^n_{\Pi'} = ew_{\Pi'}$. Since $|f'_j - f_j| \leq \epsilon$ for every j, one can perform a straightforward induction on n to prove that for all $n \geq 0$, it holds that that $|ewt^n_{\Pi}(\bot) - ewt^n_{\Pi'}(\bot)| \leq \epsilon$. □

C.1 Possible Approaches for Computing $M_{\mathbf{up}}$ and $M_{\mathbf{low}}$ of Score-recursive WPTS's

Fix a score-recursive WPTS Π, and assume it has (1) the concentration property, i.e., $\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ for $c_1, c_2 > 0$, (2) the bounded-update property, and (3) the stepwise weight is bounded by e^{c_3} for $0 < c_3 < c_2$. Then given the bounded ranges B and B' as computed in Section 5, we derive the upper bound M_{up} and the lower bound M_{low} for the expected weight from $B' \backslash B$ as follows.

For any $\mathbf{v} \in B' \backslash B$,

$$
\begin{aligned}
[\![\Pi]\!](\mathbf{v}) &= \mathbb{E}_{\mathbf{v}}\left[w_T\right] \\
&= \sum_{n=1}^{\infty} \mathbb{P}(T = n) \cdot w_n \\
&\leq \sum_{n=0}^{\infty} \mathbb{P}(T > n) \cdot w_n \\
&\leq 1 + \sum_{n=1}^{\infty} \mathbb{P}(T > n) \cdot w_n \\
&= 1 + \sum_{n=1}^{n^*-1} \mathbb{P}(T > n) \cdot w_n + \sum_{n=n^*}^{\infty} \mathbb{P}(T > n) \cdot w_n \\
&\leq 1 + M + \sum_{n=n^*}^{\infty} \mathbb{P}(T > n) \cdot w_n \\
&= M' + \sum_{n=n^*}^{\infty} \mathbb{P}(T > n) \cdot w_n \\
&\leq M' + \sum_{n=n^*}^{\infty} c_1 \cdot e^{-c_2 \cdot n} \cdot w_n \\
&\leq M' + \sum_{n=n^*}^{\infty} c_1 \cdot e^{-c_2 \cdot n} \cdot e^{c_3 \cdot n} \\
&= M' + c_1 \cdot \sum_{n=n^*}^{\infty} (e^{c_3 - c_2})^n \\
&= M' + c_1 \cdot \frac{a}{1-q} \\
&= M_{\text{up}}
\end{aligned}
$$

where

$$
\sum_{n=1}^{n^*-1} \mathbb{P}(T > n) \cdot w_n \leq \sum_{n=1}^{n^*-1} w_n \leq \sum_{n=1}^{n^*-1} (e^{c_3})^n = \frac{a' \cdot (1 - (q')^{(n^*-1)})}{1 - q'} =: M
$$

and $a' = e^{c_3}$, $q' = e^{c_3}$. The first inequality is obtained from the fact that

$$
\mathbb{P}(T > n) = \mathbb{P}(T \geq n + 1) = \mathbb{P}(T = n + 1) + \mathbb{P}(T = n + 2) + \ldots,
$$

thus,

$$\mathbb{P}(T = n + 1) \leq \mathbb{P}(T > n).$$

The second inequality is derived by the fact that $\mathbb{P}(T > 0) \leq 1$ and $w_0 = w_{\text{init}} = 1$. The third inequality is obtained by the definition of M above. The fourth inequality is obtained by the concentration property, while the fifth inequality is derived by the bounded stepwise weight condition.

For M_{low}, we trivially set $M_{\text{low}} = 0$. We can refine it heuristically, e.g., according to the monotonicity of the scoring function.

C.2 Overapproximation via Polynomial Interpolations

Given a non-polynomial function $f(x)$ over the interval $I = [a, b]$, we aim to approximate $f(x)$ by polynomials $p(x)$'s. The correctness of approximation is based on a classical theorem called Weierstrass' Theorem [24].

Theorem 9 (Weierstrass' Theorem). *Let $f(x)$ be a continuous function on the (closed) interval $[a, b]$. Then there is a sequence of polynomials $p_n(x)$ (of degree n) such that*

$$\lim_{n\to\infty} ||f - p_n||_\infty = 0.$$

We also need the following theorem to measure the derived polynomials. The property of Lipschitz continuity supports the following theorem easily.

Theorem 10. *Suppose $r(x)$ is a continuous and differentiable function on a compact convex set $\Psi \subseteq \mathbb{R}$. Assume that a collection of points $\{x_1, x_2, \ldots, x_k\}$ are sampled uniformly from Ψ and $s \in \mathbb{R}_{>0}$ is the sampling spacing. Let $r_0 = max\{|r(x_1)|, |r(x_2)|, \ldots, |r(x_k)|\}$, and $\beta = sup_{x\in\Psi} ||\nabla r(x)||$, then*

$$|r(x)| \leq \beta \cdot s + r_0, \ \forall\ x \in \Psi. \tag{6}$$

Then our scheme is as follows.

- Split the interval $I = [a, b]$ uniformly into m partitions, i.e., $I_1 = [a_1, b_1], I_2 = [a_2, b_2], \ldots, I_m = [a_m, b_m]$.
- For each partition $I_i = [a_i, b_i]$, define a n-degree polynomial $p_n^i(x) := \sum_{j=0}^{n} c_{ij} \cdot x^j$.
  1. Pick a non-negative integer $k > n$ and sample k points uniformly from f over I_i. That is,

     $$D = \{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_k, f(x_k))\}$$

     where $x_l \in I_i$ for all $1 \leq l \leq k$.
  2. Let $p_n^i(x_l) = f(x_l)$ for all $1 \leq l \leq k$, then we have a linear system $\mathbf{V} \cdot \mathbf{c} = \mathbf{f}$ where

     $$\mathbf{V} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^n \end{bmatrix},$$

     $\mathbf{c} = [c_{i0}, c_{i1}, \ldots, c_{in}]^T$ and $\mathbf{f} = [f(x_1), f(x_2), \ldots, f(x_k)]^T$.

3. By solving the above overdetermined system, we obtain $p_n^i(x)$ as the approximation of $f(x)$ over the interval $I_i = [a_i, b_i]$.
4. Having $p_n^i(x)$, evaluate an error bound γ_i such that

$$\forall\ x \in I_i,\ |f(x) - p_n^i(x)| \leq \gamma_i. \tag{7}$$

Let $r(x) = f(x) - p_n^i(x)$ and $\Psi = I_i$, then we obtain $r_0 = max\{|r(x_1)|, |r(x_2)|, \ldots, |r(x_k)|\}$ by Theorem 10. To derive the Lipschitz constant β of $r(x)$ over the interval I_i, we pick a non-negative integer $q = 10k$, and sample q points uniformly from f, i.e., we have another collection of points $\{x'_1, x'_2, \ldots, x'_q\}$. Let $\beta = max\{|\nabla r(x'_1)|, \ldots, |\nabla r(x'_q)|\}$, then

$$\gamma_i := \beta \cdot s + r_0$$

where s is the corresponding sampling spacing of the q points.

– Now we have a set D_p of tuples of intervals, polynomials and error bounds, i.e.,

$$D_p = \{(I_1, p_n^1(x), \gamma_1), \ldots, (I_m, p_n^m(x), \gamma_m)\} \tag{8}$$

The approximation error bounds γ_i's are taken into account when we synthesize the polynomial template h. Given a non-polynomial function $f(x)$ such that $\texttt{score}(f(x))$ occurs in the program, we obtain a set D_p in the form of (8). For each interval I_i, we introduce a new variable r_i and approximate $f(x)$ over I_i as $p_n^i(x) + r_i$ with $r_i \in [-\gamma_i, \gamma_i]$. That is, for $1 \leq i \leq m$, we have

$$\forall\ x \in I_i,\ f(x) \approx p_n^i(x) + r_i \text{ with } r_i \in [-\gamma_i, \gamma_i]. \tag{9}$$

For a state $(\ell, \mathbf{v})$ such that ℓ is the location before the command $\texttt{score}(f(x))$, there is the unique transition $\tau = \langle \ell, true, F \rangle$ such that $F = \langle \ell', 1, \mathbf{1}, f \rangle$ and ℓ' is the location that follows the command $\texttt{score}(f(x))$. Then for all valuations $\mathbf{v} \in I(\ell) \wedge \Phi_B$ and $1 \leq i \leq m$, it should hold that

– for all $\mathbf{v}[x] \in I_i$ and $r_i \in [-\gamma_i, \gamma_i]$, we have that $ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$ (for upper bounds) and $ewt(h)(\ell, \mathbf{v}) \geq h(\ell, \mathbf{v})$ (for lower bounds) where

$$ewt(h)(\ell, \mathbf{v}) = (p_n^i(x) + r_i) \cdot h(\ell', \mathbf{v}).$$

D Application of Positivstellensatz's

In **Step A5** of our algorithm, constraints are established in the form $\forall \mathbf{v} \in P.(g(\mathbf{v}) \geq 0)$ where P is a polyhedron over program variables V_p and are grouped conjunctively. Thus, the key point is how to tackle the each such constraint. In our algorithm, we follow the exact treatment through the application of Putinar's and Handelman's Positivstellensatz in [35,22]. Below we describe the detailed application.

D.1 Application of Putinar's Positivstellensatz

We recall Putinar's Positivstellensatz below.

Theorem 11 (Putinar's Positivstellensatz [35]). *Let V be a finite set of real-valued variables and $g, g_1, \ldots, g_m \in \mathbb{R}[V]$ be polynomials over V with real coefficients. Consider the set $\mathcal{S} := \{\mathbf{x} \in \mathbb{R}^V \mid g_i(\mathbf{x}) \geq 0$ for all $1 \leq i \leq m\}$ which is the set of all real vectors at which every g_i is non-negative. If (i) there exists some g_k such that the set $\{\mathbf{x} \in \mathbb{R}^V \mid g_k(\mathbf{x}) \geq 0\}$ is compact and (ii) $g(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{S}$, then we have that*

$$g = f_0 + \textstyle\sum_{i=1}^{m} f_i \cdot g_i \tag{10}$$

for some polynomials $f_0, f_1 \ldots, f_m \in \mathbb{R}[V]$ such that each polynomial f_i is the a sum of squares (of polynomials in $\mathbb{R}[V]$), i.e. $f_i = \sum_{j=0}^{k} q_{i,j}^2$ for polynomials $q_{i,j}$'s in $\mathbb{R}[V]$.

In this work, we utilize the sound form in (10) for witnessing a polynomial g to be non-negative over a polyhedron P for each constraint $\forall \mathbf{v} \in P.(g(\mathbf{v}) \geq 0)$ from **Step A5** of our algorithm. Let $\forall \mathbf{v} \in P.(g(\mathbf{v}) \geq 0)$ be such a constraint for which the polyhedron P is defined by the linear inequalities $g_1 \geq 0, \ldots, g_m \geq 0$. Let $V_\mathrm{p} = \{v_1, v_2, \ldots, v_t\}$ be the set of program variables and define $\mathbf{M}_d(V_\mathrm{p}) = \{m_1, m_2, \ldots, m_r\}$ as the set of all monomials of degree at most d over V_p, i.e. $\mathbf{M}_d(V_\mathrm{p}) := \{\prod_{i=1}^{t} v_i^{\alpha_i} \mid \forall i \;\; \alpha_i \in \mathbb{N} \;\wedge\; \sum_{i=1}^{t} \alpha_i \leq d\}$. The application of Putinar's to $\forall \mathbf{v} \in P.(g(\mathbf{v}) \geq 0)$ has the following steps.

- First, represent each f_i in Eq. (10) as the positive semidefinite form $f_i = \mathbf{v}^\mathrm{T}\mathbf{Q}_i\mathbf{v}$ subject to the positive semidefinite constraint where each $\mathbf{Q}_i$ is a real matrix whose every entry is an unknown parameter.
- Second, compute an equation in the form (10) whose coefficients are affine expressions in the unknown coefficients from our templates and the unknown entries in the matrices $\mathbf{Q}_i$'s.
- Third, establish the affine constraints between the unknown coefficients in the templates and the unknown entries in the matrices Q_i's by matching the coefficients at the LHS and the RHS of the equation obtained from the previous step.

The overall application processes all such constraints from **Step A5** of our algorithm by (i) collecting all the affine and the semidefinite constraints from the first and the third steps above and (ii) solve them by semidefinite programming.

D.2 Application of Handelman's Positivstellensatz

To present Handelman's Positivstellensatz, we need the notion of monoid as follows. Below we consider an arbitrary finite collection $\Gamma = \{g_1, \ldots, g_k\}$ $(k \geq 1)$ of linear functions (i.e., degree-1 polynomials) in the program variables.

Definition 7 (Monoid). *The* monoid *of Γ is defined by:*

$$\mathit{Monoid}(\Gamma) := \left\{ \prod_{i=1}^{k} h_i \mid k \in \mathbb{N}_0 \text{ and } h_1, \ldots, h_k \in \Gamma \right\} \quad .$$

Then in our context, Handelman's Positivstellensatz can be formulated as follows.

Theorem 12 (Handelman's Positivstellensatz [22]). *Let g be a polynomial in the program variables such that $g(\mathbf{v}) > 0$ for all program valuations $\mathbf{v} \in P := \{\mathbf{v}' \in \mathbb{R}^{|V_\mathrm{p}|} \mid g_1(\mathbf{v}') \geq 0, \ldots, g_k(\mathbf{v}') \geq 0\}$. If P is compact, then we have*

$$g = \sum_{i=1}^{d} a_i \cdot u_i \tag{11}$$

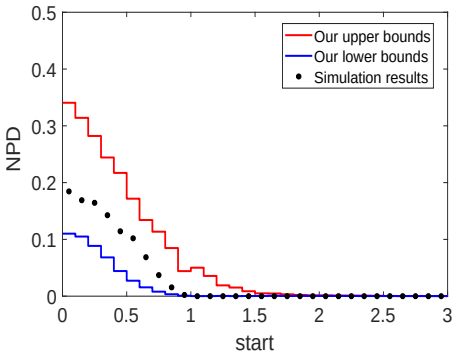
for some $d \in \mathbb{N}$, real numbers $a_1, \ldots, a_d \geq 0$ and $u_1, \ldots, u_d \in \mathit{Monoid}(\Gamma)$.

To apply Handelman's Positivstellensatz, we consider a natural number which serves as a bound on the number of multiplicands allowed to form an element in $\mathit{Monoid}(\Gamma)$. Then Eq. (11) results in a system of linear equalities that involves $a_1, \ldots, a_d$ and the coefficents of g. The application of Handelman's Positivstellensatz to each $\forall \mathbf{v} \in P.(g(\mathbf{v}) \geq 0)$ is simpler than that of Putinar's Positivstellensatz, and is as follows.

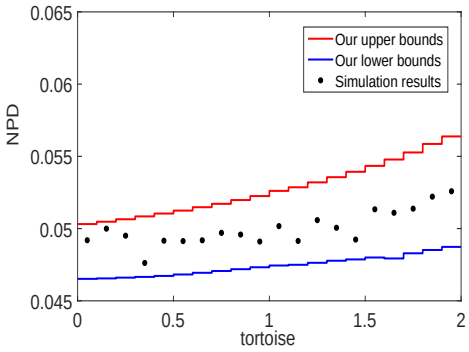
- First, compute an equation in the form (11) whose coefficients are affine expressions in the unknown coefficients from our templates and the fresh variables $a_1, \ldots, a_d$ from Eq. (11).
- Second, establish the affine constraints between the unknown coefficients in the templates and the fresh variables $a_1, \ldots, a_d$ from Eq. (11) by matching the coefficients at the LHS and the RHS of the equation obtained from the previous step.

The overall application processes all such constraints from **Step A5** of our algorithm by (i) collecting all the affine constraints from the second steps above and (ii) solve them by linear programming.

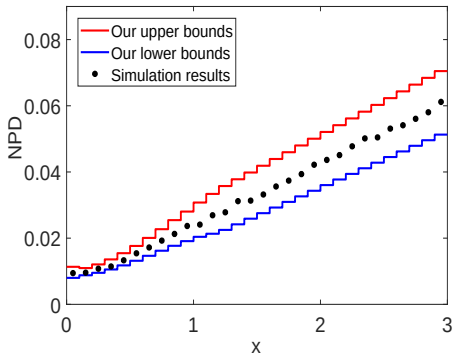
E Supplementary Materials for Experimental Results



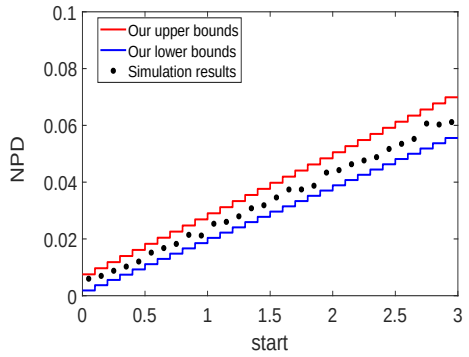
(a) Pedestrian v1



(b) Hare Tortoise Race v2

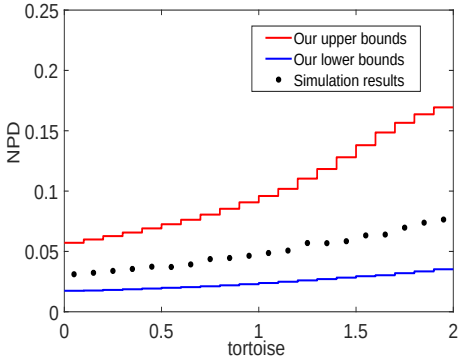


(c) Random Walk v1

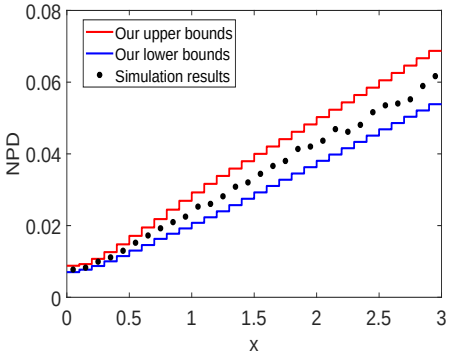


(d) Pedestrian Multi-branches v3

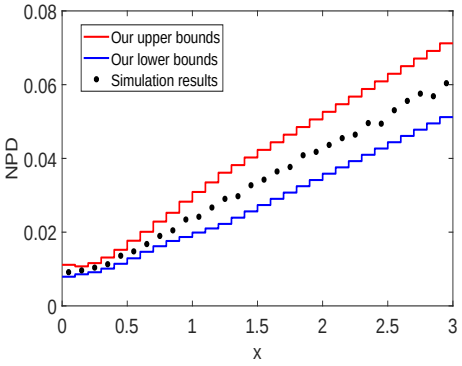
Fig. 7. Part 1: NPD Bounds of Novel Examples



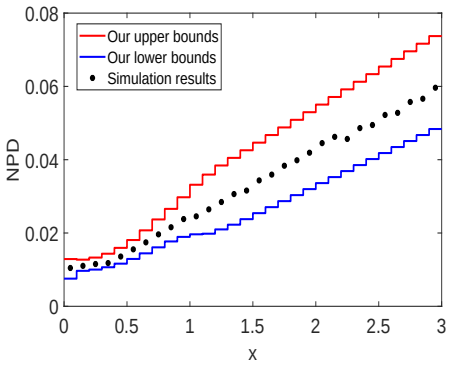
(a) Hare Tortoise Race v1



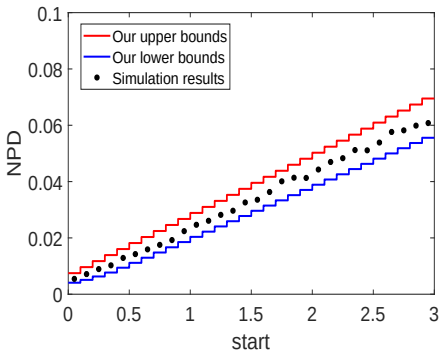
(b) Random Walk v2



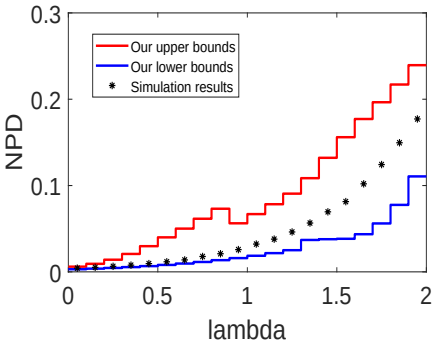
(c) Random Walk v3



(d) Random Walk v4

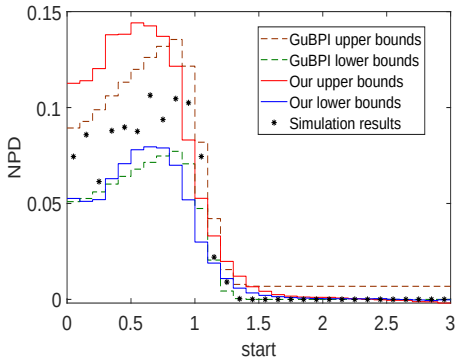


(e) Pedestrian Multi-branches v4

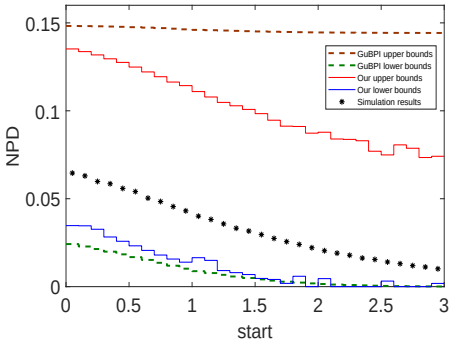


(f) Phylogenetic Birth Model

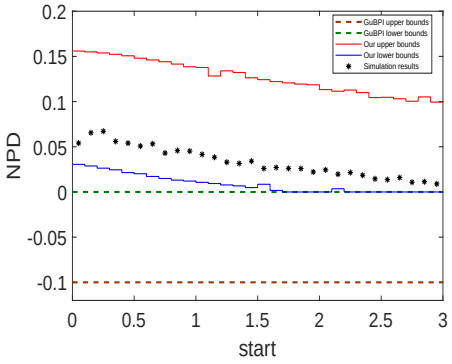
Fig. 8. Part 2: NPD Bounds of Novel Examples



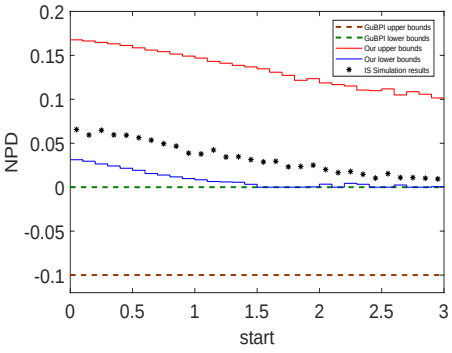
(a) Pedestrian



(b) Pedestrain Large Deviation

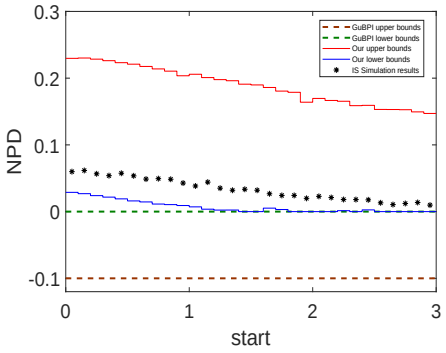


(c) Pedestrian Beta v1

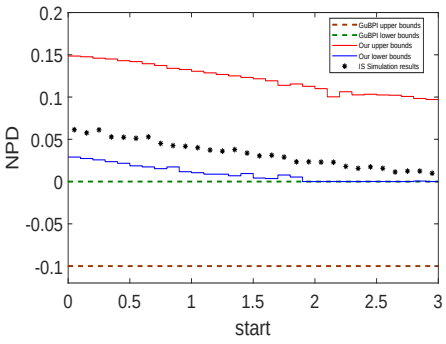


(d) Pedestrian Beta v2

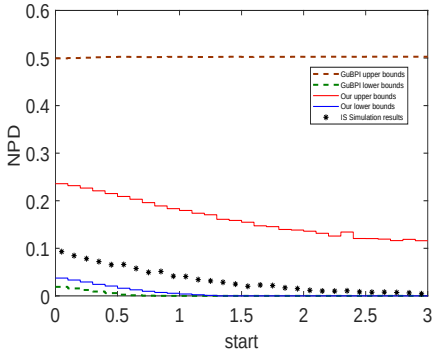
Fig. 9. Part 1: NPD Bounds of Our Approach and GuBPI



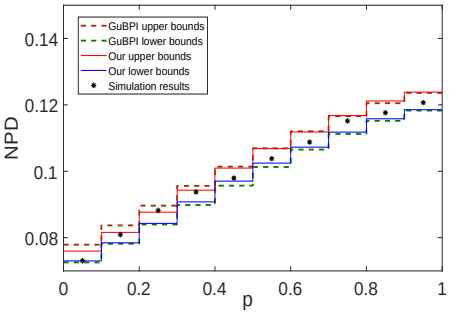
(a) Pedestrian Beta v3



(b) Pedestrian Beta v4



(c) Pedestrian Multi-branches v5



(d) Para Estimation Recursive

Fig. 10. Part 2: NPD Bounds of Our Approach and GuBPI