Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving

PEIXIN WANG*†, Nanyang Technological University, Singapore
TENGSHUN YANG*, Institute of Software at Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
HONGFEI FU†, Shanghai Jiao Tong University, China
GUANYAN LI, University of Oxford, United Kingdom
C.-H. LUKE ONG, Nanyang Technological University, Singapore

In Bayesian probabilistic programming, a central problem is to estimate the normalised posterior distribution (NPD) of a probabilistic program with conditioning via `score` (a.k.a. `observe`) statements. Most previous approaches address this problem by Markov Chain Monte Carlo and variational inference, and therefore could not generate guaranteed outcomes within a finite time limit. Moreover, existing methods for exact inference either impose syntactic restrictions or cannot guarantee successful inference in general.

In this work, we propose a novel automated approach to derive guaranteed bounds for NPD via polynomial solving. We first establish a fixed-point theorem for the wide class of *score-at-end* Bayesian probabilistic programs that terminate almost-surely and have a single bounded score statement at program termination. Then, we propose a multiplicative variant of Optional Stopping Theorem (OST) to address *score-recursive* Bayesian programs where score statements with weights greater than one could appear inside a loop. Bayesian nonparametric models, enjoying a renaissance in statistics and machine learning, can be represented by score-recursive Bayesian programs and are difficult to handle due to an integrability issue. Finally, we use polynomial solving to implement our fixed-point theorem and OST variant. To improve the accuracy of the polynomial solving, we further propose a truncation operation and the synthesis of multiple bounds over various program inputs. Our approach can handle Bayesian probabilistic programs with unbounded while loops and continuous distributions with infinite supports. Experiments over a wide range of benchmarks show that compared with the most relevant approach (Beutner *et al.*, PLDI 2022) for guaranteed NPD analysis via recursion unrolling, our approach is more time efficient and derives comparable or even tighter NPD bounds. Furthermore, our approach can handle score-recursive programs which previous approaches could not.

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

Additional Key Words and Phrases: probabilistic programming, Bayesian inference, static analysis, martingales, fixed-point theory, posterior distributions

*Equal Contribution
†Corresponding authors

Authors' addresses: Peixin Wang, Nanyang Technological University, Singapore, Singapore, peixin.wang@ntu.edu.sg; Tengshun Yang, Institute of Software at Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, yangts@ios.ac.cn; Hongfei Fu, Shanghai Jiao Tong University, Shanghai, China, jt002845@sjtu.edu.cn; Guanyan Li, University of Oxford, Oxford, United Kingdom, guanyan.li@cs.ox.ac.uk; C.-H. Luke Ong, Nanyang Technological University, Singapore, Singapore, luke.ong@ntu.edu.sg.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2024 Copyright held by the owner/author(s).
ACM 2475-1421/2024/6-ART202
https://doi.org/10.1145/3656432

ACM Reference Format:
Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and C.-H. Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. *Proc. ACM Program. Lang.* 8, PLDI, Article 202 (June 2024), 26 pages. https://doi.org/10.1145/3656432

1 INTRODUCTION

Bayesian probabilistic programming [40, 50] is a programming paradigm that incorporates Bayesian reasoning into programming languages, and aims at first modelling probabilistic models as probabilistic programs and then analyzing the models through their program representations. Compared with traditional approaches [8, 10, 32, 33] that specify an ad-hoc programming language, probabilistic programming languages (PPLs) [50] provide a universal framework to perform Bayesian inference. PPLs have two specific constructs: `sample` and `score` [7].[1] The `sample` construct describes the prior probabilities, allowing to draw samples from a (prior) distribution. The `score` construct records the likelihood of observed data in the form of "`score`(weight)",[2] and is typically used to weight the current execution in Monte Carlo simulation. Nowadays, Bayesian probabilistic programming has become an active research subject in statistics, machine learning and programming language communities, for which typical Bayesian programming languages include Pyro [5], WebPPL [21], Anglican [49], Church [20], etc.

In this work, we consider the analysis of the normalised posterior distribution (NPD) in Bayesian probabilistic programs. The general statement of the problem is that: given a prior distribution $p(z)$ over the latent variables $z \in \mathbb{R}^n$ of interest, and a probabilistic model represented by a probabilistic program whose distribution $p(x, z)$ is obtained by observing the event $x \in \mathbb{R}^m$ with the likelihood $p(x|z)$, the target is to calculate the NPD $p(z|x)$ by Bayes' rule. There are two mainstream variants of the NPD conditioning. The first is soft conditioning [46] that assigns a non-negative weight to the program based on the probability (density) of a given event occurring. The second is hard conditioning that restricts the weights to be either 0 or 1. Hard and soft conditioning are incomparable in general, as in some situations "hard conditioning is a particular case of soft conditioning" [40, Page 42], while in other cases hard conditioning is more general.

In the literature, there are two classes of approaches to address the NPD problem. The first is approximate approaches that estimate the NPD by random simulation, while the second is formal approaches that aim at deriving guaranteed bounds for NPD. In approximate approaches, two dominant methods are Markov chain Monte Carlo [17] and variational inference [6]. Although approximate approaches can produce approximate results efficiently, they cannot provide formal guarantee within a finite time limit. Moreover, as shown in Beutner et al. [4], approximate approaches may produce inconsistent results between different simulation methods, which led the machine learning community to develop new variants [31]. In formal approaches, there is a large amount of existing works such as (λ)PSI [18, 19], AQUA [25], Hakaru [35] and SPPL [43], aiming to derive 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. In recent works, Zaiser et al. [59] and Klinkenberg et al. [27] used probability generating functions (PGF) to do exact inference for NPD. However, the former work cannot handle loopy programs, and both of them require a closed-form solution to NPD and only work for discrete observations. The recent work by Beutner et al. [4] infers guaranteed bounds for NPD and handles unbounded recursion and continuous distributions. This approach relies on recursion unrolling and hence suffers from the path explosion problem.

[1] Sometimes `observe` is used instead of `score` [22], which has the same implicit effect.
[2] The argument "weight" corresponds to the likelihood each time the data is observed.

Challenges and gaps. In this work, we focus on developing formal approaches to derive guaranteed bounds for NPD over loopy probabilistic programs in the setting of soft conditioning. From the literature, a main challenge is to develop new techniques that circumvent the path explosion problem from the approach [4]. Another challenge (and gap) is that existing approaches cannot handle the situation where `score` statements with weights greater than 1 appear inside a loop (which we refer to as *score-recursive* programs), which has received significant attention in statistical phylogenetics [41, 47].

For score-recursive programs, the following example shows that `score` inside a loop may cause an integrability issue and thus requires careful treatment. Consider a simple loop "**while true do if prob**(0.5) **then break else score**(3) **fi od**". In each loop iteration, the loop terminates directly with probability $\frac{1}{2}$, and continues to execute a score command "`score(3)`" with the same probability. It follows that the normalising constant in NPD 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. This is noted in e.g. Staton et al. [46] thatunbounded weights may introduce the possibility of "infinite model evidence errors". To circumvent the drawback, previous results (e.g., Borgström et al. [7]) allow only 1-bounded weights.

In probabilistic program analysis, polynomial solving [9, 10, 12, 52, 55] is a well-established technique and naturally avoids the path explosion problem in the approach of Beutner et al. [4]. In this work, we leverage polynomial solving to address the NPD problem. Note that simply applying well-known polynomial solving techniques does not suffice for the following reasons: (a) Polynomial solving is tight usually over a bounded region (see e.g., Weierstrass Approximation Theorem [26]), and in general is not accurate if the region is unbounded; (b) Polynomial solving synthesizes a single bound. However, having a single bound is not enough to get tight bounds for NPD, as one needs different bounds for different program inputs to achieve tightness in the normalisation.

We address the challenges and gaps mentioned above. Our contributions are as follows.

Our contributions. In this work, we present the following contributions:

- First, we establish a fixed-point theorem and a multiplicative variant of Optional Stopping Theorem (OST) [15, 58]. Our fixed-point theorem targets Bayesian probabilistic programs that have almost-sure termination and a single score statement at the end of the programs with a bounded score function (referred to as *score-at-end* Bayesian programs), which is a wide class of Bayesian programs in the literature [4, 18, 19]. Our OST variant targets *score-recursive* Bayesian programs and addresses the integrability issue in these programs.
- Second, we apply polynomial solving techniques with our fixed-point theorem and OST variant. In addition to existing polynomial solving techniques, our approach improves the accuracy of the derived NPD bounds by the following: First, we propose a novel truncation operation that truncates a probabilistic program into a bounded range of program values. Second, we devise our algorithm to synthesize multiple bounds for various program inputs.

Experimental results show that our approach can handle a wide range of benchmarks including non-parametric examples such as Pedestrian [4] and score-recursive examples such as phylogenetic models [41]. Compared with the previous approach [4] over score-at-end benchmarks, our approach reduces the runtime by up to 15 times, while deriving comparable or even tighter bounds for NPD.

Limitations. Our approach has the combinatorial explosion in the degree of polynomial solving. However, by our experimental results, a moderate choice of the degree (e.g., ≤ 10) suffices. Moreover, our synthesis of multiple bounds for various inputs mitigates the combinatorial explosion. Another limitation is that in our polynomial solving, we utilize linear and semidefinite programming solvers, which may produce unsound results due to numerical errors.

Due to space constraints, we put the missing technical and proof details to our full version [57].

```
        start := sample uniform(0, 3) ;
        pos := start;   dist := 0 ;
ℓ_init : h(pos, dis) = a_1 · pos + a_2 · dis + a_3
        B = {(pos, dis) | pos ∈ [0, 5], dis ∈ [0, 5]}
        M = 2.1 × 10^{-330}
        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(pdf(normal(1.1, 0.1), dist)) ;
        polynomial approximation g
        return start
ℓ_out :
```

Fig. 1. A Pedestrian Random Walk

```
lambda := sample uniform(0, 2) ;
time := 10 ;   amount := 0 ;
ℓ_init : 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
ℓ_out :
```

Fig. 2. A Phylogenetic Birth Model

2 MOTIVATING EXAMPLES

We present two motivating examples to highlight our key novelties.

2.1 Pedestrian Random Walk

Consider the pedestrian random walk example [30] in Fig. 1. In this example, a pedestrian is lost on the way home, and she only knows that she is at most 3 km away from her house. Thus, she starts to repeatedly walk a uniformly random distance of at most 1 km in either direction of the road with equal probability, until reaching her house. Upon the arrival, an odometer tells that she has walked 1.1 km in total. However, this odometer was once broken and the measured distance is normally distributed around the true distance with a standard deviation of 0.1 km. We want to infer the posterior distribution of the starting point.

This example is modeled as a non-parametric probabilistic program whose number of loop iterations is unbounded, where the blue part is the annotations used for the Bayesian inference of this example. In the program, the variables $start$, pos, $step$, dis represent the starting point, the current position, the distance walked in the next step and the travelled distance so far of the pedestrian, respectively. As the program variables $start, step$ simply receive samples, the key program variables are $pos, dist$.

This program terminates with probability 1 and has a score statement at termination with a bounded score function indicated by the probability density function $pdf(\mathbf{normal}(1.1, 0.1), dist)$ of the normal distribution with mean 1.1 and deviation 0.1. Note that $pdf(\mathbf{normal}(1.1, 0.1), dist) = pdf(\mathbf{normal}(dist, 0.1), 1.1)$. Bayesian probabilistic programs that have score statements at termination widely exist in the literature [4, 18, 19], and we call them *score-at-end* programs. We propose a novel approach for Bayesian inference over such programs via fixed-point conditions.

Our approach decomposes the Bayesian inference into the computation of expected weights, i.e., expected score values from various initial program inputs. For this example, we have the initial value for the variable pos ranges over $[0, 3]$, and the initial value for the variable $dist$ fixed

to be 0. We partition the range of initial values into multiple pieces (e.g., dividing $[0, 3]$ into $[0, 0.1], [0.1, 0.2], \ldots, [2.9, 3]$), and solve the expected weights for each piece. Within each piece, we establish a polynomial template h for the upper bound to be solved and use fixed-point theory to solve the template. A template for this example at the entry point of the loop is given by the linear template $h(pos, dis) = a_1 \cdot pos + a_2 \cdot dis + a_3$ (see the annotations), and the prefixed-point condition to solve h as an upper bound is given by

$$ewt(h) := 0.5 \cdot \mathbb{E}_{step}\left[[pos - step \geq 0] \cdot h(pos - step, dis + step) + [pos - step < 0] \cdot g(dis + step)\right] + 0.5 \cdot \mathbb{E}_{step}\left[h(pos + step, dis + step)\right] \leq h(pos, dis).$$

where the left-hand-side $ewt(h)$ of the inequality expresses the expected value of the template after one loop iteration, for which the expectation is taken w.r.t the sampling of the $step$ variable. Note that here we use a polynomial g to approximate the density function of the normal distribution to allow a uniform polynomial reasoning.

For this example, simply solving the polynomial template h does not suffice, as we have find that polynomial solving produces trivial constant bounds even with high degrees. Hence, we consider a novel truncation operation that truncates the probabilistic program into a bounded range, so that we can utilize the strong approximation ability of polynomials over bounded ranges. In this example, we can choose the bounded range to be $B = \{(pos, dis) \mid pos \in [0, 5], dis \in [0, 5]\}$, so that the program state space is partitioned into sets of states within B and outside B. The execution of the program is also changed in the sense that once the execution jumps out of the bounded range, the program halts immediately. In conjunction with the bounded range, we associate a polynomial $\mathcal{M}$ that over-approximates the expected weights outside the bounded range. In this example, we have that when jumping out of the bounded range, either $dist \geq 5$ or $pos \geq 5$, and in both cases we have $dist \geq 5$ (as the pedestrian needs to travel at least the maximal pos in the walk). Hence, we can choose $\mathcal{M} = 2.1 \times 10^{-330}$ since $pdf(\textbf{normal}(1.1, 0.1), dist) \leq 2.1 \times 10^{-330}$ when $dist \geq 5$ according to its monotonicity. Given the bounded range B with the over-approximation $\mathcal{M}$, we then solve the template h by the following (informal) modified prefixed-point conditions: (a) When within the bounded range B, we have $ewt(h) \leq h$; (b) When outside B, we have $\mathcal{M} \leq h$.

The example is handled in [4] by exhaustive recursion unrolling that has the path-explosion problem. Our approach circumvents path explosion and derives comparable bounds to the approach in Beutner et al. [4] with runtime two-thirds of that of Beutner et al. [4].

2.2 Phylogenetic Birth Model

Consider a simplified version of the phylogenetic birth model [41], where a species arises with a birth-rate $lambda$, and it propagates with a simplified constant likelihood of 1.1 at some time interval. For simplicity, we assume constant weights that can be viewed as over-approximation for a continuous density function. This example can be modelled as a probabilistic loop in Fig. 2. In this program, the variables $lambda, time, amount, wait$ stand for the birth rate of the species, the remaining propagation time, the current amount of the species and the propagation time to be spent, respectively. The variable $lambda$ is associated with a prior distribution, and the NPD problem is to infer its posterior distribution given the species evolution described by the loop.

The main difficulty to analyze the NPD of this example is that its loop body includes a score statement `score(1.1)` with the constant score function greater than 1. We call such programs *score-recursive*. As stated previously, this incurs an integrability issue that cannot be solved by previous approaches. To address this difficulty, we propose a novel multiplicative variant of Optional Stopping Theorem (OST) that allows a stochastic process to scale by a multiplicative factor during its evolution. Based on the OST variant, we apply polynomial solving with truncation as in our

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

Fig. 3. Syntax of Our Probabilistic Programming Language

fixed-point approach. Our experimental result on this example shows that the derived bounds match the simulation result with 10^6 samples.

3 PRELIMINARIES

We first review basic concepts from probability theory, then present our Bayesian probabilistic programming language, and finally define the normalised posterior distribution (NPD) problem. We denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the sets of all natural numbers, integers, and real numbers, respectively.

3.1 Basics of Probability Theory

We recall several basic concepts and refer to standard textbooks (e.g. [37, 58]) for details.

Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a *random variable* is an $\mathcal{F}$-measurable function $X : \Omega \longrightarrow \mathbb{R} \cup \{+\infty, -\infty\}$ where the measurable space over $\mathbb{R} \cup \{+\infty, -\infty\}$ is taken as the Borel space. The *distribution function* F of X is given by $F(x) = \mathbb{P}(\{\omega : X(\omega) \le x\})$. A non-negative Borel measurable function f is a *density function* of X if it satisfies $F(x) = \int_{-\infty}^{x} f(t)\mathrm{d}t$. 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., $\int_\Omega X\, \mathrm{d}\mathbb{P}$. A *filtration* of $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence $\{\mathcal{F}_n\}_{n=0}^{\infty}$ of σ-algebras such that for every $n \ge 0$, 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 $T : \Omega \longrightarrow \mathbb{N} \cup \{0, \infty\}$ such that for every $n \ge 0$, the event $\{T \le n\}$ is in $\mathcal{F}_n$. Recall the Borel measurable space $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$ where $\Sigma_{\mathbb{R}^n}$ is the σ-algebra generated by the open subsets in $\mathbb{R}^n$.

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 $n \ge 0$, X_n is a random variable in $(\Omega, \mathcal{F}_n, \mathbb{P})$. A 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 $n \ge 0$, $\mathbb{E}(|X_n|) < \infty$ and it holds almost surely that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] = X_n$ (resp., $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \le X_n$, $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \ge X_n$). Applying martingales to the formal analysis of probabilistic programs is a well-studied technique [8, 10, 13].

3.2 Bayesian Probabilistic Programs

The syntax of our Bayesian probabilistic programming language (PPL) is given in Fig. 3, where $c, c_1, c_2 \in \mathbb{R}$ are real constants, $p \in (0, 1]$ and the metavariables S, B and E stand for statements, boolean and arithmetic expressions, respectively. Our PPL is imperative with the usual conditional, loop, sequential and probabilistic branching structures as well as the following new structures: (a) sample constructs of the form "**sample** D" that sample a value from a prescribed distribution D (e.g., normal distribution, uniform distribution, etc.) over $\mathbb{R}$; (b) score statements of the form "**score**(EW)" that weight the current execution with a value expressed by EW, where $\mathit{pdf}(D, x)$ is the value of the probability density function w.r.t. the distribution D at x. We also have return statements (i.e., **return**) that return the value of a program variable. Note that although probabilistic branches can be derived from sampling of Bernoulli distributions, we include probabilistic branches here to have specific algorithmic treatment for probabilistic control flows.

In our PPL, we distinguish two disjoint sets of variables in a program: (i) the set V_{p} of *program variables* whose values are determined by assignments (i.e., the expressions at the RHS of ":=");

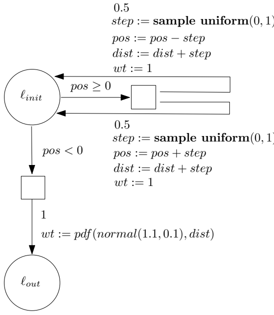


Fig. 4. The WPTS of Pedestrian

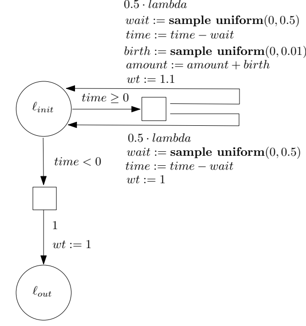


Fig. 5. The WPTS of Phylogenetic Model

(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 a sampling variable). A *valuation* on a set V of variables is a function $\mathbf{v} : V \longrightarrow \mathbb{R}$ that assigns a real value to each variable in V. A *program* (resp. *sampling*) valuation is a valuation on V_p (resp. V_r). The set of program (resp. *sampling*) valuations is denoted by Val_p (resp. Val_r), respectively. For the sake of convenience, we fix the notations in the following way: we always use $\mathbf{v} \in Val_\mathrm{p}$ to denote a program valuation, and $\mathbf{r} \in Val_\mathrm{r}$ to denote a sampling valuation.

Example 3.1. Fig. 1 shows a Bayesian probabilistic program written in our PPL. In this program, the set of program variables is $V_\mathrm{p} = \{start, pos, dis, step\}$, and the set of sampling variables is $V_\mathrm{r} = \{\textbf{sample uniform}(0, 3), \textbf{sample uniform}(0, 1)\}$. At the execution of **sample uniform**$(0, 3)$, it samples a value uniformly from $[0, 3]$ and assigns it to the variable $start$ in the initialization. During the loop iteration, each time **sample uniform**$(0, 1)$ is executed, it samples a value uniformly from $[0, 1]$ and assigns the value to the variable $step$. □

Below we present the semantics for our PPL. In the literature, existing semantics are either measure-based [29, 46] or sampling-based [4, 30]. To facilitate the development of our approach, we consider the *transition-based* semantics [9, 11] so that each probabilistic program is transformed into an equivalent form of *weighted probabilistic transition system* (WPTS). A WPTS extends a PTS [9, 11] with weights and an initial probability distribution.

Definition 3.2 (WPTS). A *weighted probabilistic transition system* (WPTS) Π 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 *sampling* variables.
- L is a finite set of *locations* with special locations $\ell_\mathrm{init}, \ell_\mathrm{out} \in L$. Informally, a location corresponds to a cut point in a Bayesian probabilistic program, ℓ_init is the initial location and ℓ_out represents program termination.
- μ_init is the *initial probability distribution* over $\mathbb{R}^{|V_\mathrm{p}|}$ with a bounded support (denoted by supp (μ_init)), We call each $\mathbf{v} \in$ supp (μ_init) an *initial program valuation.*
- $\mathcal{D}$ is a function that assigns a probability distribution $\mathcal{D}(r)$ to each $r \in V_\mathrm{r}$. We abuse the notation so that $\mathcal{D}$ also denotes the joint distribution of all independent variables $r \in V_\mathrm{r}$.
- $\mathfrak{T}$ 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 (a) $\ell \in L$ is the *source location*, (b) ϕ is the *guard condition* which is a logical formula over program variables V_p, and (c) each $F_j := \langle \ell'_j, p_j, upd_j, wt_j \rangle$ is called a *weighted fork* for which

(i) $\ell'_j \in L$ is the *destination location* of the fork, (ii) $p_j \in (0,1]$ is the probability of occurrence of this fork, (iii) $upd_j : \mathbb{R}^{|V_\mathrm{p}|} \times \mathbb{R}^{|V_\mathrm{r}|} \longrightarrow \mathbb{R}^{|V_\mathrm{p}|}$ is an *update function* that takes as inputs the current program and sampling valuations and returns an updated program valuation, and (iv) $wt_j : \mathbb{R}^{|V_\mathrm{p}|} \times \mathbb{R}^{|V_\mathrm{r}|} \longrightarrow [0,\infty)$ is a *score function* that gives the likelihood weight of this fork depending on the current program and sampling valuations.

In a WPTS, update functions correspond to assignment statements to program variables, and score functions correspond to the cumulative multiplicative weight of a basic block of statements from the score statements in the block. Note that if there is no score statement in the block, then the score function of the block is constantly 1. We always assume that a WPTS Π is *deterministic* and *total*, i.e., (1) there is no program valuation that simultaneously satisfies the guard conditions of two distinct transitions from the same source location, and (2) the disjunction of the guard conditions of all the transitions from any source location is a tautology. The transformation from a probabilistic program into its WPTS can be done in a straightforward way (see e.g. [9, 12]).

Example 3.3. Fig. 4 shows the WPTS of the program in Fig. 1 which has two locations $\ell_{\mathrm{init}}, \ell_{\mathrm{out}}$. The value of $step$ is initialised to 0. The initial probability distribution μ_{init} over $\mathbb{R}^{|V_\mathrm{p}|}$ is determined by the joint distribution of $(start, pos, dis, step)$ where $start \sim uniform(0,3)$ and $pos, dis, step$ observe the Dirac measures $Dirac(\{start\})$, $Dirac(\{0\})$ and $Dirac(\{0\})$, respectively, e.g., the probability of the event "$dis \in \{0\}$" equals 1. The circle nodes represent locations and square nodes model the forking behavior of transitions. An edge entering a square node is labeled with the guard condition of its respective transition, while an edge entering a circle node stands for a fork, which is associated with its probability, update functions and score functions. The WPTS of the program in Fig. 2 is analogously given in Fig. 5. [3] □

Below we specify the semantics of a WPTS. Consider a WPTS Π in the form of (†). Given a program valuation $\mathbf{v}$ and a guard condition 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 $\mathbf{v}$.

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 cumulative multiplicative likelihood weight.

The semantics of Π is formalized by the infinite sequence $\Gamma = \{\widehat{\Theta}_n = (\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)\}_{n\geq 0}$ where each $(\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)$ is the random weighted state at the n-th 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 program valuation, the random variable for the multiplicative likelihood weight) at the n-th 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}, $\widehat{\mathbf{v}}_0 \in \mathrm{supp}\,(\mu_{\mathrm{init}})$ is sampled from the initial distribution μ_{init} and the initial weight $\widehat{w}_0$ is constantly set to 1.[4] Then, given the current random weighted state $\widehat{\Theta}_n = (\widehat{\ell}_n, \widehat{\mathbf{v}}_n, \widehat{w}_n)$ at the n-th step, 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 $\widehat{\ell}_n = \ell_{\mathrm{out}}$, 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 the probability p_j.
- Third, we obtain a sampling valuation $\mathbf{r} \in \mathrm{supp}\,(\mathcal{D})$ by sampling each $r \in V_\mathrm{r}$ independently from the probability distribution $\mathcal{D}(r)$.

[3] Here we omit the update functions if the values of program variables are unchanged.

[4] This follows the traditional setting in e.g. [4].

- 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 $(\ell'_j, upd_j(\hat{\mathbf{v}}_n, \mathbf{r}), \widehat{w}_n \cdot wt_j(\widehat{\mathbf{v}}_n, \mathbf{r}))$. Note that the weight is obtained in the style of the multiplicative score.

Unlike several semantics (such as Staton et al. [46]) that integrates score statements with sampling, our semantics separates them and have a special construct **score**$(-)$ for score statements.

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 n-th step in this program run with location ℓ_n, program valuation $\mathbf{v}_n$ and cumulative 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 $\Theta_n = (\ell, \mathbf{v}, w)$ for some n and w.

Example 3.4. Consider the WPTS in Example 3.3. Suppose the initial program valuation is $(1, 1, 0, 0)$ which means that the initial values of $start, pos, dis, step$ are $1, 1, 0, 0$, respectively. Then starting from the initial weighted state $(\ell_{\text{init}}, (1, 1, 0), 1)$, a program run could be

$$(\ell_{\text{init}}, (1, 1, 0, 0), 1) \to (\ell_{\text{init}}, (1, 0.5, 0.5, 0.5), 1) \to (\ell_{\text{init}}, (1, -0.1, 1.1, 0.6), 1) \to (\ell_{\text{out}}, (1, -0.1, 1.1, 0.6), 3.9894).$$

After the final execution, the program valuation becomes $(1, -0.1, 1.1, 0.6)$ and the loop terminates. We capture the likelihood weight with $pdf(\mathbf{normal}(1.1, 0.1), 1.1) = \frac{1}{\sqrt{2\pi}\times 0.1} e^{-\frac{0}{2\times 0.1^2}} = 3.9894$ and multiply it to the current weight (i.e., 1). Thus the final weight of this run is 3.9894. □

Given an initial program valuation $\mathbf{v}_{\text{init}}$ of a WPTS, one could construct a probability space over the program runs via their probabilistic execution described above and standard constructions such as general state space Markov chains [34]. We denote the probability measure in this probability space by $\mathbb{P}_{\mathbf{v}_{\text{init}}}(-)$ and the expectation operator by $\mathbb{E}_{\mathbf{v}_{\text{init}}}[-]$.

3.3 Normalised Posterior Distribution

Below we fix a WPTS Π in the form of (†). The *termination time* of the WPTS Π is the random variable T 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}. The WPTS Π is *almost-surely terminating* (AST) if $\mathbb{P}_{\mathbf{v}_{\text{init}}}(T < \infty) = 1$ for all initial program valuations $\mathbf{v}_{\text{init}} \in \text{supp}\,(\mu_{\text{init}})$. Given 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* $[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathcal{U})$ restricted to $\mathcal{U}$ is defined as $[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathcal{U}) := \mathbb{E}_{\mathbf{v}_{\text{init}}}[[\widehat{\mathbf{v}}_T \in \mathcal{U}] \cdot \widehat{w}_T]$ where $[-]$ is the Iverson bracket such that $[\phi] = 1$ if ϕ holds and $[\phi] = 0$ otherwise. (Recall that $\widehat{\mathbf{v}}_T$ and $\widehat{w}_T$ are the random vector and variable of the program valuation and the multiplicative likelihood weight at termination, respectively.) If $\mathcal{U} = \mathbb{R}^{|V_{\text{p}}|}$, then $[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_{\text{p}}|})$ is called the *unrestricted expected weight*, or simply *expected weight* for short. The normalised posterior distribution (NPD) is defined as follows.

Definition 3.5 (NPD). The *normalised posterior distribution* (NPD) posterior_Π of Π is defined by:

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

where $[\![\Pi]\!](\mathcal{U}) := \int_{\mathcal{V}} [\![\Pi]\!]_{\mathbf{v}}(\mathcal{U}) \cdot \mu_{\text{init}}(\mathrm{d}\mathbf{v})$ is the *unnormalised posterior distribution* w.r.t. $\mathcal{U}$ with $\mathcal{V} := \text{supp}\,(\mu_{\text{init}})$, and $Z_\Pi := [\![\Pi]\!](\mathbb{R}^{|V_{\text{p}}|})$ is the *normalising constant.* Π is *integrable* if $0 < Z_\Pi < \infty$.

In this work, we consider the automated interval bound analysis for the NPD of a WPTS. Formally, we aim to derive a tight interval $[l, u] \subseteq [0, \infty)$ for an integrable WPTS Π and any measurable set $\mathcal{U} \in \Sigma_{\mathbb{R}^{|V_{\text{p}}|}}$ such that $l \leq \text{posterior}_\Pi(\mathcal{U}) \leq u$. To achieve this, we consider bounds on $[\![\Pi]\!](\mathcal{U}), Z_\Pi$.

NPD Bounds. Assume we have two intervals $[l_{\mathcal{U}}, u_{\mathcal{U}}], [l_Z, u_Z] \subseteq [0, \infty)$ such that the unnormalised posterior distribution $[\![\Pi]\!](\mathcal{U}) \in [l_{\mathcal{U}}, u_{\mathcal{U}}]$ and the normalising constant $Z_\Pi \in [l_Z, u_Z]$. If Π is integrable, then we have the NPD $\text{posterior}_\Pi(\mathcal{U}) \in [\frac{l_{\mathcal{U}}}{u_Z}, \frac{u_{\mathcal{U}}}{l_Z}]$.

To analyze the quantity $[\![\Pi]\!](\mathcal{U})$ derived from expected weights restricted to $\mathcal{U}$, we construct a new WPTS $\Pi_{\mathcal{U}}$ from the original Π and a measurable set $\mathcal{U} \in \Sigma_{\mathbb{R}^{|V_p|}}$. Consider a probabilistic program P and its WPTS Π, given a measurable set $\mathcal{U} \in \Sigma_{\mathbb{R}^{|V_p|}}$, we construct a new program $P_{\mathcal{U}}$ by adding a conditional branch of the form "**if** $\mathbf{v}_T \notin \mathcal{U}$ **then score**(0) **fi**" immediately after the termination of P and obtain the WPTS $\Pi_{\mathcal{U}}$ of $P_{\mathcal{U}}$. Thus, $\Pi_{\mathcal{U}}$ simply restricts the program valuation at termination to $\mathcal{U}$ when taking the final cumulative weight. In this way, we transform the analysis of $[\![\Pi]\!](\mathcal{U})$ into that of $[\![\Pi_{\mathcal{U}}]\!](\mathbb{R}^{|V_p|})$ in the following, since $[\![\Pi]\!](\mathcal{U}) = [\![\Pi_{\mathcal{U}}]\!](\mathbb{R}^{|V_p|})$. Therefore, from Definition 3.5, we can reduce the bound analysis of NPD to that of unrestricted expected weights in the rest of the paper. See details in the full version [57].

4 THEORETICAL APPROACHES

Below we present two approaches for deriving bounds of unrestricted expected weights.

4.1 The Fixed-Point Approach

Our fixed-point approach targets *score-at-end* Bayesian programs that terminate almost-surely and have a single score statement with a bounded weight at the termination. We formally define the notion of score-at-end programs directly over WPTS's. A WPTS Π is *score-at-end* if: (i) Π has AST; (ii) there is exactly one transition τ in the WPTS that involves the destination location ℓ_{out}, and this transition τ takes the form $\tau = \langle \ell, -, F \rangle$ with the single weighted fork $F = \langle \ell_{\text{out}}, 1, id, wt \rangle$ where id is the identity function and wt is *score-bounded* by a constant $M > 0$, i.e., $wt \in [0, M]$; and (iii) all transitions other than the transition τ have the score function constantly equal to 1. Such programs widely exist in the literature [4, 18, 19].

To demonstrate the approach, we recall several basic concepts in lattice theory. Given a complete lattice $(K, \sqsubseteq)$ and a function $f : K \to K$, the *supremum* of K is denoted by $\bigsqcup K$, while the *infimum* of K is denoted by $\bigsqcap K$. An element $k \in K$ is called a *fixed-point* if $f(k) = k$. The *least* (resp. *greatest*) *fixed-point* of f, denoted by $\text{lfp} f$ (resp. $\text{gfp} f$), is the fixed-point that is no greater (resp. smaller) than every fixed-point under $\sqsubseteq$. Moreover, k is a *prefixed-point* if $f(k) \sqsubseteq k$ and a *postfixed-point* if $k \sqsubseteq f(k)$. We apply Tarski's fixed-point theorem as follows.

Theorem 4.1 (*Tarski* [48]). *Let $(K, \sqsubseteq)$ be a complete lattice and $f : K \to K$ be a monotone function. Then, both $\text{lfp}\ f$ and $\text{gfp}\ f$ exist. Moreover, $\text{lfp}\ f = \bigsqcap \{x \mid f(x) \sqsubseteq x\}$ and $\text{gfp}\ f = \bigsqcup \{x \mid x \sqsubseteq f(x)\}$.*

Based on Theorem 4.1, we present our fixed-point approach for score-at-end WPTS's. Below we fix a score-at-end WPTS Π. We define Λ as the set of states $(\ell, \mathbf{v})$ where ℓ is a location and $\mathbf{v}$ is a program valuation. Given a maximum finite value $M \in [0, \infty)$, we define a *state function* as a function $h : \Lambda \to [-M, M]$ such that for all $\mathbf{v} \in \mathbb{R}^{|V_p|}$, $h(\ell_{\text{out}}, \mathbf{v}) \in [0, M]$. The intuition of a state function h is that $h(\ell, \mathbf{v})$ gives an estimation of the expected weight when the WPTS Π starts with the initial location ℓ and the initial program valuation $\mathbf{v}$. We denote the set of all state functions with the maximum value M by $\mathcal{K}_M$. We also use the usual partial order $\leq$ on $\mathcal{K}_M$ that is defined in the pointwise fashion, i.e., for any $h_1, h_2 \in \mathcal{K}_M$, $h_1 \leq h_2$ iff $h_1(\ell, \mathbf{v}) \leq h_2(\ell, \mathbf{v})$ for all $(\ell, \mathbf{v}) \in \Lambda$. It is straightforward to verify that $(\mathcal{K}_M, \leq)$ is a complete lattice.

To connect the complete lattice $(\mathcal{K}_M, \leq)$ with expected weights, we define the *expected-weight function* ew_Π by $ew_\Pi(\ell_{\text{init}}, \mathbf{v}) := [\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_p|})$, and omit the subscript Π if it is clear from the context.

Informally, $ew_\Pi(\ell_{\text{init}}, \mathbf{v})$ is the expected weight of the program starting from an initial program valuation $\mathbf{v}$. Note that if the weight at termination is bounded by a maximum value M, then we have $ew_\Pi \in \mathcal{K}_M$. We consider the following higher-order function over the complete lattice $(\mathcal{K}_M, \leq)$.

Definition 4.2 (Expected-Weight Transformer). Given a finite maximum value $M \in [1, \infty)$, 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

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

In (1), the expectation $\mathbb{E}_{\mathbf{r}}[-]$ is taken over a sampling valuation $\mathbf{r}$ that observes the distribution $\mathcal{D}$.

Informally, given a state function h, the expected-weight transformer ewt_Π computes the expected weight $ewt_\Pi(h)$ after one step of WPTS transition. Note that in (1), the weight $wt_j(\mathbf{v}, \mathbf{r})$ equals 1 when the location ℓ does not refer to the score statement at the end of the program, as we consider that the WPTS Π is score-at-end. This implies that ewt_Π is indeed a higher-order operator for the complete lattice $(\mathcal{K}_M, \leq)$ when the maximum value M is a bound for the weights in the score statement. By the monotonicity of expectation, we have that ewt_Π is monotone.

We will omit the subscript Π in $ewt_\Pi(h)$ if it is clear from the context. In the next definition, we define potential weight functions that are prefixed/postfixed points of the operator $ewt_\Pi(h)$.

Definition 4.3 (Potential Weight Functions). A *potential upper weight function* (PUWF) is a function $h : L \times Val_{\text{p}} \to \mathbb{R}$ that has the following properties:

(C1) for all reachable states $(\ell, \mathbf{v})$ with $\ell \neq \ell_{\text{out}}$, we have $ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$;
(C2) for all reachable 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 $h : L \times Val_{\text{p}} \to \mathbb{R}$ that satisfies the conditions (C1') and (C2), for which the condition (C1') is almost the same as (C1) except for that "$ewt(h)(\ell, \mathbf{v}) \leq h(\ell, \mathbf{v})$" is replaced with "$ewt(h)(\ell, \mathbf{v}) \geq h(\ell, \mathbf{v})$".

Informally, a PUWF is a state function that satisfies the prefixed-point condition of ewt at non-terminating locations, and equals one at termination. A PLWF is defined similarly by using the postfixed-point condition. The main theorem below states that potential weight functions serve as bounds for expected weights. The proof establishes that the fixed point of ewt_Π is unique and equals ew_Π when the WPTS Π has AST, and applies Tarski's Fixed Point Theorem to use prefixed and post-fixed points to derive upper and lower bounds. See our full version [57] for the proof.

THEOREM 4.4 (FIXED-POINT APPROACH). *$[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_{\text{p}}|}) \leq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$ (resp. $[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_{\text{p}}|}) \geq h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$) for any bounded PUWF (resp. PLWF) h over Π and initial state $(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$.*

REMARK 1. *Although our fixed-point approach targets score-at-end programs, it can be extended to Bayesian programs such that in any execution of the program, only a bounded number of* `score` *statements are executed and they are all bounded. This is because one can find a bound for the overall weight of the boundedly many score functions to apply our fixed point theorem. The intuition here is that our fixed point approach can handle Bayesian programs whose multiplicative cumulative score values are always bounded by some constant. Especially, this approach can handle programs where all execution cycles are score-bounded by one.* □

4.2 The OST Approach

Below we propose a novel approach to address the NPD problem of Bayesian programs that have score statements with weights greater than 1 inside loop bodies. We refer to such programs *score-recursive*, and they have received significant attention recently (such as the phylogenetic birth model in Section 2.2). We say that a WPTS is *score-recursive* if it has a cycle of transitions on which there is a score statement with score function taking a value greater than 1. Our fixed-point approach requires bounded score, and therefore cannot handle all score-recursive programs.

We first derive a novel multiplicative variant of the classical Optional Stopping Theorem (OST) that tackles the multiplicative feature of score statements, then applies our OST variant to potential weight functions (Definition 4.3) to address the NPD problem. The classical OST requires bounded changes of the weight value, and thus cannot handle score-recursive programs.

Our OST variant applies to bounded-update score-recursive WPTS's. Informally, a WPTS has the bounded-update property if the change of values of the program variables is bounded by a global constant for every transition in the WPTS. This property is fulfilled in many realistic probabilistic models as the change of value of a variable in a single step is often bounded. Formally, a WPTS Π has the *bounded-update* property if there exists a real constant $\varkappa > 0$ such that for every reachable state $(\ell, \mathbf{v})$ and fork $F_j = \langle \ell'_j, p_j, upd_j, wt_j \rangle$ from a transition with the source location ℓ, we have that $\forall \mathbf{r} \in \mathrm{supp}\,(\mathcal{D})\ \forall x \in V_{\mathrm{p}},\ |upd_j(\mathbf{v}, \mathbf{r})(x) - \mathbf{v}(x)| \leq \varkappa$. The OST variant is given as follows.

Theorem 4.5 (OST Variant). *Let $\{X_n\}_{n=0}^{\infty}$ be a supermartingale adapted to a filtration $\mathcal{F} = \{\mathcal{F}_n\}_{n=0}^{\infty}$, and κ be a stopping time w.r.t. the filtration $\mathcal{F}$. Suppose that there exist positive real numbers b_1, b_2, c_1, c_2, c_3 such that $c_2 > c_3$ and*

(A1) *$\mathbb{P}(\kappa > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ for sufficiently large $n \in \mathbb{N}$, and*
(A2) *for all $n \in \mathbb{N}$, $|X_{n+1} - X_n| \leq b_1 \cdot n^{b_2} \cdot e^{c_3 \cdot n}$ holds almost surely.*

Then we have that $\mathbb{E}\,(|X_\kappa|) < \infty$ and $\mathbb{E}\,(X_\kappa) \leq \mathbb{E}(X_0)$.

Our OST variant relaxes the classical OST that we allow the next random variable X_{n+1} to be bounded by that of X_n with a multiplicative factor e^{c_3}. The intuition is to cancel the multiplicative factor with the exponential decrease in $\mathbb{P}(\kappa > n) \leq c_1 \cdot e^{-c_2 \cdot n}$. We note that the exponential decrease is essential to cancel multiplicative scaling in score statements, as shown in *challenges and gaps* in Section 1. The proof resembles [55, Theorem 5.2] and can be found in the full version [57].

Below we show how our OST variant can be applied to handle bounded-update score-recursive programs. Fix a bounded-update score-recursive WPTS Π in the form of (†). We reuse the expected-weight transformer defined in Definition 4.2 and potential weight functions given in Definition 4.3. The difference with our fixed-point approach is that we no longer require that the score function wt_j in (1) equals one for locations that do not lead to termination.

Theorem 4.6 (OST Approach). *Let Π be a bounded-update score-recursive WPTS. Suppose that there exist real numbers $c_1 > 0$ and $c_2 > c_3 > 0$ such that*

(E1) *$\mathbb{P}(T > n) \leq c_1 \cdot e^{-c_2 \cdot n}$ for sufficiently large $n \in \mathbb{N}$, and*
(E2) *for each score function wt in Π, we have $|wt| \leq e^{c_3}$.*

Then for any polynomial PUWF (resp. PLWF) h over Π, we have that $[\![\Pi]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \leq h(\ell_{\mathrm{init}}, \mathbf{v}_{\mathrm{init}})$ (resp. $[\![\Pi]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \geq h(\ell_{\mathrm{init}}, \mathbf{v}_{\mathrm{init}})$) for any initial state $(\ell_{\mathrm{init}}, \mathbf{v}_{\mathrm{init}})$, respectively.

Proof Sketch. For upper bounds, we 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 we construct another 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. We consider the termination time T of Π and prove that $\{Y_n\}_{n=0}^{\infty}$ satisfies the prerequisites of our OST variant (Theorem 4.5) by matching (A1) with (E1) and (A2) with (E2).

Then by Theorem 4.5, we obtain that $\mathbb{E}[Y_T] \le \mathbb{E}[Y_0]$. By (C2) in Definition 4.3, we have that $Y_T = h(\ell_T, \mathbf{v}_T) \cdot \prod_{i=0}^{T-1} W_i = \widehat{w}_T$. Thus, we have that $[\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_{\text{p}}|}) = \mathbb{E}_{\mathbf{v}_{\text{init}}}[\widehat{w}_T] = \mathbb{E}\left[\prod_{i=0}^{T-1} W_i\right] \le \mathbb{E}[Y_0] = h(\ell_{\text{init}}, \mathbf{v}_{\text{init}})$. Lower bounds are derived similarly. The full proof is put in the full version [57]. □

Remark 2. *Our OST approach handles programs with unbounded score by weights greater than* 1 *that appear inside loop bodies, but with prerequisites in Theorem 4.6 (i.e., bounded update, (E1) and (E2)) to ensure the integrability to apply our extended OST. Note that our OST approach directly handles non-score-recursive WPTS's as in such WPTS the score value inside loops is bounded by one.* □

5 ALGORITHMIC APPROACHES

In this section, we present algorithms for our fixed-point and OST approaches. Recall the task is to compute bounds for unrestricted expected weights over Bayesian probabilistic programs.

5.1 Fixed-point Approach for Score-at-end Programs

Our algorithm for the fixed-point approach solves a polynomial template h w.r.t the fixed point conditions in Theorem 4.4. The details of our algorithm (inputs and stages) are as follows.

Inputs: The inputs include a score-at-end WPTS $\Pi = (V_{\text{p}}, V_{\text{r}}, L, \ell_{\text{init}}, \ell_{\text{out}}, \mu_{\text{init}}, \mathcal{D}, \mathfrak{T})$ parsed from a Bayesian probabilistic program P written in our PPL, and extra parameters d, m, for which d is the degree of the polynomial to be solved and m is the number of partitions that divides the support $\text{supp}(\mu_{\text{init}})$ of the initial distribution uniformly into m pieces for which our algorithm solves a polynomial for each piece. Note that whether a WPTS is score-at-end or not can be done by checking the score statement and applying techniques from [9, 11] to check AST.

Stage 1: Pre-processing. Our algorithm has the following pre-processing to obtain auxiliary information for the input WPTS.

Invariant: To have an over-approximation of the set of reachable program states, our algorithm leverages external invariant generators (such as [45]) to generate numerical invariants for the WPTS. We denote the generated invariant at each location ℓ by $I(\ell)$ and treat each invariant $I(\ell)$ directly as the set of program valuations satisfying $I(\ell)$.

Bounded range: Our algorithm calculates a bounded subset B of program valuations encoded as a logical formula Φ_B (so that $B = \{\mathbf{v} \mid \mathbf{v} \models \Phi_B\}$) by heuristics. A simple heuristic here would be to run the WPTS for a small number of transitions and determine the bounded range of each program variable as the range covered by these transitions. The input WPTS will be truncated onto the bounded range B to increase the accuracy of the polynomial solving. Moreover, our algorithm calculates an extended bounded range B' of B by examining all transitions from B, i.e., for all $\mathbf{v} \in B$, all locations $\ell \neq \ell_{\text{out}}$ and all weighted forks $F = \langle \ell'_j, p_j, upd_j, wt_j \rangle$ in some transition with source location ℓ and all $\mathbf{r} \in \text{supp}(\mathcal{D})$, we have that $upd_j(\mathbf{v}, \mathbf{r}) \in B'$. Note that the exact choice of the bounded range is irrelevant to the soundness of our fixed-point approach.

Polynomial approximation: In the case that the score function g at the termination is non-polynomial, our algorithm leverages external polynomial interpolators to calculate a piecewise polynomial approximation of g over B' with an error bound $\epsilon > 0$ such that $|g(\mathbf{v}) - g'(\mathbf{v})| \le \epsilon$ for all $\mathbf{v} \in B'$. Our algorithm then replaces g with g' to avoid non-polynomial arithmetic. We prove that the replacement is sound up to the additive error ϵ in our full version [57].

Example 5.1. Recall the Pedestrian example in Section 2.1 and its WPTS Π in Fig. 4. We choose the algorithm parameters as $d = 1$ and $m = 30$ to exemplify our algorithm. We derive an invariant I simply from the loop guard so that $I(\ell_{\text{init}}) = pos \ge 0$ and $I(\ell_{\text{out}}) = pos < 0$. As the program variables $start, step$ simply receive samples, we only consider the key program variables $pos, dist$. The bounded range is denoted by $B = \{(pos, dis) \mid pos \in [0, 5], dis \in [0, 5]\}$, and the extended bounded

range is given by $B' = \{(pos, dis) \mid pos \in [-1, 6], dis \in [0, 6]\}$. Since the program is score-at-end and its score function g at the termination is non-polynomial (i.e., $g(dis) = pdf(normal(1.1, 0.1), dis)$), we choose a polynomial approximation g' of g with the error bound $\epsilon = 10^{-5}$. □

Stage 2: Partition. Our algorithm splits the set $\mathcal{V} := \mathrm{supp}\,(\mu_{\mathrm{init}})$ of initial program valuations uniformly into $m \geq 1$ disjoint partitions $\mathcal{V}_1, \dots, \mathcal{V}_m$ and construct a set $\mathcal{W} = \{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ such that each $\mathbf{v}_i \in \mathcal{V}_i$. Our approach tackles each $\mathcal{V}_i$ ($1 \leq i \leq m$) separately to obtain polynomial bounds l_i, u_i such that $l_i(\mathbf{v}) \leq [\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \leq u_i(\mathbf{v})$ for all $\mathbf{v} \in \mathcal{V}_i$. It follows that the interval bound for $[\![\Pi]\!](\mathbb{R}^{|V_{\mathrm{p}}|})$ can be derived by integrals of polynomial bounds over all $\mathcal{V}_i$'s, that is,

$$\sum_{i=1}^{m} \int_{\mathcal{V}_i} l_i(\mathbf{v}) \mu_{\mathrm{init}}(\mathrm{d}\mathbf{v}) \leq [\![\Pi]\!](\mathbb{R}^{|V_{\mathrm{p}}|}) = \int_{\mathcal{V}} [\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \cdot \mu_{\mathrm{init}}(\mathrm{d}\mathbf{v}) \leq \sum_{i=1}^{m} \int_{\mathcal{V}_i} u_i(\mathbf{v}) \mu_{\mathrm{init}}(\mathrm{d}\mathbf{v}). \qquad (\clubsuit)$$

Example 5.2. We obtain the set $\mathcal{V} = \{(pos, dis) \mid pos \in [0, 3], dis = 0\}$, and partition $\mathcal{V}$ uniformly into $m = 30$ disjoint subsets on the dimension pos, i.e., $\mathcal{V}_1 = \{(pos, dis) \mid pos \in [0, 0.1], dis = 0\}, \dots, \mathcal{V}_{30} = \{(pos, dis) \mid pos \in [2.9, 3], dis = 0\}$. We calculate the midpoints of the dimension pos for all $\mathcal{V}_i$'s, and construct the set $\mathcal{W} = \{(0.05, 0), (0.15, 0), \dots, (2.95, 0)\}$. □

Stage 3: Truncation. Our algorithm performs a truncation operation to improve the accuracy. Intuitively, the truncation operation restricts the program values into the bounded range B calculated from the pre-processing, and over-approximates the expected weight outside the bounded range by a truncation approximation. A *truncation approximation* is a function $\mathcal{M} : \mathbb{R}^{|V_{\mathrm{p}}|} \to [0, \infty)$ such that each $\mathcal{M}(\mathbf{v})$ is intended to be an over- or under-approximation of the expected weight $[\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_{\mathrm{p}}|})$ outside the bounded range B. The truncation operation is given as follows.

Definition 5.3 (Truncation Operation). Given the bounded range B and a truncation approximation $\mathcal{M}$,the *truncated* WPTS $\Pi_{B,\mathcal{M}}$ 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 termination location and the transition relation $\mathfrak{T}_{\mathcal{B},\mathcal{M}}$ is given by

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

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

Informally, we obtain the truncated WPTS by restraining each transition to the bounded range B and redirecting all transitions jumping out of the bounded range but not into the location ℓ_{out} to the fresh termination location $\sharp$. We add the self-loop $\langle \sharp, \mathbf{true}, F_\sharp \rangle$ to ensure determinism and totality.

We call a truncation approximation $\mathcal{M}$ *upper* (reps. *lower*) if for all reachable state $(\ell, \mathbf{v})$ such that $\mathbf{v} \notin B$, it holds that $[\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \leq \mathcal{M}(\mathbf{v})$ (resp. $[\![\Pi]\!]_{\mathbf{v}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \geq \mathcal{M}(\mathbf{v})$). We prove the correctness that $[\![\Pi]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \leq [\![\Pi_{B,\mathcal{M}}]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|})$ for all initial program valuations $\mathbf{v}_{\mathrm{init}}$ if $\mathcal{M}$ is an upper truncation approximation, and $[\![\Pi]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|}) \geq [\![\Pi_{\mathcal{B},\mathcal{M}}]\!]_{\mathbf{v}_{\mathrm{init}}}(\mathbb{R}^{|V_{\mathrm{p}}|})$ if $\mathcal{M}$ is lower. The detailed theorem statement and proof is given in Wang et al. [57].

To calculate truncation approximations, our algorithm takes a bound M for the score function and has 0 and M as the trivial lower and upper truncation approximations. To sharpen the truncation approximations, our algorithm either utilizes the monotonicity of program variables to tighten the estimation of the values of the score function at the termination, or derives polynomial truncation approximations by applying polynomial solving to our fixed-point approach without truncation.

Example 5.4. We choose the bounded range $[0,5]\times[0,5]$ that specifies $[0,5]$ for both the variable pos, dis. The truncation approximations are set to $\mathcal{M}_{\text{up}} = 2.1\times 10^{-330}$ and $\mathcal{M}_{\text{low}} = 0$. The values $2.1\times 10^{-330}, 0$ are calculated by the monotonicity of the density function $pdf(normal(1.1, 0.1), dis)$ and the values of dis at 5, 6, respectively. We obtain two truncated WPTS's $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$ and $\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}$. □

Stage 4: Polynomial Solving. Our algorithm establishes d-degree polynomial templates for $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$ and $\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}$, and derives polynomial upper and lower bounds for the expected weights $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}_i}(\mathbb{R}^{|V_{\text{p}}|})$ and $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!]_{\mathbf{v}_i}(\mathbb{R}^{|V_{\text{p}}|})$ for each initial program valuation $\mathbf{v}_i \in \mathcal{V}_i$ in $\mathcal{W}$ by solving the templates w.r.t the PUWF and PLWF constraints (i.e., (C1), (C2), (C1') from Definition 4.3), respectively. The correctness of this stage follows from Theorem 4.4 and that polynomials are bounded over a bounded range. Below we present the details in **Step A1** – **A3**. We focus on $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$ and polynomial upper bounds. The case of lower bounds considers $\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}$ and is similar.

Step A1. In this step, for each location $\ell \notin \{\ell_{\text{out}}, \sharp\}$, our algorithm sets up a d-degree polynomial template h_ℓ over the program variables V_{p}. Each template is a summation of all monomials in the program variables of degree no more than d, for which each monomial is multiplied with an unknown coefficient. For $\ell \in \{\ell_{\text{out}}, \sharp\}$, our algorithm assumes $h_\ell \equiv 1$.

Step A2. In this step, our algorithm establishes constraints for the templates h_ℓ's from (C1), (C1') in Definition 4.3 (as (C2) is satisfied directly by the form of h_ℓ). For every location $\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$, we have the following relaxed constraints of (C1) to synthesize a PUWF over $\Pi_{\mathcal{B},M_{up}}$:

(D1) For every program valuation $\mathbf{v} \in I(\ell) \cap B$, we have that $ewt(h)(\ell, \mathbf{v}) \le h(\ell, \mathbf{v})$.
(D2) For every program valuation $\mathbf{v} \in I(\ell) \cap (B' \setminus B)$, we have that $\mathcal{M}_{\text{up}}(\mathbf{v}) \le h(\ell, \mathbf{v})$.

For lower bounds over $\Pi_{\mathcal{B},M_{low}}$, our algorithms have the relaxed PLWF constraints (D1') and (D2') which are obtained from (D1) and resp. (D2) by replacing "$ewt(h)(\ell, \mathbf{v}) \le h(\ell, \mathbf{v})$" with "$ewt(h)(\ell, \mathbf{v}) \ge h(\ell, \mathbf{v})$" in (D1) and resp. "$\mathcal{M}_{\text{up}}(\mathbf{v}) \le h(\ell, \mathbf{v})$" with "$\mathcal{M}_{\text{low}}(\mathbf{v}) \ge h(\ell, \mathbf{v})$" in (D2), respectively. We have that (D1) and (D2) together ensure (C1) since $\mathcal{M}_{\text{up}}(\mathbf{v}) \le h(\ell, \mathbf{v})$ implies that $ewt(h)(\ell, \mathbf{v}) \le h(\ell, \mathbf{v})$ for every location $\ell \in L \setminus \{\ell_{\text{out}}, \sharp\}$ and program valuation $\mathbf{v} \in I(\ell) \cap (B' \setminus B)$. The same holds for (D1') and (D2').

Note that in (D1), the calculation of $ewt(h)(\ell, \mathbf{v})$ has the piecewise nature that different sampling valuations $\mathbf{r}$ may cause the next program valuation to be within or outside the bounded range, and to satisfy or violate the guards of the transitions in the WPTS. In our algorithm, we have a refined treatment for (D1) that enumerates all possible situations for a sampling valuation $\mathbf{r}$ that satisfy different guards in the calculation of $ewt(h)(\ell, \mathbf{v})$, for which we use an SMT solver (e.g., Z3 [14]) to compute the situations. As for (D2), we use (D2) to avoid handling the piecewise feature in the computation of $ewt(h)(\ell, \mathbf{v})$ from within/outside the bounded range (i.e., the computation is a direct computation over a single-piece polynomial), so that the amount of computation of $ewt(h)(\ell, \mathbf{v})$ is reduced by ignoring the piecewise feature. The use of (D2) to reduce the computation follows from the extended bounded range in the pre-processing. The same holds for (D1') and (D2').

Example 5.5. Recall Examples 5.1, 5.2 and 5.4, we set up the 1-degree template $h_{\ell_{\text{init}}}(pos, dis) = a_1 \cdot pos + a_2 \cdot dis + a_3$ withe unknown coefficients $a_1, a_2, a_3 \in \mathbb{R}$ and $h_\ell(pos, dis) = 1$ for $\ell = \{\ell_{\text{out}}, \sharp\}$. Then the bounded range $\mathbb{D}_1 = I(\ell_{\text{init}}) \cap B$ in (D1) is defined such that $pos \in [0,5] \wedge dis \in [0,5]$. Consider to derive upper bounds from $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$. We make a fine-grained treatment for (D1) by splitting the range $\mathbb{D}_1$ and enumerating all possible situations for the sampling valuation **sample uniform**$(0,1)$ that the next program valuation satisfies or violates the loop guard "$pos \ge 0$". In detail, $\mathbb{D}_1$ is split into $\mathbb{D}_{11} = \{(pos, dis) \mid pos \in [0,1), dis \in [0,5]\}$ and $\mathbb{D}_{12} = \{(pos, dis) \mid pos \in (1,5], dis \in [0,5]\}$, where $\mathbb{D}_{11}$ stands for the situation that with different sampling valuations the next program valuation may satisfy the loop guard (so that the next location is ℓ_{init}) or violate the loop guard (so

that the next location is directed to ℓ_{out}), and $\mathbb{D}_{12}$ stands for the situation that the next program valuation will definitely satisfy the loop guard and the next location is ℓ_{init}.

We first show the PUWF constraints for $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$. Recall that the program has two probabilistic branches with probability 0.5. When the current program valuation is in $\mathbb{D}_{11}$, we observe that (a) if the loop takes the branch $pos := pos + step$, then the next value of pos remains to be non-negative and the loop continues, and (b) if the loop takes the branch $pos := pos - step$, then the next value of pos either satisfies or violates the loop guard, depending on the exact value of $step \in [0, 1]$.

Then we have the constraint (D1.1) over $\mathbb{D}_{11}$ that has expectation over a piecewise function on $step$ derived from whether the loop terminates in the next iteration or not, as follows:

(D1.1) $\forall pos, dis.\ pos \in [0, 1) \wedge dis \in [0, 5] \Rightarrow$

$$0.5 \cdot \mathbb{E}_{step}\left[[pos - step \geq 0] \cdot h_{\ell_{\text{init}}}(pos - step, dis + step) + [pos - step < 0] \cdot g'(dis + step)\right]$$
$$+0.5 \cdot \mathbb{E}_{step}\left[h_{\ell_{\text{init}}}(pos + step, dis + step)\right] \leq h_{\ell_{\text{init}}}(pos, dis).$$

When the current program valuation is in $\mathbb{D}_{12}$, we observe that the next program valuation is guaranteed to satisfy the loop guard, and hence we have the constraint (D1.2) over $\mathbb{D}_{12}$ as follows:

(D1.2) $\forall pos, dis.\ pos \in (1, 5] \wedge dis \in [0, 5] \Rightarrow$

$$0.5 \cdot \mathbb{E}_{step}\left[h_{\ell_{\text{init}}}(pos - step, dis + step)\right] + 0.5 \cdot \mathbb{E}_{step}\left[h_{\ell_{\text{init}}}(pos + step, dis + step)\right] \leq h_{\ell_{\text{init}}}(pos, dis).$$

The range $\mathbb{D}_2 = I(\ell_{\text{init}}) \cap (B' \setminus B)$ in (D2) is represented by the disjunctive formula $\Phi := (pos \in [0, 6] \wedge dis \in [5, 6]) \vee (pos \in [5, 6] \wedge dis \in [0, 6])$. For the program valuation in $\mathbb{D}_2$, its next location is $\ell_\sharp$. Recall the truncation approximation $\mathcal{M}_{\text{up}} = 2.1 \times 10^{-330}$. From (D2), we have two constraints from the disjunctive clauses in Φ as follows:

(D2.1) $\forall pos, dis.\ pos \in [0, 6] \wedge dis \in [5, 6] \Rightarrow 2.1 \times 10^{-330} \leq h_{\ell_{\text{init}}}(pos, dis)$.
(D2.2) $\forall pos, dis.\ pos \in [5, 6] \wedge dis \in [0, 6] \Rightarrow 2.1 \times 10^{-330} \leq h_{\ell_{\text{init}}}(pos, dis)$.

For $\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}$, the PLWF constraints (D1'.1) to (D2'.2) are obtained from the PUWF constraints above by replacing "$\leq$" with "$\geq$" and 2.1×10^{-330} with 0. □

Step A3. In this step, for every initial program valuation $\mathbf{v}_i \in \mathcal{V}_i$ in $\mathcal{W}$ where $\mathcal{V}_i$'s and $\mathcal{W}$ are obtained from **Stage 2**, 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 Putinar's Positivstellensatz [38] or Handelman's Theorem [23]. In detail, our algorithm minimizes the objective function $h_{\ell_{\text{init}}}(\mathbf{v}_i)$ for each $\mathbf{v}_i$ in $\mathcal{W}$ which subjects to the PUWF constraints from the previous step to derive polynomial upper bounds over $\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}$. For lower bounds, we use PLWF and consider to maximize the objective function.

Note that the PUWF or PLWF constraints from **Step A2** can be represented as a conjunction of formulas in the form $\forall \mathbf{v} \in \mathcal{P}.(\mathbf{g}(\mathbf{v}) \geq 0)$ where the set $\mathcal{P}$ is defined by a conjunction of polynomial inequalities in the program variables and $\mathbf{g}$ is a polynomial over V_{p} whose coefficients are affine expressions in the unknown coefficients from the templates, and such formulas can be guaranteed by the sound forms of Putinar's Positivstellensatz and Handelman's Theorem. 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 is restricted to the affine case (i.e., every condition and assignment in the WPTS or the original program is affine) and leads to linear constraints and can be solved by linear programming (LP). The details on the application of Putinar's Positivstellensatz and Handelman's Theorem are given in the full version [57].

Example 5.6. Recall Examples 5.2, 5.4 and 5.5, we have that $\mathcal{V}_1 = \{(pos, dis) \mid pos \in [0, 0.1], dis = 0\}, \ldots, \mathcal{V}_{30} = \{(pos, dis) \mid pos \in [2.9, 3], dis = 0\}$ and the set $\mathcal{W} = \{(0.05, 0), (0.15, 0), \ldots, (2.95, 0)\}$.

Pick a point $\mathbf{v}_1 = (0.05, 0) \in \mathcal{V}_1$ from $\mathcal{W}$, and to synthesize the upper bound for $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}_1}(\mathbb{R}^{|V_\text{p}|})$, we solve the following optimization problem whose objective function is $h_{\ell_{\text{init}}}(0.05, 0)$, i.e.,

$$\textbf{Min } 0.05 \cdot a_1 + a_3 \qquad \textbf{s.t. } \text{constraints (D1.1)–(D2.2)}$$

Dually, the lower bound for $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!]_{\mathbf{v}_1}(\mathbb{R}^{|V_\text{p}|})$ is solved by **Max** $0.05 \cdot a_1 + a_3$ **s.t.** (D1'.1)–(D2'.2). Although the constraints are universally quantified, the universal quantifiers can be soundly (but not completely) removed and relaxed into semidefinite constraints over the unknown coefficients a_i's by applying Putinar's Positivstellensatz, where we over-approximate all strict inequalities (e.g., "$<$") by non-strict ones (e.g., "$\leq$"). Then we call an SDP solver to solve the two optimization problems and find the solutions of a_i's, which will generate two polynomial bound functions Up_1, Lw_1 such that $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}}(\mathbb{R}^{|V_\text{p}|}) \leq Up_1(\mathbf{v}) + \epsilon$ and $Lw_1(\mathbf{v}) - \epsilon \leq [\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!]_{\mathbf{v}}(\mathbb{R}^{|V_\text{p}|})$ for all program valuation $\mathbf{v} \in \mathcal{V}_1$ with the polynomial approximation error $\epsilon = 10^{-5}$. □

Stage 5: Integration. As a consequence of **Stage 4**, our algorithm obtains polynomial upper bounds $Upper = \{Up_1, \dots, Up_m\}$ for the expected weights $\{[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}_1}(\mathbb{R}^{|V_\text{p}|}), \dots, [\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}_m}(\mathbb{R}^{|V_\text{p}|})\}$ w.r.t. the m partitions $\mathcal{V}_i, \dots, \mathcal{V}_m$. Then our algorithm integrates these polynomial upper bounds to derive the upper bound for $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!](\mathbb{R}^{|V_\text{p}|})$. In detail, we have that

$$[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!](\mathbb{R}^{|V_\text{p}|}) \leq \textstyle\sum_{i=1}^{m} \int_{\mathcal{V}_i} Up_i(\mathbf{v}) \cdot \frac{1}{m} \mathrm{d}\mathbf{v} =: u \tag{2}$$

Similarly, the lower bound for $[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!](\mathbb{R}^{|V_\text{p}|})$ is given by

$$l := \textstyle\sum_{i=1}^{m} \int_{\mathcal{V}_i} Lw_i(\mathbf{v}) \cdot \frac{1}{m} \mathrm{d}\mathbf{v} \leq [\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!](\mathbb{R}^{|V_\text{p}|}) \tag{3}$$

with the polynomial lower bounds $Lower = \{Lw_1, \dots, Lw_m\}$ generated in **Stage 4**. Note that $[l, u]$ is also the interval bound for the original $[\![\Pi]\!](\mathbb{R}^{|V_\text{p}|})$ as $[\![\Pi_{B,\mathcal{M}_{\text{low}}}]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_\text{p}|}) \leq [\![\Pi]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_\text{p}|}) \leq [\![\Pi_{B,\mathcal{M}_{\text{up}}}]\!]_{\mathbf{v}_{\text{init}}}(\mathbb{R}^{|V_\text{p}|})$ for all initial program valuations $\mathbf{v}_{\text{init}}$. The details are in the full version [57].

If the score function g at the termination is non-polynomial, our algorithm integrates the polynomial approximation error $\varsigma = \text{volume}(\mathcal{V}) \cdot \epsilon$ caused by its polynomial approximation g' to the two bounds, i.e, $l' = l - \varsigma$ and $u' = u + \varsigma$, where volume$(\mathcal{V})$ is the volume of $\mathcal{V}$ and ϵ is the error bound. In practice, to ensure the tightness of the interval bounds, we can control the amount of ϵ so that the approximation error ς is at least one magnitude smaller than the values of l, u.

The correctness of our algorithm is stated as follows. The pseudo code is in Algorithm 1.

Theorem 5.7 (Soundness). *If our algorithm finds valid solutions for the unknown coefficients in the polynomial templates, then it returns correct interval bounds for* $[\![\Pi]\!](\mathbb{R}^{|V_\text{p}|})$.

Proof. Let h_ℓ's ($\ell \in L$) be the solved polynomial templates. Since the extended range B' is bounded, we can find a bound M such that all the values these h_ℓ's take fall in $[-M, M]$ within B'. By applying Theorem 4.4 to the truncated WPTS and Eq. (♣), we obtain the desired result. □

5.2 OST Approach for Bounded-Update Score-Recursive Programs

The algorithm for our OST approach over bounded-update score-recursive programs follows similar stages as for our fixed-point approach. The main difference lies at the pre-processing and truncation stages. In the pre-processing stage, the difference includes the following: (i) To apply Theorem 4.6, our approach checks the prerequisites (E1) and (E2) by external approaches. For example, the condition (E1) can be checked by existing approaches in concentration and tail bound analysis [12, 52], and the condition (E2) by SMT solvers. (ii) Our current algorithmic approach does not tackle score statements with non-polynomial score functions inside a loop body. In general, non-polynomial score functions inside the loop body can also be handled by piecewise polynomial

Algorithm 1: The Algorithm for Fixed-Point Approach

Input : A score-at-end WPTS Π, Parameters d, m
Output : The upper bound u and the lower bound l for $[\![\Pi]\!](\mathbb{R}^{|V_p|})$

Pre-processing: (1) Invariant generation I. (2) Bounded range B and extended bounded range B'. (3) Polynomial approximation for non-polynomial score function at termination.
Partition: Splits the set $\mathcal{V} := \text{supp}\,(\mu_{\text{init}})$ into $m \geq 1$ disjoint partitions $\mathcal{V}_1, \ldots, \mathcal{V}_m$ and construct a set $\mathcal{W} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ **s.t.** $\mathbf{v}_i \in \mathcal{V}_i$.
Truncation:

(1) Restricts the program values into the bounded range B;
(2) Over (resp. under) -approximate the expected weight outside the bounded range B with truncation approximation $\mathcal{M}_{\text{up}}$ (resp. $\mathcal{M}_{\text{low}}$);
(3) Construct the truncated WPTS $\Pi_{B,\mathcal{M}_{\text{up}}}$ (resp. $\Pi_{B,\mathcal{M}_{\text{low}}}$) for upper (resp. lower) bounds.

Polynomial Solving: Establish d-degree template h and constraints over h,
for *each* ℓ **do**
  Establish constraints (D1) (within truncate range $I(\ell) \cap B$) and (D2) (outside truncated range $I(\ell) \cap (B' \setminus B)$) ; /* upper bound */
  Establish constraints (D1') (within truncate range $I(\ell) \cap B$) and (D2') (outside truncated range $I(\ell) \cap (B' \setminus B)$) ; /* lower bound */
end
Optimize $h_{\ell_{\text{init}}}(\mathbf{v}_i)$ for each $\mathbf{v}_i$ in $\mathcal{W}$ with the constraints above, and produce polynomial upper and lower bounds $Upper = \{Up_1, \ldots, Up_m\}$, $Lower = \{Lw_1, \ldots, Lw_m\}$.
Integration: Integrate the upper and lower bounds,
$[\![\Pi_{\mathcal{B},\mathcal{M}_{\text{up}}}]\!](\mathbb{R}^{|V_p|}) \leq \sum_{i=1}^{m} \int_{\mathcal{V}_i} Up_i(\mathbf{v})d\mathbf{v} =: u, \quad [\![\Pi_{\mathcal{B},\mathcal{M}_{\text{low}}}]\!](\mathbb{R}^{|V_p|}) \geq \sum_{i=1}^{m} \int_{\mathcal{V}_i} Lw_i(\mathbf{v})d\mathbf{v} =: l$

approximation. A direct way is to provide upper and lower polynomial bounds for the original non-polynomial function, which avoids the calculation of the propagation of approximation errors along loop iterations. The same applies to sampling variables inside loop bodies with non-polynomial probability density (or mass) functions.

In the truncation, the difference is that our OST approach derives polynomial truncation approximations by applying polynomial solving to our OST approach without truncation. The soundness of the algorithm (producing correct bounds for expected weights) follows from Theorem 4.6.

6 EXPERIMENTAL RESULTS

In this section, we present the experimental evaluation of our approach over a variety of benchmarks. First, we show that our approach can handle novel examples that cannot be addressed by existing tools such as [18, 19, 25, 35]. Then we compare our approach with the state-of-the-art tool GuBPI [4] over score-at-end Bayesian programs. (Note that GuBPI could only handle score-at-end programs.) Finally, even though the problem of path probability estimations is not the focus of our work, we demonstrate that our approach works well for this problem, for which we also compare the performance of our approach with GuBPI. We implement our algorithms in Matlab. All results are obtained on an Intel Core i7 (2.3 GHz) machine with 16 GB of memory, running MS Windows 10.

6.1 Experimental Setup

Since our approach is completely orthogonal to GuBPI, we conservatively have the experimental setup in order to minimize the advantage of the external inputs to our algorithms in the comparison.
Inputs. All benchmarks are in the form of a single while loop. We set two locations for each benchmark, i.e., ℓ_{init} for the entry of the while loop and ℓ_{out} for termination. We denote the loop

guard by ϕ. We implement a parser from probabilistic programs into WPTS's in F#. We conduct our experiments with various parameters combination of d, m and present the results in Tables 1 to 3.

Stage 1: Pre-processing. The pre-processing is illustrated as follows.

Invariant: We take the conservative setting that simply derives the invariant from the loop guard ϕ so that $I(\ell_{\text{init}}) = \phi$ and $I(\ell_{\text{out}}) = \neg\phi$.

Bounded range: To minimize the advantage of the choice of the bounded range to our approach, we have a conservative setting that has the bounded range to cover a majority part of program executions. Having a smaller bounded range would result in more accurate results, as polynomial solving is more accurate over a smaller bounded range. In detail, for each program variable x, we have B''_x as the interval that is the projection of the support of the initial distribution onto the variable x. Then we choose a large deviation δ for all B''_x ($x \in V_{\text{p}}$) to get intermediate intervals IB_x, i.e., each IB_x is given by $IB_x := [\zeta_1 - \delta, \zeta_2 + \delta]$ where $[\zeta_1, \zeta_2] = B''_x$. The final bounded range B is given as the intersection of the Cartesian product of all IB_x's and the loop guard.

Polynomial approximation: In the case that the input is a score-at-end Bayesian program and the score function is non-polynomial, we use the polynomial interpolator in Matlab to obtain a piecewise polynomial approximation for the score function. We do not have polynomial approximations in our OST approach since the benchmarks considered do not have non-polynomial score functions.

Stage 2: Partition. We partition the set of initial valuations uniformly into m disjoint subsets $\mathcal{V}_1, \ldots, \mathcal{V}_m$, and choose the midpoints v_i of each partition $\mathcal{V}_i$.

Stage 3: Truncation. Our approach calculates the truncation approximations as described in Section 5. For score-at-end programs, our approach either gets them by the direct bounds from the score function, further improves them by heuristics such as monotonicity, or derives polynomial truncation approximations by applying polynomial solving to our fixed-point approach without truncation. For score-recursive programs, our approach gets the truncation approximations by directly applying polynomial solving under our OST variant without truncation.

Stage 4: Polynomial solving. In applying the Positivstellensatz's, we use the LP solver in Matlab (resp. Mosek [1]) for solving linear (resp. semidefinite) programming, respectively.

6.2 Results

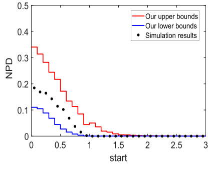
We focus on unbounded while loops as they distinguish our approach with previous approaches most significantly, and compare with the most relevant tool GuBPI [4]. To compare our approach with GuBPI fairly, our measurement of the time cost of our approach includes the time taken to generate all the extra inputs.

NPD - Novel Examples. We consider 10 novel examples adapted from the literature, where all 7 examples with prefix "Pd" or "RdWalk" are from [4], the two "RACE" examples are from [52], and the last example is from statistical phylogenetics [41] (see also Section 2). 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 4.6); other examples are score-at-end probabilistic while loops with unsupported types of scoring by previous tools (e.g., polynomial scoring **score**(y) where y is a single-variable polynomial). Therefore, no existing 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 parameters of each example used in our approach (i.e., the degree d of the polynomial template, the number m of partitions 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,

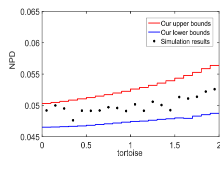
Table 1. Results for Novel Examples

Benchmark	Parameters	Solver	Upper Time (s)	Lower Time (s)
PD(v1)	$d = 6$, $m = 60$,$pos, dis \in [0, 5]$	SDP	54.65	52.39
RACE(v1)	$d = 6$,$m = 40$,$h, t \in [0, 5]$	LP	87.43	86.27
RACE(v2)*	$d = 6$,$m = 40$, $h, t \in [0, 5]$	LP	81.19	81.18
RDWALK(v1)	$d = 6$,$m = 60$, $x, y \in [0, 5]$	LP	46.65	47.72
RDWALK(v2)	$d = 6$,$m = 60$, $x, y \in [0, 5]$	LP	97.45	103.65
RDWALK(v3)	$d = 6$,$m = 60$, $x, y \in [0, 5]$	LP	48.61	49.39
RDWALK(v4)	$d = 6$, $m = 60$,$x, y \in [0, 5]$	LP	98.75	98.21
PDMB(v3)	$d = 4$, $m = 60$,$pos, dis \in [0, 5]$	LP	15.26	14.57
PDMB(v4)	$d = 4$,$m = 60$, $pos, dis \in [0, 5]$	LP	16.07	16.12
BIRTH*	$d = 6$, $m = 40$, $lambda \in [0, 2]$, $time \in [0, 10]$	LP	14.72	16.66

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

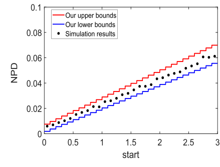


(a) PD(v1)



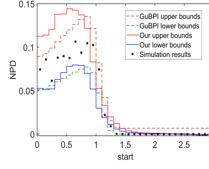
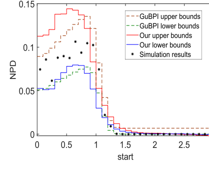
(b) RACE(v2)

(c) RDWALK(v1)

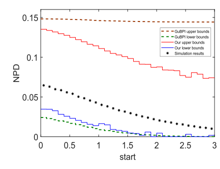
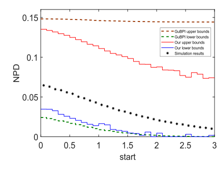


(d) PDMB(v3)

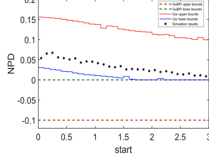
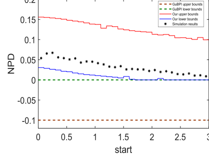
Fig. 6. NPD Bounds of Novel Examples

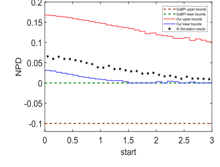
(a) PD

(b) PDLD

(c) PDBETA(v1)



(d) PDBETA(v2)

Fig. 7. NPD Bounds of Comparison

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

that is, most examples can obtain tight bounds within 100 seconds, and the simulation results by Pyro [5] (10^6 samples per case) match our derived bounds. We display part of the comparison in Fig. 6, see our full version [57] for other figures.

NPD - Comparision with GuBPI [4]. Since the parameters used in GuBPI and our approach are completely different, it is infeasible to compare the two approaches directly. Instead, we choose the parameters to our algorithms that can achieve at least comparable results with GuBPI. The main parameters are shown in Table 2. We consider the Pedestrian example "PD" from [4] (see also Section 2.1), and its variants. For the variants of "PD", we enlarged the standard deviation of the observed normal distribution to be 5 **in all** 6 **variants** whose prefix name are "PD"; for the four "PDBETA" examples, we also adjust the original uniform sampling **uniform**(0, 1) in the loop body by

Table 2. Comparison with GuBPI

Benchmark	Our Tool				GuBPI	
	Parameters	Solver	Time (s)	#	Time (s)	#
PD	$d = 10$, $m = 60$, $pos, dis \in [0, 5]$	SDP	3176.685	●	5266.063	●
PDLD	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	41.99	●	648.151	●
PDBETA(v1)	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	99.86	●	645.055	○
PDBETA(v2)	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	228.43	●	653.237	○
PDBETA(v3)	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	101.36	●	657.645	○
PDBETA(v4)	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	208.86	●	686.207	○
PDMB(v5)	$d = 6$, $m = 60$, $pos, dis \in [0, 5]$	LP	88.41	●	391.772	●
PARA-RECUR	$d = 8$, $m = 60$, $p \in [0, 1]$	LP	36.61	●	253.728	●

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

different beta distributions. The main purpose to introduce these variants is to test the robustness of our approach. The last example is from [18].

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. 7 (see Wang et al. [57] for other figures). Our runtime is up to 15 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 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.[5] Note that the simulation results near 1 deviates largely from our bounds, for which a possible reason is that PD is a difficult example whose simulation results can have high variances. For all 6 variants of "PD", 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 found that GuBPI produced zero-valued unnormalised lower bounds, and its results w.r.t. NPD are trivial, i.e., $[0, \infty]$. However, we can produce non-trivial results and our runtime is at least 2 times faster than GuBPI. We believe that the main reason why our approach outperforms GuBPI is that GuBPI has widening that may lose precision, while our approach uses polynomial solving with truncation to achieve better precision.

Path Probability Estimation. We consider five recursive examples in [4, 44], which were also cited from the PSI repository [18]. Since all five examples are non-paramteric and with unbounded numbers of loop iterations, PSI cannot handle them as mentioned in [4]. We estimated the path probability of certain events, i.e., queries over program variables, and thus constructed a new bounded range for each query Q by the conjunction of the corresponding B (see **Stage 3** in Section 6.1) and query Q. The results are reported in Table 3 where the second column corresponds to different queries, and the parameters in the third column are the degree d, the number m and the bounded range, respectively. 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 normalising constant should be exactly one. However,

[5] 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.

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, 1, [0, 30], [0, 4]	15.062	[0.9698, 1.0000]	38.834	[0.7381, 1.0000]	0.9938
	Q2	6, 1, [0, 40], [0, 4]	16.321	[0.9985, 1.0000]	37.651	[0.7381, 1.0000]	0.9993
AddUni(L)	Q1	6, 1, [0, 10], [0, 1]	8.85	[0.9940, 1.0000]	21.064	[0.9375, 1.0000]	0.9991
	Q2	6, 1, [0, 15], [0, 1]	8.80	[0.9995, 1.0000]	14.941	[0.9375, 1.0000]	0.9999
RdBox	Q1	4, 1, [−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, 1, [20, ∞], [0, 10]	33.17	[0.8251, 0.9351]	229.623	[0.5768, 0.6374]	0.9098
	Q2	6, 1, [20, ∞], [0, 20]	38.373	[0.9405, 1.0000]	224.504	[0.5768, 0.6375]	0.9645
GWalk **	Q1	8, 1, [1, ∞], [0, 0.1]	7.255	[0.0023, 0.0023]	33.246	[0.0023, 0.0024]	0.0023
	Q2	8, 1, [1, ∞], [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 unnormalised distributions for this example (the NPD can be derived in the same manner above).

the upper bound of the normailising constant obtained by GuBPI is smaller than 1 (i.e., 0.6981). A simulation using 10^6 samples yielded the results that fall within our bounds but violate those by GuBPI. Thus, GuBPI possibly omitted some valid program runs of this example and produces wrong results. All our results match the simulation (10^6 samples per case).

7 RELATED WORKS

Approximate methods. Statistical approaches such as MCMC [17, 42], variational inference [6] and Hausdorff measure [39] cannot provide formal guarantees for the outcomes w.r.t. posterior distributions in a finite time limit. The work [24] proposes a novel sampling framework by combining control-data separation and logical condition propagation, which is actually approximate methods. In contrast, our approach has formal guarantees on the derived NPD bounds.

Guaranteed NPD inference. Most works on guaranteed inference (such as (λ)PSI [18, 19], AQUA [25], Hakaru [35], SPPL [43], etc.) 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 unbounded while-loops/recursion. The most relevant work [4] infers the NPD bounds by recursion unrolling. Our approach circumvents the path explosion problem from recursion unrolling by polynomial solving, and outperforms this approach over various benchmarks. Several recent works [27, 59] consider efficient posterior inference in Bayesian Probabilistic Programming. Note that PGF-based inference methods [27, 59] can only be applied in discrete probabilistic programs. [27] presents an extended denotational semantics for discrete Bayesian probabilistic while loops and performs exact inference for loop-free programs and a syntactic class of AST programs, while [59] calculates the PGFs in Bayesian probabilistic programs efficiently using approximation and automatic differentiation, but cannot handle while loop. Our methods can handle continuous Bayesian probabilistic programs and derive tight bounds for posterior distributions.

Static analysis of probabilistic programs. In recent years, there have been an abundance of works on the static analysis of probabilistic programs. Most of them address fundamental aspects such as termination [9, 11, 16], sensitivity [2, 54], expectation [3, 36, 55], tail bounds [28, 51, 53], assertion probability [44, 52], etc. Compared with these results, we have that: (a) Our work focuses on normalised posterior distribution in Bayesian probabilistic programming, and hence is an orthogonal objective. (b) Although our algorithms follow the previous works on polynomial template solving [9, 10, 12, 55], we have a truncation operation to increase the accuracy which to our best knowledge is novel. (c) Our approach extends the classical OST as the previous works [51, 55] do, but we consider a multiplicative variant, while the work [55] considers only additive variants.

DATA-AVAILABILITY STATEMENT

The artifact of this paper is available at the link [56].

ACKNOWLEDGMENTS

We thank the anonymous reviewers of PLDI 2024 for their valuable comments and helpful suggestions. This work was supported by the National Research Foundation, Singapore, under its RSS Scheme (NRF-RSS2022-009), the Engineering and Physical Sciences Research Council (EP/T006579/1), and the National Natural Science Foundation of China (NSFC) under Grant No. 62172271.

REFERENCES

[1] MOSEK ApS. 2022. *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.* http://docs.mosek.com/10.0/toolbox/index.html

[2] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. Proving expected sensitivity of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 57:1–57:29. https://doi.org/10.1145/3158145

[3] Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13994)*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer, 410–429. https://doi.org/10.1007/978-3-031-30820-8_25

[4] Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. 2022. Guaranteed bounds for posterior inference in universal probabilistic programming. In *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, Ranjit Jhala and Isil Dillig (Eds.). ACM, 536–551. https://doi.org/10.1145/3519939.3523721

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

[6] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.

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

[8] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV 2013*. 511–526.

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

[10] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *CAV 2016*. 3–22.

[11] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 327–342. https://doi.org/10.1145/2837614.2837639

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

[13] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017. Stochastic invariants for probabilistic termination. In *POPL 2017*. 145–160.

[14] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[15] Joseph L Doob. 1971. What is a Martingale? *The American Mathematical Monthly* 78, 5 (1971), 451–463.

[16] Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *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)*, Constantin Enea and Ruzica Piskac (Eds.). Springer, 468–490. https://doi.org/10.1007/978-3-030-11245-5_22

[17] Dani Gamerman and Hedibert F Lopes. 2006. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference.* CRC press.

[18] Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9779)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

[19] Timon Gehr, Samuel Steffen, and Martin T. Vechev. 2020. λPSI: exact inference for higher-order probabilistic programs. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 883–897. https://doi.org/10.1145/3385412.3386006

[20] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Kallista A. Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, David A. McAllester and Petri Myllymäki (Eds.). AUAI Press, 220–229.

[21] Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org.

[22] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. 2014. Probabilistic programming. In *Future of Software Engineering Proceedings.* 167–181.

[23] David Handelman. 1988. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.* 132, 1 (1988), 35–62.

[24] Ichiro Hasuo, Yuichiro Oyabu, Clovis Eberhart, Kohei Suenaga, Kenta Cho, and Shin-ya Katsumata. 2021. Control-Data Separation and Logical Condition Propagation for Efficient Inference on Probabilistic Programs. *CoRR* abs/2101.01502 (2021). arXiv:2101.01502 https://arxiv.org/abs/2101.01502

[25] Zixin Huang, Saikat Dutta, and Sasa Misailovic. 2021. AQUA: Automated Quantized Inference for Probabilistic Programs. In *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)*, Zhe Hou and Vijay Ganesh (Eds.). Springer, 229–246. https://doi.org/10.1007/978-3-030-88885-5_16

[26] H Jeffreys. 1988. "Weierstrass's theorem on approximation by polynomials" and "Extension of Weierstrass's approximation theory". *Methods of Mathematical Physics* (1988), 446–448.

[27] L. Klinkenberg, C. Blumenthal, M. Chen, D. Haase, and J.-P. Katoen. 2024. Exact Bayesian Inference for Loopy Probabilistic Programs using Generating Functions. ACM. to appear in OOPSLA 2024.

[28] Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. 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.* Springer, 135–153.

[29] Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. Towards verified stochastic variational inference for probabilistic programs. *Proc. ACM Program. Lang.* 4, POPL (2020), 16:1–16:33. https://doi.org/10.1145/3371084

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

[31] Carol Mak, Fabian Zaiser, and Luke Ong. 2022. Nonparametric Involutive Markov Chain Monte Carlo. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 14802–14859. https://proceedings.mlr.press/v162/mak22a.html

[32] Annabelle McIver and Carroll Morgan. 2004. Developing and Reasoning About Probabilistic Programs in *pGCL*. In *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)*, Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock (Eds.). Springer, 123–155. https://doi.org/10.1007/11889229_4

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

[34] Sean P Meyn and Richard L Tweedie. 2012. *Markov chains and stochastic stability.* Springer Science & Business Media.

[35] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *Functional and Logic Programming - 13th*

International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9613), Oleg Kiselyov and Andy King (Eds.). Springer, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

[36] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, Jeffrey S. Foster and Dan Grossman (Eds.). ACM, 496–512. https://doi.org/10.1145/3192366.3192394

[37] David Pollard. 2002. *A user's guide to measure theoretic probability.* Number 8. Cambridge University Press.

[38] Mihai Putinar. 1993. Positive Polynomials on Compact Semi-algebraic Sets. *Indiana University Mathematics Journal* 42, 3 (1993), 969–984. http://www.jstor.org/stable/24897130

[39] Alexey Radul and Boris Alexeev. 2021. The Base Measure Problem and its Solution. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 130)*, Arindam Banerjee and Kenji Fukumizu (Eds.). PMLR, 3583–3591. http://proceedings.mlr.press/v130/radul21a.html

[40] Thomas Rainforth. 2017. *Automating inference, learning, and design using probabilistic programming.* Ph. D. Dissertation. University of Oxford.

[41] Fredrik Ronquist, Jan Kudlicka, Viktor Senderov, Johannes Borgström, Nicolas Lartillot, Daniel Lundén, Lawrence Murray, Thomas B Schön, and David Broman. 2021. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *Communications biology* 4, 1 (2021), 1–10.

[42] Reuven Y Rubinstein and Dirk P Kroese. 2016. *Simulation and the Monte Carlo method.* Vol. 10. John Wiley & Sons.

[43] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: probabilistic programming with fast exact symbolic inference. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 804–819. https://doi.org/10.1145/3453483.3454078

[44] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, Hans-Juergen Boehm and Cormac Flanagan (Eds.). ACM, 447–458. https://doi.org/10.1145/2491956.2462179

[45] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Constraint-Based Linear-Relations Analysis. In *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3148)*, Roberto Giacobazzi (Ed.). Springer, 53–68. https://doi.org/10.1007/978-3-540-27864-1_7

[46] Sam Staton, Hongseok Yang, Frank D. Wood, Chris Heunen, and Ohad Kammar. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 525–534. https://doi.org/10.1145/2933575.2935313

[47] Christiaan Swanepoel, Mathieu Fourment, Xiang Ji, Hassan Nasif, Marc A Suchard, Frederick A Matsen IV, and Alexei Drummond. 2022. TreeFlow: probabilistic programming and automatic differentiation for phylogenetics. *arXiv preprint arXiv:2211.05220* (2022).

[48] Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics* 5, 2 (1955), 285–309.

[49] David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9286)*, Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou (Eds.). Springer, 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

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

[51] Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 559–573. https://doi.org/10.1145/3453483.3454062

[52] Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2021. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 1171–1186. https://doi.org/10.1145/3453483.3454102

[53] Peixin Wang. 2022. Tail-Bound Cost Analysis over Nondeterministic Probabilistic Programs. *Journal of Shanghai Jiaotong University (Science)* (2022), 1–11.

[54] Peixin Wang, Hongfei Fu, Krishnendu Chatterjee, Yuxin Deng, and Ming Xu. 2020. Proving expected sensitivity of probabilistic programs with randomized variable-dependent termination time. *Proc. ACM Program. Lang.* 4, POPL (2020), 25:1–25:30. https://doi.org/10.1145/3371093

[55] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost Analysis of Nondeterministic Probabilistic Programs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 204–220. https://doi.org/10.1145/3314221.3314581

[56] Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and Luke Ong. 2024. Artifact of this paper. https://doi.org/10.5281/zenodo.10897200

[57] Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. arXiv:2307.13160 [cs.PL]

[58] David Williams. 1991. *Probability with martingales.* Cambridge university press.

[59] Fabian Zaiser, Andrzej S. Murawski, and Chih-Hao Luke Ong. 2023. Exact Bayesian Inference on Discrete Models via Probability Generating Functions: A Probabilistic Programming Approach. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/0747af6f877c0cb555fea595f01b0e83-Abstract-Conference.html

Received 2023-11-16; accepted 2024-03-31