Central Moment Analysis for Cost Accumulators in Probabilistic Programs

Di Wang
Carnegie Mellon University
USA

Jan Hoffmann
Carnegie Mellon University
USA

Thomas Reps
University of Wisconsin
USA

Abstract

For probabilistic programs, it is usually not possible to automatically derive exact information about their properties, such as the distribution of states at a given program point. Instead, one can attempt to derive approximations, such as upper bounds on *tail probabilities*. Such bounds can be obtained via concentration inequalities, which rely on the *moments* of a distribution, such as the expectation (the first *raw* moment) or the variance (the second *central* moment). Tail bounds obtained using central moments are often tighter than the ones obtained using raw moments, but automatically analyzing central moments is more challenging.

This paper presents an analysis for probabilistic programs that automatically derives symbolic upper and lower bounds on variances, as well as higher central moments, of *cost accumulators*. To overcome the challenges of higher-moment analysis, it generalizes analyses for expectations with an algebraic abstraction that simultaneously analyzes different moments, utilizing relations between them. A key innovation is the notion of *moment-polymorphic recursion*, and a practical derivation system that handles recursive functions.

The analysis has been implemented using a template-based technique that reduces the inference of polynomial bounds to linear programming. Experiments with our prototype central-moment analyzer show that, despite the analyzer's upper/lower bounds on various quantities, it obtains tighter tail bounds than an existing system that uses only raw moments, such as expectations.

CCS Concepts: • **Theory of computation → Probabilistic computation**; **Program analysis**; **Automated reasoning**.

Keywords: Probabilistic programs, central moments, cost analysis, tail bounds

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

ACM ISBN 978-1-4503-8391-2/21/06.
https://doi.org/10.1145/3453483.3454062

ACM Reference Format:
Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21), June 20–25, 2021, Virtual, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3453483.3454062

1 Introduction

Probabilistic programs [16, 23, 28] can be used to manipulate *uncertain* quantities modeled by probability distributions and random control flows. Uncertainty arises naturally in Bayesian networks, which capture statistical dependencies (e.g., for diagnosis of diseases [21]), and cyber-physical systems, which are subject to sensor errors and peripheral disturbances (e.g., airborne collision-avoidance systems [27]). Researchers have used probabilistic programs to implement and analyze randomized algorithms [2], cryptographic protocols [3], and machine-learning algorithms [15]. A probabilistic program propagates uncertainty through a computation, and produces a distribution over results.

In general, it is not tractable to compute the result distributions of probabilistic programs automatically and precisely: Composing simple distributions can quickly complicate the result distribution, and randomness in the control flow can easily lead to state-space explosion. Monte-Carlo simulation [33] is a common approach to study the result distributions, but the technique does not provide formal guarantees, and can sometimes be inefficient [5].

In this paper, we focus on a specific yet important kind of uncertain quantity: *cost accumulators*, which are program variables that can only be incremented or decremented through the program execution. Examples of cost accumulators include termination time [5, 10, 11, 22, 31], rewards in Markov decision processes (MDPs) [32], position information in control systems [4, 7, 34], and cash flow during bitcoin mining [41]. Recent work [7, 24, 29, 41] has proposed successful static-analysis approaches that leverage *aggregate* information of a cost accumulator X, such as X's *expected* value $\mathbb{E}[X]$ (i.e., X's "first moment"). The intuition why it is beneficial to compute aggregate information—in lieu of distributions—is that aggregate measures like expectations *abstract* distributions to a single number, while still indicating non-trivial properties. Moreover, expectations are transformed by statements in a probabilistic program in a

manner similar to the weakest-precondition transformation of formulas in a non-probabilistic program [28].

One important kind of aggregate information is *moments*. In this paper, we show how to obtain *central* moments (i.e., $\mathbb{E}[(X-\mathbb{E}[X])^k]$ for any $k \geq 2$), whereas most previous work focused on *raw* moments (i.e., $\mathbb{E}[X^k]$ for any $k \geq 1$). Central moments can provide more information about distributions. For example, the *variance* $\mathbb{V}[X]$ (i.e., $\mathbb{E}[(X-\mathbb{E}[X])^2]$, X's "second central moment") indicates how X can deviate from its mean, the *skewness* (i.e., $\frac{\mathbb{E}[(X-\mathbb{E}[X])^3]}{(\mathbb{V}[X])^{3/2}}$, X's "third standardized moment") indicates how lopsided the distribution of X is, and the *kurtosis* (i.e., $\frac{\mathbb{E}[(X-\mathbb{E}[X])^4]}{(\mathbb{V}[X])^2}$, X's "fourth standardized moment") measures the heaviness of the tails of the distribution of X. One application of moments is to answer queries about *tail bounds*, e.g., the assertions about probabilities of the form $\mathbb{P}[X \geq d]$, via *concentration-of-measure* inequalities from probability theory [12]. With central moments, we find an opportunity to obtain more precise tail bounds of the form $\mathbb{P}[X \geq d]$, and become able to derive bounds on tail probabilities of the form $\mathbb{P}[|X-\mathbb{E}[X]| \geq d]$.

Central moments $\mathbb{E}[(X-\mathbb{E}[X])^k]$ can be seen as polynomials of raw moments $\mathbb{E}[X], \cdots, \mathbb{E}[X^k]$, e.g., the variance $\mathbb{V}[X] = \mathbb{E}[(X-\mathbb{E}[X])^2]$ can be rewritten as $\mathbb{E}[X^2] - \mathbb{E}^2[X]$, where $\mathbb{E}^k[X]$ denotes $(\mathbb{E}[X])^k$. To derive bounds on central moments, we need both *upper* and *lower* bounds on the raw moments, because of the presence of *subtraction*. For example, to upper-bound $\mathbb{V}[X]$, a static analyzer needs to have an *upper* bound on $\mathbb{E}[X^2]$ and a *lower* bound on $\mathbb{E}^2[X]$.

In this work, we present and implement the first fully automatic analysis for deriving symbolic *interval* bounds on higher central moments for cost accumulators in probabilistic programs with general recursion and continuous distributions. One challenge is to support *inter*procedural reasoning to reuse analysis results for functions. Our solution makes use of a "lifting" technique from the natural-language-processing community. That technique derives an algebra for second moments from an algebra for first moments [25]. We generalize the technique to develop *moment semirings*, and use them to derive a novel *frame* rule to handle function calls with *moment-polymorphic recursion* (see §2.2).

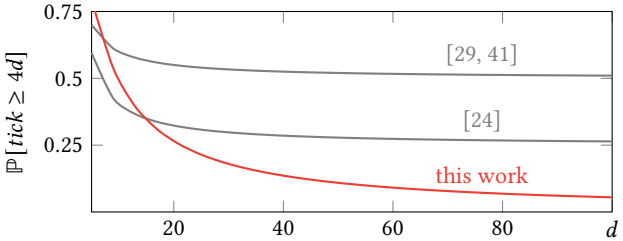
Recent work has successfully automated inference of upper [29] or lower bounds [41] on the expected cost of probabilistic programs. Kura et al. [24] developed a system to derive upper bounds on higher *raw* moments of program runtimes. However, even in combination, existing approaches can*not* solve tasks such as deriving a lower bound on the second raw moment of runtimes, or deriving an upper bound on the variance of accumulators that count live heap cells. Fig. 1(a) summarizes the features of related work on moment inference for probabilistic programs. To the best of our knowledge, our work is the first moment-analysis tool that supports all of the listed programming and analysis features. Fig. 1(b) and (c) compare our work with related work in terms of tail-bound analysis on a concrete program (see §5). The bounds are derived for the cost accumulator *tick* in a random-walk program that we will present in §2. It can be observed that for $d \geq 20$, the most precise tail bound for *tick* is the one obtained via an upper bound on the variance $\mathbb{V}[tick]$ (*tick*'s second central moment).

feature	[7]	[29]	[24]	[41]	this work
loop		✓	✓	✓	✓
recursion		✓			✓
continuous distributions	✓		✓	✓	✓
non-monotone costs	✓			✓	✓
higher moments	✓		✓		✓
interval bounds	✓			✓	✓

(a)

	[29, 41]	[24]	this work
Derived bound	$\mathbb{E}[tick] \leq 2d+4$	$\mathbb{E}[tick^2] \leq 4d^2+22d+28$	$\mathbb{V}[tick] \leq 22d+28$
Moment type	raw	raw	central
Concentration inequality	Markov (degree = 1)	Markov (degree = 2)	Cantelli
Tail bound $\mathbb{P}[tick \geq 4d]$	$\approx \frac{1}{2}$	$\approx \frac{1}{4}$	$\xrightarrow{d\to\infty} 0$

(b)



(c)

Figure 1. (a) Comparison in terms of supporting features. (b) Comparison in terms of moment bounds for the running example. (c) Comparison in terms of derived tail bounds.

Our work incorporates ideas known from the literature:

- Using the expected-potential method (or ranking supermartingales) to derive upper bounds on the expected program runtimes or monotone costs [9–11, 13, 24, 29].
- Using the *Optional Stopping Theorem* from probability theory to ensure the soundness of lower-bound inference for probabilistic programs [1, 17, 35, 41].
- Using *linear programming* (LP) to efficiently automate the (expected) potential method for (expected) cost analysis [18, 19, 40].

The contributions of our work are as follows:

- We develop moment semirings to compose the moments for a cost accumulator from two computations, and to enable interprocedural reasoning about higher moments.
- We instantiate moment semirings with the symbolic interval domain, use that to develop a derivation system for interval bounds on higher central moments for cost accumulators, and automate the derivation via LP solving.

```
1  func rdwalk() begin
2   { 2(d − x) + 4 }
3   if x < d then
4    { 2(d − x) + 4 }
5    t ∼ uniform(−1, 2);
6    { 2(d − x − t) + 5 }
7    x := x + t;
8    { 2(d − x) + 5 }
9    call rdwalk;
10   { 1 }
11   tick(1)
12   { 0 }
13  fi
14 end
```

```
1  # XID ≝ {x, d, t}
2  # FID ≝ {rdwalk}
3  # pre-condition: {d > 0}
4  func main() begin
5   x := 0;
6   call rdwalk
7  end
```

Figure 2. A bounded, biased random walk, implemented using recursion. The annotations show the derivation of an *upper* bound on the expected accumulated cost.

- We prove the soundness of our derivation system for programs that satisfy the criterion of our recent extension to the Optional Stopping Theorem, and develop an algorithm for checking this criterion automatically.
- We implemented our analysis and evaluated it on a broad suite of benchmarks from the literature. Our experimental results show that on a variety of examples, our analyzer is able to use higher central moments to obtain tighter tail bounds on program runtimes than the system of Kura et al. [24], which uses only upper bounds on raw moments.

2 Overview

In this section, we demonstrate the expected-potential method for both first-moment analysis (previous work) and higher central-moment analysis (this work) (§2.1), and discuss the challenges to supporting interprocedural reasoning and to ensuring the soundness of our approach (§2.2).

Example 2.1. The program in Fig. 2 implements a bounded, biased random walk. The main function consists of a single statement "**call** rdwalk" that invokes a recursive function. The variables x and d represent the current position and the ending position of the random walk, respectively. We assume that $d > 0$ holds initially. In each step, the program samples the length of the current move from a uniform distribution on the interval $[-1, 2]$. The statement **tick**(1) adds one to a cost accumulator that counts the number of steps before the random walk ends. We denote this accumulator by *tick* in the rest of this section. The program terminates with probability one and its expected accumulated cost is bounded by $2d + 4$.

2.1 The Expected-Potential Method for Higher-Moment Analysis

Our approach to higher-moment analysis is inspired by the *expected-potential method* [29], which is also known as *ranking super-martingales* [9, 10, 24, 41], for expected-cost bound analysis of probabilistic programs.

The classic *potential method* of amortized analysis [36] can be automated to derive symbolic cost bounds for non-probabilistic programs [18, 19]. The basic idea is to define a *potential function* $\phi : \Sigma \to \mathbb{R}^+$ that maps program states $\sigma \in \Sigma$ to nonnegative numbers, where we assume each state σ contains a cost-accumulator component $\sigma.\alpha$. If a program executes with initial state σ to final state σ', then it holds that $\phi(\sigma) \geq (\sigma'.\alpha - \sigma.\alpha) + \phi(\sigma')$, where $(\sigma'.\alpha - \sigma.\alpha)$ describes the accumulated cost from σ to σ'. The potential method also enables *compositional* reasoning: if a statement S_1 executes from σ to σ' and a statement S_2 executes from σ' to σ'', then we have $\phi(\sigma) \geq (\sigma'.\alpha - \sigma.\alpha) + \phi(\sigma')$ and $\phi(\sigma') \geq (\sigma''.\alpha - \sigma'.\alpha) + \phi(\sigma'')$; therefore, we derive $\phi(\sigma) \geq (\sigma''.\alpha - \sigma.\alpha) + \phi(\sigma'')$ for the sequential composition $S_1; S_2$. For non-probabilistic programs, the initial potential provides an *upper* bound on the accumulated cost.

This approach has been adapted to reason about expected costs of probabilistic programs [29, 41]. To derive upper bounds on the *expected* accumulated cost of a program S with initial state σ, one needs to take into consideration the *distribution* of all possible executions. More precisely, the potential function should satisfy the following property:

$$\phi(\sigma) \geq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[C(\sigma, \sigma') + \phi(\sigma')], \tag{1}$$

where the notation $\mathbb{E}_{x\sim\mu}[f(x)]$ represents the expected value of $f(x)$, where x is drawn from the distribution μ, $[\![S]\!](\sigma)$ is the distribution over final states of executing S from σ, and $C(\sigma, \sigma') \stackrel{\text{def}}{=} \sigma'.\alpha - \sigma.\alpha$ is the execution cost from σ to σ'.

Example 2.2. Fig. 2 annotates the rdwalk function from Ex. 2.1 with the derivation of an upper bound on the expected accumulated cost. The annotations, taken together, define an expected-potential function $\phi : \Sigma \to \mathbb{R}^+$ where a program state $\sigma \in \Sigma$ consists of a program point and a valuation for program variables. To justify the upper bound $2(d - x) + 4$ for the function rdwalk, one has to show that the potential right before the **tick**(1) statement should be at least 1. This property is established by *backward* reasoning on the function body:

- For **call** rdwalk, we apply the "induction hypothesis" that the expected cost of the function rdwalk can be upper-bounded by $2(d - x) + 4$. Adding the 1 unit of potential need by the tick statement, we obtain $2(d - x) + 5$ as the pre-annotation of the function call.
- For $x := x + t$, we substitute x with $x + t$ in the post-annotation of this statement to obtain the pre-annotation.
- For $t \sim$ **uniform**$(-1, 2)$, because its post-annotation is $2(d - x - t) + 5$, we compute its pre-annotation as

$$\begin{aligned}&\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[2(d-x-t)+5]\\ &=2(d-x)+5-2\cdot\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[t]\\ &=2(d-x)+5-2\cdot{}^1\!/_2=2(d-x)+4,\end{aligned}$$

which is exactly the upper bound we want to justify.

Our approach. In this paper, we focus on derivation of higher central moments. Observing that a central moment $\mathbb{E}[(X-\mathbb{E}[X])^k]$ can be rewritten as a polynomial of raw moments $\mathbb{E}[X],\cdots,\mathbb{E}[X^k]$, we reduce the problem of bounding central moments to reasoning about upper and lower bounds on raw moments. For example, the variance can be written as $\mathbb{V}[X]=\mathbb{E}[X^2]-\mathbb{E}^2[X]$, so it suffices to analyze the *upper* bound of the second moment $\mathbb{E}[X^2]$ and the *lower* bound on the square of the first moment $\mathbb{E}^2[X]$. For higher central moments, this approach requires both upper and lower bounds on higher raw moments. For example, consider the fourth central moment of a *nonnegative* random variable X: $\mathbb{E}[(X-\mathbb{E}[X])^4]=\mathbb{E}[X^4]-4\mathbb{E}[X^3]\mathbb{E}[X]+6\mathbb{E}[X^2]\mathbb{E}^2[X]-3\mathbb{E}^4[X]$. Deriving an upper bound on the fourth central moment requires lower bounds on the first ($\mathbb{E}[X]$) and third ($\mathbb{E}[X^3]$) raw moments.

We now sketch the development of *moment semirings*. We first consider only the upper bounds on higher moments of *nonnegative* costs. To do so, we extend the range of the expected-potential function ϕ to real-valued vectors $(\mathbb{R}^+)^{m+1}$, where $m\in\mathbb{N}$ is the degree of the target moment. We update the potential inequality (1) as follows:

$$\phi(\sigma)\geq\mathbb{E}_{\sigma'\sim[\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k\rangle_{0\leq k\leq m}}\otimes\phi(\sigma')],\qquad(2)$$

where $\overrightarrow{\langle v_k\rangle_{0\leq k\leq m}}$ denotes an $(m+1)$-dimensional vector, the order $\leq$ on vectors is defined pointwise, and $\otimes$ is some *composition* operator. Recall that $[\![S]\!](\sigma)$ denotes the distribution over final states of executing S from σ, and $C(\sigma,\sigma')$ describes the cost for the execution from σ to σ'. Intuitively, for $\phi(\sigma)=\overrightarrow{\langle\phi(\sigma)_k\rangle_{0\leq k\leq m}}$ and each k, the component $\phi(\sigma)_k$ is an upper bound on the k-th moment of the cost for the computation starting from σ. The 0-th moment is the *termination probability* of the computation, and we assume it is always one for now. We *cannot* simply define $\otimes$ as pointwise addition because, for example, $(a+b)^2\neq a^2+b^2$ in general. If we think of b as the cost for some probabilistic computation, and we prepend a constant cost a to the computation, then by linearity of expectations, we have $\mathbb{E}[(a+b)^2]=\mathbb{E}[a^2+2ab+b^2]=a^2+2\cdot a\cdot\mathbb{E}[b]+\mathbb{E}[b^2]$, i.e., reasoning about the second moment requires us to keep track of the first moment. Similarly, we should have $\phi(\sigma)_2\geq\mathbb{E}_{\sigma'\sim[\![S]\!](\sigma)}[C(\sigma,\sigma')^2+2\cdot C(\sigma,\sigma')\cdot\phi(\sigma')_1+\phi(\sigma')_2]$, for the second-moment component, where $\phi(\sigma')_1$ and $\phi(\sigma')_2$ denote $\mathbb{E}[b]$ and $\mathbb{E}[b^2]$, respectively. Therefore, the composition operator $\otimes$ for second-moment analysis (i.e., $m=2$) should be defined as

$$\langle 1,r_1,s_1\rangle\otimes\langle 1,r_2,s_2\rangle\stackrel{\text{def}}{=}\langle 1,r_1+r_2,s_1+2r_1r_2+s_2\rangle.\qquad(3)$$

```
1  func rdwalk() begin
2   { ⟨1, 2(d − x) + 4, 4(d − x)^2 + 22(d − x) + 28⟩ }
3   if x < d then
4    { ⟨1, 2(d − x) + 4, 4(d − x)^2 + 22(d − x) + 28⟩ }
5    t ~ uniform(−1, 2);
6    { ⟨1, 2(d − x − t) + 5, 4(d − x − t)^2 + 26(d − x − t) + 37⟩ }
7    x := x + t;
8    { ⟨1, 2(d − x) + 5, 4(d − x)^2 + 26(d − x) + 37⟩ }
9    call rdwalk;
10   { ⟨1, 1, 1⟩ }
11   tick(1)
12   { ⟨1, 0, 0⟩ }
13  fi
14 end
```

Figure 3. Derivation of an *upper* bound on the first and second moment of the accumulated cost.

Example 2.3. Fig. 3 annotates the rdwalk function from Ex. 2.1 with the derivation of an upper bound on both the first and second moment of the accumulated cost. To justify the first and second moment of the accumulated cost for the function rdwalk, we again perform backward reasoning:

- For **tick**(1), it transforms a post-annotation a by $\lambda a.(\langle 1,1,1\rangle\otimes a)$; thus, the pre-annotation is $\langle 1,1,1\rangle\otimes\langle 1,0,0\rangle=\langle 1,1,1\rangle$.
- For **call** rdwalk, we apply the "induction hypothesis", i.e., the upper bound shown on line 2. We use the $\otimes$ operator to compose the induction hypothesis with the post-annotation of this function call:
$$\begin{aligned}&\langle 1,2(d-x)+4,4(d-x)^2+22(d-x)+28\rangle\otimes\langle 1,1,1\rangle\\ &=\langle 1,2(d-x)+5,(4(d-x)^2+22(d-x)+28)+2\cdot(2(d-x)+4)+1\rangle\\ &=\langle 1,2(d-x)+5,4(d-x)^2+26(d-x)+37\rangle.\end{aligned}$$
- For $x:=x+t$, we substitute x with $x+t$ in the post-annotation of this statement to obtain the pre-annotation.
- For $t\sim\mathbf{uniform}(-1,2)$, because the post-annotation involves both t and t^2, we compute from the definition of uniform distributions that
$$\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[t]={}^1\!/_2,\quad\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[t^2]=1.$$
Then the upper bound on the second moment is derived as follows:
$$\begin{aligned}&\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[4(d-x-t)^2+26(d-x-t)+37]\\ &=(4(d-x)^2+26(d-x)+37)-(8(d-x)+26)\cdot\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[t]\\ &\quad+4\cdot\mathbb{E}_{t\sim\mathbf{uniform}(-1,2)}[t^2]\\ &=4(d-x)^2+22(d-x)+28,\end{aligned}$$
which is the same as the desired upper bound on the second moment of the accumulated cost for the function rdwalk. (See Fig. 3, line 2.)

We generalize the composition operator $\otimes$ to moments with arbitrarily high degrees, via a family of algebraic structures, which we name *moment semirings* (see §3.2). These semirings are *algebraic* in the sense that they can be instantiated with any partially ordered semiring, not just $\mathbb{R}^+$.

Interval bounds. Moment semirings not only provide a general method to analyze higher moments, but also enable reasoning about upper and lower bounds on moments *simultaneously*. The simultaneous treatment is also essential for analyzing programs with *non-monotone* costs (see §3.3).

We instantiate moment semirings with the standard interval semiring $\mathcal{I} = \{[a, b] \mid a \le b\}$. The algebraic approach allows us to systematically incorporate the interval-valued bounds, by *reinterpreting* operations in eq. (3) under $\mathcal{I}$:

$$\begin{aligned}
&\langle [1,1], [r_1^{\mathsf{L}}, r_1^{\mathsf{U}}], [s_1^{\mathsf{L}}, s_1^{\mathsf{U}}]\rangle \otimes \langle [1,1], [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}], [s_2^{\mathsf{L}}, s_2^{\mathsf{U}}]\rangle \\
&\overset{\text{def}}{=} \langle [1,1], [r_1^{\mathsf{L}}, r_1^{\mathsf{U}}] +_{\mathcal{I}} [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}], \\
&\quad [s_1^{\mathsf{L}}, s_1^{\mathsf{U}}] +_{\mathcal{I}} 2 \cdot ([r_1^{\mathsf{L}}, r_1^{\mathsf{U}}] \cdot_{\mathcal{I}} [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}]) +_{\mathcal{I}} [s_2^{\mathsf{L}}, s_2^{\mathsf{U}}]\rangle \\
&= \langle [1,1], [r_1^{\mathsf{L}} + r_2^{\mathsf{L}}, r_1^{\mathsf{U}} + r_2^{\mathsf{U}}], [s_1^{\mathsf{L}} + 2 \cdot \min S + s_2^{\mathsf{L}}, s_1^{\mathsf{U}} + 2 \cdot \max S + s_2^{\mathsf{U}}]\rangle,
\end{aligned}$$

where $S \overset{\text{def}}{=} \{r_1^{\mathsf{L}} r_2^{\mathsf{L}}, r_1^{\mathsf{L}} r_2^{\mathsf{U}}, r_1^{\mathsf{U}} r_2^{\mathsf{L}}, r_1^{\mathsf{U}} r_2^{\mathsf{U}}\}$. We then update the potential inequality eq. (2) as follows:

$$\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle [C(\sigma,\sigma')^k, C(\sigma,\sigma')^k]\rangle_{0 \le k \le m}} \otimes \phi(\sigma')],$$

where the order $\sqsubseteq$ is defined as pointwise interval inclusion.

Example 2.4. Suppose that the interval bound on the first moment of the accumulated cost of the rdwalk function from Ex. 2.1 is $[2(d-x), 2(d-x)+4]$. We can now derive the upper bound on the variance $\mathbb{V}[tick] \le 22d + 28$ shown in Fig. 1(b) (where we substitute x with 0 because the main function initializes x to 0 on line 5 in Fig. 2):

$$\begin{aligned}
\mathbb{V}[tick] &= \mathbb{E}[tick^2] - \mathbb{E}^2[tick] \\
&\le (\text{upper bnd. on } \mathbb{E}[tick^2]) - (\text{lower bnd. on } \mathbb{E}[tick])^2 \\
&= (4d^2 + 22d + 28) - (2d)^2 = 22d + 28.
\end{aligned}$$

In §5, we describe how we use moment bounds to derive the tail bounds shown in Fig. 1(c).

2.2 Two Major Challenges

Interprocedural reasoning. Recall that in the derivation of Fig. 3, we use the $\otimes$ operator to compose the upper bounds on moments for **call** rdwalk and its post-annotation $\langle 1,1,1\rangle$. However, this approach does *not* work in general, because the post-annotation might be symbolic (e.g., $\langle 1, x, x^2\rangle$) and the callee might mutate referenced program variables (e.g., x). One workaround is to derive a *pre*-annotation for each possible *post*-annotation of a recursive function, i.e., the moment annotations for a recursive function is *polymorphic*. This workaround would *not* be effective for non-tail-recursive functions: for example, we need to reason about the rdwalk function in Fig. 3 with *infinitely* many post-annotations $\langle 1,0,0\rangle$, $\langle 1,1,1\rangle$, $\langle 1,2,4\rangle$, …, i.e., $\langle 1, i, i^2\rangle$ for all $i \in \mathbb{Z}^+$.

Our solution to *moment-polymorphic recursion* is to introduce a *combination* operator $\oplus$ in a way that if ϕ_1 and ϕ_2 are two expected-potential functions, then

$$\phi_1(\sigma) \oplus \phi_2(\sigma) \ge \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k\rangle_{0 \le k \le m}} \otimes (\phi_1(\sigma') \oplus \phi_2(\sigma'))].$$

We then use the $\oplus$ operator to derive a *frame* rule:

$$\frac{\{\ Q_1\ \}\ S\ \{\ Q_1'\ \} \qquad \{\ Q_2\ \}\ S\ \{\ Q_2'\ \}}{\{\ Q_1 \oplus Q_2\ \}\ S\ \{\ Q_1' \oplus Q_2'\ \}}$$

We define $\oplus$ as pointwise addition, i.e., for second moments,

$$\langle p_1, r_1, s_1\rangle \oplus \langle p_2, r_2, s_2\rangle \overset{\text{def}}{=} \langle p_1 + p_2, r_1 + r_2, s_1 + s_2\rangle, \quad (4)$$

and because the 0-th-moment (i.e., termination-probability) component is no longer guaranteed to be one, we redefine $\otimes$ to consider the termination probabilities:

$$\langle p_1, r_1, s_1\rangle \otimes \langle p_2, r_2, s_2\rangle \overset{\text{def}}{=} \langle p_1 p_2, p_2 r_1 + p_1 r_2, p_2 s_1 + 2 r_1 r_2 + p_1 s_2\rangle. \quad (5)$$

Remark 2.5. As we will show in §3.2, the composition operator $\otimes$ and combination operator $\oplus$ form a moment semiring*; consequently, we can use* algebraic *properties of semirings (e.g., distributivity) to aid higher-moment analysis. For example, a vector $\langle 0, r_1, s_1\rangle$ whose termination-probability component is zero does not seem to make sense, because moments with respect to a zero distribution should also be zero. However, by distributivity, we have*

$$\begin{aligned}
&\langle 1, r_3, s_3\rangle \otimes \langle 1, r_1 + r_2, s_1 + s_2\rangle \\
&= \langle 1, r_3, s_3\rangle \otimes (\langle 0, r_1, s_1\rangle \oplus \langle 1, r_2, s_2\rangle) \\
&= (\langle 1, r_3, s_3\rangle \otimes \langle 0, r_1, s_1\rangle) \oplus (\langle 1, r_3, s_3\rangle \oplus \langle 1, r_2, s_2\rangle).
\end{aligned}$$

If we think of $\langle 1, r_1 + r_2, s_1 + s_2\rangle$ as a post-annotation of a computation whose moments are bounded by $\langle 1, r_3, s_3\rangle$, the equation above indicates that we can use $\oplus$ to decompose *the post-annotation into subparts, and then reason about each subpart separately. This fact inspires us to develop a decomposition technique for moment-polymorphic recursion.*

Example 2.6. With the $\oplus$ operator and the frame rule, we only need to analyze the rdwalk function from Ex. 2.1 with *three* post-annotations: $\langle 1,0,0\rangle$, $\langle 0,1,1\rangle$, and $\langle 0,0,2\rangle$, which form a kind of "elimination sequence." We construct this sequence in an *on-demand* manner; the first post-annotation is the identity element $\langle 1,0,0\rangle$ of the moment semiring.

For post-annotation $\langle 1,0,0\rangle$, as shown in Fig. 3, we need to know the moment bound for rdwalk with the post-annotation $\langle 1,1,1\rangle$. Instead of reanalyzing rdwalk with the post-annotation $\langle 1,1,1\rangle$, we use the $\oplus$ operator to compute the "difference" between it and the previous post-annotation $\langle 1,0,0\rangle$. Observing that $\langle 1,1,1\rangle = \langle 1,0,0\rangle \oplus \langle 0,1,1\rangle$, we now analyze rdwalk with $\langle 0,1,1\rangle$ as the post-annotation:

```
1  call rdwalk; { ⟨0, 1, 3⟩ }  # = ⟨1, 1, 1⟩ ⊗ ⟨0, 1, 1⟩
2  tick(1) { ⟨0, 1, 1⟩ }
```

Again, because $\langle 0,1,3\rangle = \langle 0,1,1\rangle \oplus \langle 0,0,2\rangle$, we need to further analyze rdwalk with $\langle 0,0,2\rangle$ as the post-annotation:

```
1  call rdwalk; { ⟨0, 0, 2⟩ }  # = ⟨1, 1, 1⟩ ⊗ ⟨0, 0, 2⟩
2  tick(1) { ⟨0, 0, 2⟩ }
```

With the post-annotation $\langle 0,0,2\rangle$, we can now reason *monomorphically* without analyzing any new post-annotation! We can perform a succession of reasoning steps similar to what we have done in Ex. 2.2 to justify the following bounds ("unwinding" the elimination sequence):

```
1 func geo() begin { ⟨1, 2^x⟩ }
2  x ≔ x + 1; { ⟨1, 2^{x−1}⟩ }
3  # expected-potential method for lower bounds:
4  # 2^{x−1} < 1/2 · (2^x + 1) + 1/2 · 0
5  if prob(1/2) then { ⟨1, 2^x + 1⟩ }
6   tick(1); { ⟨1, 2^x⟩ }
7   call geo { ⟨1, 0⟩}
8  fi
9 end
```

Figure 4. A purely probabilistic loop with annotations for a *lower* bound on the first moment of the accumulated cost.

- $\{\langle 0,0,2\rangle\}$ rdwalk $\{\langle 0,0,2\rangle\}$: Directly by backward reasoning with the post-annotation $\langle 0,0,2\rangle$.
- $\{\langle 0,1,4(d-x)+9\rangle\}$ rdwalk $\{\langle 0,1,1\rangle\}$: To analyze the recursive call with post-annotation $\langle 0,1,3\rangle$, we use the frame rule with the post-call-site annotation $\langle 0,0,2\rangle$ to derive $\langle 0,1,4(d-x)+11\rangle$ as the pre-annotation:
  ```
  1 { ⟨0, 1, 4(d − x) + 11⟩ }  # = ⟨0, 1, 4(d − x) + 9⟩ ⊕ ⟨0, 0, 2⟩
  2 call rdwalk;
  3 { ⟨0, 1, 3⟩ }  # = ⟨0, 1, 1⟩ ⊕ ⟨0, 0, 2⟩
  ```
- $\{\langle 1,2(d-x)+4,4(d-x)^2+22(d-x)+28\rangle\}$ rdwalk $\{\langle 1,0,0\rangle\}$: To analyze the recursive call with post-annotation $\langle 1,1,1\rangle$, we use the frame rule with the post-call-site annotation $\langle 0,1,1\rangle$ to derive $\langle 1,2(d-x)+5,4(d-x)^2+26(d-x)+37\rangle$ as the pre-annotation:
  ```
  1 { ⟨1, 2(d−x)+5, 4(d−x)^2+26(d−x)+37⟩ }
  2 # = ⟨1, 2(d−x)+4, 4(d−x)^2+22(d−x)+28⟩ ⊕ ⟨0, 1, 4(d−x)+9⟩
  3 call rdwalk;
  4 { ⟨1, 1, 1⟩ }  # = ⟨1, 0, 0⟩ ⊕ ⟨0, 1, 1⟩
  ```

In §3.3, we present an automatic inference system for the expected-potential method that is extended with interval-valued bounds on higher moments, with support for moment-polymorphic recursion.

Soundness of the analysis. Unlike the classic potential method, the expected-potential method is *not* always sound when reasoning about the moments for cost accumulators in probabilistic programs.

Counterexample 2.7. Consider the program in Fig. 4 that describes a purely probabilistic loop that exits the loop with probability 1/2 in each iteration. The expected accumulated cost of the program should be *one* [17]. However, the annotations in Fig. 4 justify a potential function 2^x as a *lower* bound on the expected accumulated cost, no matter what value x has at the beginning, which is apparently *unsound*.

Why does the expected-potential method fail in this case? The short answer is that dualization only works for some problems: upper-bounding the sum of nonnegative ticks is equivalent to lower-bounding the sum of nonpositive ticks; lower-bounding the sum of nonnegative ticks—the issue in Fig. 4—is equivalent to upper-bounding the sum of nonpositive ticks; however, the two kinds of problems are *inherently different* [17]. Intuitively, the classic potential method for bounding the costs of non-probabilistic programs is a *partial*-correctness method, i.e., derived upper/lower bounds are sound if the analyzed program terminates [30]. With probabilistic programs, many programs do not terminate *definitely*, but only *almost surely*, i.e., they terminate with probability one, but have some execution traces that are non-terminating. The programs in Figs. 2 and 4 are both almost-surely terminating. For the expected-potential method, the potential at a program state can be seen as an *average* of potentials needed for all possible computations that continue from the state. If the program state can lead to a non-terminating execution trace, the potential associated with that trace might be problematic, and as a consequence, the expected-potential method might fail.

$$
\begin{aligned}
S &::= \textbf{skip} \mid \textbf{tick}(c) \mid x \coloneqq E \mid x \sim D \mid \textbf{call}\ f \mid \textbf{while}\ L\ \textbf{do}\ S\ \textbf{od}\\
&\quad \mid \textbf{if prob}(p)\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid \textbf{if}\ L\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid S_1; S_2\\
L &::= \textbf{true} \mid \textbf{not}\ L \mid L_1\ \textbf{and}\ L_2 \mid E_1 \le E_2\\
E &::= x \mid c \mid E_1 + E_2 \mid E_1 \times E_2\\
D &::= \textbf{uniform}(a,b) \mid \cdots
\end{aligned}
$$

Figure 5. Syntax of Appl, where $p \in [0,1]$, $a,b,c \in \mathbb{R}$, $a<b$, $x \in$ VID is a variable, and $f \in$ FID is a function identifier.

Recent research [1, 17, 35, 41] has employed the *Optional Stopping Theorem* (OST) from probability theory to address this soundness issue. The classic OST provides a collection of *sufficient* conditions for reasoning about expected gain *upon termination* of stochastic processes, where the expected gain at any time is *invariant*. By constructing a stochastic process for executions of probabilistic programs and setting the expected-potential function as the invariant, one can apply the OST to justify the soundness of the expected-potential function. In a companion paper [39], we study and propose an extension to the classic OST with a *new* sufficient condition that is suitable for reasoning about higher moments; in this work, we prove the soundness of our central-moment inference for programs that satisfy this condition, and develop an algorithm to check this condition automatically (see §4).

3 Derivation System for Higher Moments

In this section, we describe the inference system used by our analysis. We first present a probabilistic programming language (§3.1). We then introduce *moment semirings* to compose higher moments for a cost accumulator from two computations (§3.2). We use moment semirings to develop our derivation system, which is presented as a declarative program logic (§3.3). Finally, we sketch how we reduce the inference of a derivation to LP solving (§3.4).

3.1 A Probabilistic Programming Language

This paper uses an imperative arithmetic probabilistic programming language Appl that supports general recursion and continuous distributions, where program variables are

real-valued. We use the following notational conventions. Natural numbers $\mathbb{N}$ *exclude* 0, i.e., $\mathbb{N} \stackrel{\text{def}}{=} \{1, 2, 3, \cdots\} \subseteq \mathbb{Z}^+ \stackrel{\text{def}}{=} \{0, 1, 2, \cdots\}$. The *Iverson brackets* $[\cdot]$ are defined by $[\varphi] = 1$ if φ is true and otherwise $[\varphi] = 0$. We denote updating an existing binding of x in a finite map f to v by $f[x \mapsto v]$. We will also use the following standard notions from probability theory: σ-algebras, measurable spaces, measurable functions, random variables, probability measures, and expectations. We include a review of those notions in the technical report [38].

Fig. 5 presents the syntax of Appl, where the metavariables S, L, E, and D stand for statements, conditions, expressions, and distributions, respectively. Each distribution D is associated with a probability measure $\mu_D \in \mathbb{D}(\mathbb{R})$. We write $\mathbb{D}(X)$ for the collection of all probability measures on the measurable space X. For example, $\mathbf{uniform}(a, b)$ describes a uniform distribution on the interval $[a, b]$, and its corresponding probability measure is the integration of its density function $\mu_{\mathbf{uniform}(a,b)}(O) \stackrel{\text{def}}{=} \int_O \frac{[a \le x \le b]}{b-a} dx$. The statement "$x \sim D$" is a *random-sampling* assignment, which draws from the distribution μ_D to obtain a sample value and then assigns it to x. The statement "**if prob**(p) **then** S_1 **else** S_2 **fi**" is a *probabilistic-branching* statement, which executes S_1 with probability p, or S_2 with probability $(1 - p)$.

The statement "**call** f" makes a (possibly recursive) call to the function with identifier $f \in \mathsf{FID}$. In this paper, we assume that the functions only manipulate states that consist of global program variables. The statement **tick**(c), where $c \in \mathbb{R}$ is a constant, is used to define the *cost model*. It adds c to an anonymous global cost accumulator. Note that our implementation supports local variables, function parameters, return statements, as well as accumulation of non-constant costs; the restrictions imposed here are not essential, and are introduced solely to simplify the presentation.

We use a pair $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ to represent an Appl program, where $\mathscr{D}$ is a finite map from function identifiers to their bodies and S_{main} is the body of the main function. We present an operational semantics for Appl in §4.1.

3.2 Moment Semirings

As discussed in §2.1, we want to design a *composition* operation $\otimes$ and a *combination* operation $\oplus$ to compose and combine higher moments of accumulated costs such that

$$\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma, \sigma')^k \rangle_{0 \le k \le m}} \otimes \phi(\sigma')],$$

$$\phi_1(\sigma) \oplus \phi_2(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma, \sigma')^k \rangle_{0 \le k \le m}} \otimes (\phi_1(\sigma') \oplus \phi_2(\sigma'))],$$

where the expected-potential functions ϕ, ϕ_1, ϕ_2 map program states to interval-valued vectors, $C(\sigma, \sigma')$ is the cost for the computation from σ to σ', and m is the degree of the target moment. In eqs. (4) and (5), we gave a definition of $\otimes$ and $\oplus$ suitable for first and second moments, respectively. In this section, we generalize them to reason about upper and lower bounds of higher moments. Our approach is inspired by the work of Li and Eisner [25], which develops a method to "lift" techniques for first moments to those for second moments. Instead of restricting the elements of semirings to be vectors of numbers, we propose *algebraic* moment semirings that can also be instantiated with vectors of intervals, which we need for the interval-bound analysis that was demonstrated in §2.1.

Definition 3.1. The m-th order *moment semiring* $\mathcal{M}_{\mathcal{R}}^{(m)} = (|\mathcal{R}|^{m+1}, \oplus, \otimes, \underline{0}, \underline{1})$ is parametrized by a partially ordered semiring $\mathcal{R} = (|\mathcal{R}|, \le, +, \cdot, 0, 1)$, where

$$\overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \oplus \overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \stackrel{\text{def}}{=} \overrightarrow{\langle u_k + v_k \rangle_{0 \le k \le m}}, \tag{6}$$

$$\overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \otimes \overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \stackrel{\text{def}}{=} \overrightarrow{\langle \textstyle\sum_{i=0}^{k} \binom{k}{i} \times (u_i \cdot v_{k-i}) \rangle_{0 \le k \le m}}, \tag{7}$$

$\binom{k}{i}$ is the binomial coefficient; the scalar product $n \times u$ is an abbreviation for $\sum_{i=1}^{n} u$, for $n \in \mathbb{Z}^+, u \in \mathcal{R}$; $\underline{0} \stackrel{\text{def}}{=} \langle 0, 0, \cdots, 0 \rangle$; and $\underline{1} \stackrel{\text{def}}{=} \langle 1, 0, \cdots, 0 \rangle$. We define the partial order $\sqsubseteq$ as the pointwise extension of the partial order $\le$ on $\mathcal{R}$.

Intuitively, the definition of $\otimes$ in eq. (7) can be seen as the multiplication of two moment-generating functions for distributions with moments $\overrightarrow{\langle u_k \rangle_{0 \le k \le m}}$ and $\overrightarrow{\langle v_k \rangle_{0 \le k \le m}}$, respectively. We prove a composition property for moment semirings.

Lemma 3.2. *For all $u, v \in \mathcal{R}$, it holds that*

$$\overrightarrow{\langle (u+v)^k \rangle_{0 \le k \le m}} = \overrightarrow{\langle u^k \rangle_{0 \le k \le m}} \otimes \overrightarrow{\langle v^k \rangle_{0 \le k \le m}},$$

where u^n is an abbreviation for $\prod_{i=1}^{n} u$, for $n \in \mathbb{Z}^+, u \in \mathcal{R}$.

3.3 Inference Rules

We present the derivation system as a declarative program logic that uses moment semirings to enable compositional reasoning and moment-polymorphic recursion.

Interval-valued moment semirings. Our derivation system infers upper and lower bounds simultaneously, rather than separately, which is essential for *non-monotone* costs. Consider a program "**tick**(-1); S" and suppose that we have $\langle 1, 2, 5 \rangle$ and $\langle 1, -2, 5 \rangle$ as the upper and lower bound on the first two moments of the cost for S, respectively. If we only use the upper bound, we derive $\langle 1, -1, 1 \rangle \otimes \langle 1, 2, 5 \rangle = \langle 1, 1, 2 \rangle$, which is *not* an upper bound on the moments of the cost for the program; if the *actual* moments of the cost for S are $\langle 1, 0, 5 \rangle$, then the *actual* moments of the cost for "**tick**(-1); S" are $\langle 1, -1, 1 \rangle \otimes \langle 1, 0, 5 \rangle = \langle 1, -1, 4 \rangle \not\le \langle 1, 1, 2 \rangle$. Thus, in the analysis, we instantiate moment semirings with the interval domain $\mathcal{I}$. For the program "**tick**(-1); S", its interval-valued bound on the first two moments is $\langle [1, 1], [-1, -1], [1, 1] \rangle \otimes \langle [1, 1], [-2, 2], [5, 5] \rangle = \langle [1, 1], [-3, 1], [2, 10] \rangle$.

Template-based expected-potential functions. The basic approach to automated inference using potential functions is to introduce a *template* for the expected-potential

functions. Let us fix $m \in \mathbb{N}$ as the degree of the target moment. Because we use $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential functions whose range is vectors of intervals, the templates are vectors of intervals whose ends are represented *symbolically*. In this paper, we represent the ends of intervals by *polynomials* in $\mathbb{R}[\mathsf{VID}]$ over program variables.

More formally, we lift the interval semiring $\mathcal{I}$ to a *symbolic* interval semiring $\mathcal{PI}$ by representing the ends of the k-th interval by polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$ up to degree kd for some fixed $d \in \mathbb{N}$. Let $\mathcal{M}_{\mathcal{PI}}^{(m)}$ be the m-th order moment semiring instantiated with the symbolic interval semiring. Then the potential annotation is represented as $Q = \overrightarrow{\langle [L_k, U_k] \rangle_{0 \le k \le m}} \in \mathcal{M}_{\mathcal{PI}}^{(m)}$, where L_k's and U_k's are polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$. Q defines an $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential function $\phi_Q(\sigma) \stackrel{\text{def}}{=} \overrightarrow{\langle [\sigma(L_k), \sigma(U_k)] \rangle_{0 \le k \le m}}$, where σ is a program state, and $\sigma(L_k)$ and $\sigma(U_k)$ are L_k and U_k evaluated over σ, respectively.

Inference rules. We formalize our derivation system for moment analysis in a Hoare-logic style. The judgment has the form $\Delta \vdash_h \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}$, where S is a statement, $\{\Gamma; Q\}$ is a precondition, $\{\Gamma'; Q'\}$ is a postcondition, $\Delta = \langle \Delta_k \rangle_{0 \le k \le m}$ is a context of function specifications, and $h \in \mathbb{Z}^+$ specifies some restrictions put on Q, Q' that we will explain later. The *logical context* $\Gamma : (\mathsf{VID} \to \mathbb{R}) \to \{\top, \bot\}$ is a predicate that describes reachable states at a program point. The *potential annotation* $Q \in \mathcal{M}_{\mathcal{PI}}^{(m)}$ specifies a map from program states to the moment semiring that is used to define interval-valued expected-potential functions. The semantics of the triple $\{\cdot; Q\}\ S\ \{\cdot; Q'\}$ is that if the rest of the computation after executing S has its moments of the accumulated cost bounded by $\phi_{Q'}$, then the whole computation has its moments of the accumulated cost bounded by ϕ_Q. The parameter h restricts all i-th-moment components in Q, Q', such that $i < h$, to be $[0, 0]$. We call such potential annotations *h-restricted*; this construction is motivated by an observation from Ex. 2.6, where we illustrated the benefits of carrying out interprocedural analysis using an "elimination sequence" of annotations for recursive function calls, where the successive annotations have a greater number of zeros, filling from the left. *Function specifications* are valid pairs of pre- and post-conditions for all declared functions in a program. For each k, such that $0 \le k \le m$, and each function f, a valid specification $(\Gamma; Q, \Gamma'; Q') \in \Delta_k(f)$ is justified by the judgment $\Delta \vdash_k \{\Gamma; Q\}\ \mathscr{D}(f)\ \{\Gamma'; Q'\}$, where $\mathscr{D}(f)$ is the function body of f, and Q, Q' are k-restricted. To perform context-sensitive analysis, a function can have multiple specifications.

Fig. 6 presents some of the inference rules. The rule (Q-Tick) is the only rule that deals with costs in a program. To accumulate the moments of the cost, we use the $\otimes$ operation in the moment semiring $\mathcal{M}_{\mathcal{PI}}^{(m)}$. The rule (Q-Sample) accounts for sampling statements. Because "$x \sim D$" randomly

$$\text{(Q-Tick)}\quad \frac{Q = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q'}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{tick}(c)\ \{\Gamma; Q'\}} \qquad \text{(Q-Sample)}\quad \frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \quad Q = \mathbb{E}_{x \sim \mu_D}[Q']}{\Delta \vdash_h \{\Gamma; Q\}\ x \sim D\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Loop)}\quad \frac{\Delta \vdash_h \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\}}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{while}\ L\ \mathbf{do}\ S\ \mathbf{od}\ \{\Gamma \wedge \neg L; Q\}} \qquad \text{(Q-Call-Mono)}\quad \frac{(\Gamma; Q, \Gamma'; Q') \in \Delta_m(f)}{\Delta \vdash_m \{\Gamma; Q\}\ \mathbf{call}\ f\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Seq)}\quad \frac{\Delta \vdash_h \{\Gamma; Q\}\ S_1\ \{\Gamma'; Q'\} \quad \Delta \vdash_h \{\Gamma'; Q'\}\ S_2\ \{\Gamma''; Q''\}}{\Delta \vdash_h \{\Gamma; Q\}\ S_1; S_2\ \{\Gamma''; Q''\}} \qquad \text{(Q-Call-Poly)}\quad \frac{h < m \quad \Delta_h(f) = (\Gamma; Q_1, \Gamma'; Q_1') \quad \Delta \vdash_{h+1} \{\Gamma; Q_2\}\ \mathscr{D}(f)\ \{\Gamma'; Q_2'\}}{\Delta \vdash_h \{\Gamma; Q_1 \oplus Q_2\}\ \mathbf{call}\ f\ \{\Gamma'; Q_1' \oplus Q_2'\}}$$

$$\text{(Q-Prob)}\quad \frac{\Delta \vdash_h \{\Gamma; Q_1\}\ S_1\ \{\Gamma'; Q'\} \quad \Delta \vdash_h \{\Gamma; Q_2\}\ S_2\ \{\Gamma'; Q'\} \quad Q = P \oplus R \quad P = \langle [p, p], [0, 0], \cdots, [0, 0] \rangle \otimes Q_1 \quad R = \langle [1-p, 1-p], [0, 0], \cdots, [0, 0] \rangle \otimes Q_2}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{if}\ \mathbf{prob}(p)\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{\Gamma'; Q'\}}$$

Figure 6. Selected inference rules of the derivation system.

assigns a value to x in the support of distribution D, we quantify x out universally from the logical context. To compute $Q = \mathbb{E}_{x \sim \mu_D}[Q']$, where x is drawn from distribution D, we assume the moments for D are well-defined and computable, and substitute x^i, $i \in \mathbb{N}$ with the corresponding moments in Q'. We make this assumption because every component of Q' is a polynomial over program variables. For example, if $D = \mathbf{uniform}(-1, 2)$, we know the following facts $\mathbb{E}_{x \sim \mu_D}[x^0] = 1, \mathbb{E}_{x \sim \mu_D}[x^1] = 1/2, \mathbb{E}_{x \sim \mu_D}[x^2] = 1, \mathbb{E}_{x \sim \mu_D}[x^3] = 5/4$. Then for $Q' = \langle [1, 1], [1 + x^2, xy^2 + x^3 y] \rangle$, by the linearity of expectations, we compute $Q = \mathbb{E}_{x \sim \mu_D}[Q']$ as follows:

$$\begin{aligned} \mathbb{E}_{x \sim \mu_D}[Q'] &= \langle [1, 1], [\mathbb{E}_{x \sim \mu_D}[1 + x^2], \mathbb{E}_{x \sim \mu_D}[xy^2 + x^3 y] \rangle \\ &= \langle [1, 1], [1 + \mathbb{E}_{x \sim \mu_D}[x^2], y^2 \mathbb{E}_{x \sim \mu_D}[x] + y \mathbb{E}_{x \sim \mu_D}[x^3]] \rangle \\ &= \langle [1, 1], [2, 1/2 \cdot y^2 + 5/4 \cdot y] \rangle. \end{aligned}$$

The other probabilistic rule (Q-Prob) deals with probabilistic branching. Intuitively, if the moments of the execution of S_1 and S_2 are q_1 and q_2, respectively, and those of the accumulated cost of the computation after the branch statement is bounded by $\phi_{Q'}$, then the moments for the whole computation should be bounded by a "weighted average" of $(q_1 \otimes \phi_{Q'})$ and $(q_2 \otimes \phi_{Q'})$, with respect to the branching probability p. We implement the weighted average by the combination operator $\oplus$ applied to $\langle [p, p], [0, 0], \cdots, [0, 0] \rangle \otimes q_1 \otimes \phi_{Q'}$ and $\langle [1-p, 1-p], [0, 0], \cdots, [0, 0] \rangle \otimes q_2 \otimes \phi_{Q'}$, because the 0-th moments denote probabilities.

The rules (Q-Call-Poly) and (Q-Call-Mono) handle function calls. Recall that in Ex. 2.6, we use the $\oplus$ operator to combine multiple potential functions for a function to reason about recursive function calls. The restriction parameter h is used to ensure that the derivation system only needs to reason about *finitely* many post-annotations for each call site. In rule (Q-Call-Poly), where h is smaller than the target moment m, we fetch the pre- and post-condition Q_1, Q_1' for

```
1  func rdwalk() begin
2   { x<d+2; ⟨[1,1], [2(d−x), 2(d−x)+4],
3     [4(d−x)^2 + 6(d−x)−4, 4(d−x)^2+22(d−x)+28]⟩ }
4   if x < d then
5    { x<d; ⟨[1,1], [2(d−x), 2(d−x)+4],
6      [4(d−x)^2+6(d−x)−4, 4(d−x)^2+22(d−x)+28]⟩ }
7    t ~ uniform(−1, 2);
8    { x<d∧t≤2; ⟨[1,1], [2(d−x−t)+1, 2(d−x−t)+5],
9      [4(d−x−t)^2 + 10(d−x−t)−3, 4(d−x−t)^2+26(d−x−t)+37)]⟩ }
10   x := x + t;
11   { x<d+2; ⟨[1,1], [2(d−x)+1, 2(d−x)+5],
12     [4(d−x)^2+10(d−x)−3, 4(d−x)^2+26(d−x)+37]⟩ }
13   call rdwalk;
14   { ⊤; ⟨[1,1], [1,1], [1,1]⟩ }
15   tick(1)
16   { ⊤; ⟨[1,1], [0,0], [0,0]⟩ }
17  fi
18 end
```

Figure 7. The rdwalk function with annotations for the interval-bounds on the first and second moments.

the function f from the specification context Δ_h. We then combine it with a *frame* of $(h+1)$-restricted potential annotations Q_2, Q_2' for the function f. The frame is used to account for the interval bounds on the moments for the computation after the function call for most non-tail-recursive programs. When h reaches the target moment m, we use the rule (Q-Call-Mono) to reason *moment-monomorphically*, because setting h to $m+1$ implies that the frame can only be $\langle [0,0], [0,0], \cdots, [0,0]\rangle$.

Example 3.3. Fig. 7 presents the logical context and the complete potential annotation for the first and second moments for the cost accumulator *tick* of the rdwalk function from Ex. 2.1. Similar to the reasoning in Ex. 2.6, we can justify the derivation using moment-polymorphic recursion and the moment bounds for rdwalk with post-annotations $\langle [0,0], [1,1], [1,1]\rangle$ and $\langle [0,0], [0,0], [2,2]\rangle$.

3.4 Automatic Linear-Constraint Generation

We adapt existing techniques [8, 29] to automate our inference system by (i) using an abstract interpreter to infer logical contexts, (ii) generating templates and linear constraints by inductively applying the derivation rules to the analyzed program, and (iii) employing an off-the-shelf LP solver to discharge the linear constraints. During the generation phase, the coefficients of monomials in the polynomials from the ends of the intervals in every qualitative context $Q \in \mathcal{M}^{(m)}_{\mathcal{PI}}$ are recorded as symbolic names, and the inequalities among those coefficients—derived from the inference rules in Fig. 6—are emitted to the LP solver.

Example 3.4. We demonstrate linear-constraint generation for the upper bound on the first moment for the sampling statement $x \sim \textbf{uniform}(-1,2)$ with a pre-annotation $\langle [0,0], [0, q_{x^2}\cdot x^2 + q_x \cdot x + q_1 \cdot 1]\rangle$ and a post-annotation $\langle [0,0], [0, q'_{x^2}\cdot x^2 + q'_x \cdot x + q'_1 \cdot 1]\rangle$, where we use polynomials of x up to degree 2 as the templates, and $q_{x^2}, q_x, q_1, q'_{x^2}, q'_x, q'_1$ are unknown numeric coefficients. By (Q-Sample), we generate constraints to perform "partial evaluation" on the polynomials by substituting x with the moments of $\textbf{uniform}(-1,2)$. Let D denote $\textbf{uniform}(-1,2)$. Because

$$\mathbb{E}_{x\sim\mu_D}[q'_{x^2}\cdot x^2 + q'_x \cdot x + q'_1 \cdot 1] = (q'_{x^2}\cdot 1 + q'_x \cdot {}^1\!/_2 + q'_1 \cdot 1),$$

we generate these linear constraints:

$$q_{x^2} = 0, \quad q_x = 0, \quad q_1 = q'_{x^2} + q'_x \cdot {}^1\!/_2 + q'_1.$$

Constraint generation for other inference rules is similar to the process we describe in Ex. 3.4. For example, let us consider the loop rule (Q-Loop). Instead of computing the loop invariant Q explicitly, our system represents Q directly as a template with unknown coefficients, then uses Q as the post-annotation to analyze the loop body and obtain a pre-annotation, and finally generates linear constraints that indicate that the pre-annotation equals Q. Details of the linear-constraint generation of our system are included in the technical report [38].

The LP solver not only finds assignments to the coefficients that satisfy the constraints, it can also optimize a linear objective function. In our central-moment analysis, we construct an objective function that tries to minimize imprecision. For example, let us consider upper bounds on the variance. We randomly pick a concrete valuation of program variables that satisfies the pre-condition (e.g., $d > 0$ in Fig. 2), and then substitute program variables with the concrete valuation in the polynomial for the upper bound on the variance (obtained from bounds on the raw moments). The resulting linear combination of coefficients, which we set as the objective function, stands for the variance under the concrete valuation. Thus, minimizing the objective function produces the most precise upper bound on the variance under the specific concrete valuation. Also, we can extract a *symbolic* upper bound on the variance using the assignments to the coefficients. Because the derivation of the bounds only uses the given pre-condition, the symbolic bounds apply to all valuations that satisfy the pre-condition.

4 Soundness of Higher-Moment Analysis

In this section, we study the soundness of our derivation system for higher-moment analysis. We first present a Markov-chain semantics for the probabilistic programming language Appl to reason about how *stepwise* costs contribute to the *global* accumulated cost (§4.1). We then formulate higher-moment analysis with respect to the semantics and prove the soundness of our derivation system for higher-moment analysis based on a recent extension to the *Optional Stopping Theorem* (§4.2). Finally, we sketch the algorithmic approach for ensuring the soundness of our analysis (§4.3).

4.1 A Markov-Chain Semantics

Operational semantics. We start with a small-step operational semantics with continuations, which we will use later to construct the Markov-chain semantics. We follow a distribution-based approach [6, 23] to define an operational cost semantics for Appl. Full details of the semantics are included in the technical report [38]. A *program configuration* $\sigma \in \Sigma$ is a quadruple $\langle \gamma, S, K, \alpha \rangle$ where $\gamma : \mathsf{VID} \to \mathbb{R}$ is a program state that maps variables to values, S is the statement being executed, K is a continuation that describes what remains to be done after the execution of S, and $\alpha \in \mathbb{R}$ is the global cost accumulator. An execution of an Appl program $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ is initialized with $\langle \lambda\_.0, S_{\mathsf{main}}, \mathbf{Kstop}, 0 \rangle$, and the termination configurations have the form $\langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle$, where **Kstop** is an empty continuation.

Different from a standard semantics where each program configuration steps to at most one new configuration, a probabilistic semantics may pick several different new configurations. The evaluation relation for Appl has the form $\sigma \mapsto \mu$ where $\mu \in \mathbb{D}(\Sigma)$ is a probability measure over configurations. Below are two example rules. The rule (E-Prob) constructs a distribution whose support has exactly two elements, which stand for the two branches of the probabilistic choice. We write $\delta(\sigma)$ for the *Dirac measure* at σ, defined as $\lambda A.[\sigma \in A]$ where A is a measurable subset of Σ. We also write $p \cdot \mu_1 + (1-p) \cdot \mu_2$ for a convex combination of measures μ_1 and μ_2 where $p \in [0,1]$, defined as $\lambda A. p \cdot \mu_1(A) + (1-p) \cdot \mu_2(A)$. The rule (E-Sample) "pushes" the probability distribution of D to a distribution over post-sampling program configurations.

$$\frac{S = \textbf{if } \textbf{prob}(p) \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}}{\langle \gamma, S, K, \alpha \rangle \mapsto p \cdot \delta(\langle \gamma, S_1, K, \alpha \rangle) + (1-p) \cdot \delta(\langle \gamma, S_2, K, \alpha \rangle)} \text{ (E-Prob)}$$

$$\frac{}{\langle \gamma, x \sim D, K, \alpha \rangle \mapsto \lambda A. \mu_D(\{r \mid \langle \gamma[x \mapsto r], \mathbf{skip}, K, \alpha \rangle \in A\})} \text{ (E-Sample)}$$

Example 4.1. Suppose that a random sampling statement is being executed, i.e., the current configuration is

$$\langle \{t \mapsto t_0\}, (t \sim \textbf{uniform}(-1, 2)), K_0, \alpha_0 \rangle.$$

The probability measure for the uniform distribution is $\lambda O. \int_O \frac{[-1 \le x \le 2]}{3} dx$. Thus, by the rule (E-Sample), we derive the post-sampling probability measure over configurations:

$$\lambda A. \int_{\mathbb{R}} [\langle \{t \mapsto r\}, \mathbf{skip}, K_0, \alpha_0 \rangle \in A] \cdot \tfrac{[-1 \le r \le 2]}{3} dr.$$

A Markov-chain semantics. In this work, we harness Markov-chain-based reasoning [22, 31] to develop a Markov-chain cost semantics for Appl, based on the evaluation relation $\sigma \mapsto \mu$. An advantage of this approach is that it allows us to study how the cost of every single evaluation step contributes to the accumulated cost at the exit of the program. Details of this semantics are included in the technical report [38].

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be the probability space where $\Omega \stackrel{\text{def}}{=} \Sigma^{\mathbb{Z}^+}$ is the set of all *infinite* traces over program configurations, $\mathcal{F}$ is a σ-algebra on Ω, and $\mathbb{P}$ is a probability measure on $(\Omega, \mathcal{F})$ obtained by the evaluation relation $\sigma \mapsto \mu$ and the initial configuration $\langle \lambda\_.0, S_{\mathsf{main}}, \mathbf{Kstop}, 0 \rangle$. Intuitively, $\mathbb{P}$ specifies the probability distribution over all possible executions of a probabilistic program. The probability of an assertion θ with respect to $\mathbb{P}$, written $\mathbb{P}[\theta]$, is defined as $\mathbb{P}(\{\omega \mid \theta(\omega) \text{ is true}\})$.

To formulate the accumulated cost at the exit of the program, we define the *stopping time* $T : \Omega \to \mathbb{Z}^+ \cup \{\infty\}$ of a probabilistic program as a random variable on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of program traces:

$$T(\omega) \stackrel{\text{def}}{=} \inf\{n \in \mathbb{Z}^+ \mid \omega_n = \langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle\},$$

i.e., $T(\omega)$ is the number of evaluation steps before the trace ω reaches some termination configuration $\langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle$. We define the accumulated cost $A_T : \Omega \to \mathbb{R}$ with respect to the stopping time T as

$$A_T(\omega) \stackrel{\text{def}}{=} A_{T(\omega)}(\omega),$$

where $A_n : \Omega \to \mathbb{R}$ captures the accumulated cost at the n-th evaluation step for $n \in \mathbb{Z}^+$, which is defined as

$$A_n(\omega) \stackrel{\text{def}}{=} \alpha_n \text{ where } \omega_n = \langle \_, \_, \_, \alpha_n \rangle.$$

The m-th moment of the accumulated cost is given by the expectation $\mathbb{E}[A_T^m]$ with respect to $\mathbb{P}$.

4.2 Soundness of the Derivation System

Proofs for this section are included in the technical report [38].

The expected-potential method for moment analysis. We fix a degree $m \in \mathbb{N}$ and let $\mathcal{M}_{\mathcal{I}}^{(m)}$ be the m-th order moment semiring instantiated with the interval semiring $\mathcal{I}$. We now define $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential functions.

Definition 4.2. A measurable map $\phi : \Sigma \to \mathcal{M}_{\mathcal{I}}^{(m)}$ is said to be an *expected-potential function* if

(i) $\phi(\sigma) = \underline{1}$ if $\sigma = \langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle$, and

(ii) $\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim \mapsto(\sigma)}[\overrightarrow{([(\alpha' - \alpha)^k, (\alpha' - \alpha)^k])_{0 \le k \le m}} \otimes \phi(\sigma')]$ where $\sigma = \langle \_, \_, \_, \alpha \rangle, \sigma' = \langle \_, \_, \_, \alpha' \rangle$ for all $\sigma \in \Sigma$.

Intuitively, $\phi(\sigma)$ is an interval bound on the moments for the accumulated cost of the computation that *continues from* the configuration σ. We define $\mathbf{\Phi}_n$ and $\mathbf{Y}_n$, where $n \in \mathbb{Z}^+$, to be $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued random variables on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of the Markov-chain semantics as

$$\mathbf{\Phi}_n(\omega) \stackrel{\text{def}}{=} \phi(\omega_n), \mathbf{Y}_n(\omega) \stackrel{\text{def}}{=} \overrightarrow{\langle [A_n(\omega)^k, A_n(\omega)^k] \rangle_{0 \le k \le m}} \otimes \mathbf{\Phi}_n(\omega).$$

In the definition of $\mathbf{Y}_n$, we use $\otimes$ to compose the powers of the accumulated cost at step n and the expected potential function that stands for the moments of the accumulated cost for the rest of the computation.

Lemma 4.3. *By the properties of potential functions, we can prove that* $\mathbb{E}[\mathbf{Y}_{n+1} \mid \mathbf{Y}_n] \sqsubseteq \mathbf{Y}_n$ *almost surely, for all* $n \in \mathbb{Z}^+$.

We call $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ a *moment invariant*. Our goal is to establish that $\mathbb{E}[\mathbf{Y}_T] \sqsubseteq \mathbb{E}[\mathbf{Y}_0]$, i.e., the initial interval-valued potential $\mathbb{E}[\mathbf{Y}_0] = \mathbb{E}[\underline{1} \otimes \mathbf{\Phi}_0] = \mathbb{E}[\mathbf{\Phi}_0]$ brackets the higher moments of the accumulated cost $\mathbb{E}[\mathbf{Y}_T] = \mathbb{E}[\overrightarrow{\langle [A_T^k, A_T^k]\rangle_{0\le k\le m}} \otimes \underline{1}] = \overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]]\rangle_{0\le k\le m}}$.

Soundness. The soundness of the derivation system is proved with respect to the Markov-chain semantics. Let $\|\overrightarrow{\langle [a_k, b_k]\rangle_{0\le k\le m}}\|_\infty \stackrel{\text{def}}{=} \max_{0\le k\le m}\{\max\{|a_k|, |b_k|\}\}$.

Theorem 4.4. *Let $\langle \mathscr{D}, S_{\mathsf{main}}\rangle$ be a probabilistic program. Suppose $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; \underline{1}\}$, where $Q \in \mathcal{M}^{(m)}_{\mathcal{PI}}$ and the ends of the k-th interval in Q are polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$. Let $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ be the moment invariant extracted from the Markov-chain semantics with respect to the derivation of $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; \underline{1}\}$. If the following conditions hold:*

(i) $\mathbb{E}[T^{md}] < \infty$, and

(ii) there exists $C \ge 0$ such that for all $n \in \mathbb{Z}^+$, $\|\mathbf{Y}_n\|_\infty \le C \cdot (n+1)^{md}$ almost surely,

Then $\overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]]\rangle_{0\le k\le m}} \sqsubseteq \phi_Q(\lambda\_.0)$.

The intuitive meaning of $\overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]]\rangle_{0\le k\le m}} \sqsubseteq \phi_Q(\lambda\_.0)$ is that the moment $\mathbb{E}[A_T^k]$ of the accumulated cost upon program termination is bounded by the interval in the k^{th}-moment component of $\phi_Q(\lambda\_.0)$, where Q is the quantitative context and $\lambda\_.0$ is the initial state.

As we discussed in §2.2 and Ex. 2.7, the expected-potential method is *not* always sound for deriving bounds on higher moments for cost accumulators in probabilistic programs. The extra conditions Thm. 4.4(i) and (ii) impose constraints on the analyzed program and the expected-potential function, which allow us to reduce the soundness to the *optional stopping problem* from probability theory.

Optional stopping. Let us represent the moment invariant $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ as

$$\{\langle [L_n^{(0)}, U_n^{(0)}], [L_n^{(1)}, U_n^{(1)}], \cdots, [L_n^{(m)}, U_n^{(m)}]\rangle\}_{n\in\mathbb{Z}^+},$$

where $L_n^{(k)}, U_n^{(k)} : \Omega \to \mathbb{R}$ are real-valued random variables on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of the Markov-chain semantics, for $n \in \mathbb{Z}^+$, $0 \le k \le m$. We then have the observations below as direct corollaries of Lem. 4.3:

- For any k, the sequence $\{U_n^{(k)}\}_{n\in\mathbb{Z}^+}$ satisfies $\mathbb{E}[U_{n+1}^{(k)} \mid U_n^{(k)}] \le U_n^{(k)}$ almost surely, for all $n \in \mathbb{Z}^+$, and we want to find sufficient conditions for $\mathbb{E}[U_T^{(k)}] \le \mathbb{E}[U_0^{(k)}]$.
- For any k, the sequence $\{L_n^{(k)}\}_{n\in\mathbb{Z}^+}$ satisfies $\mathbb{E}[L_{n+1}^{(k)} \mid L_n^{(k)}] \ge L_n^{(k)}$ almost surely, for all $n \in \mathbb{Z}^+$, and we want to find sufficient conditions for $\mathbb{E}[L_T^{(k)}] \ge \mathbb{E}[L_0^{(k)}]$.

These kinds of questions can be reduced to *optional stopping problem* from probability theory. Recent research [1, 17, 35, 41] has used the *Optional Stopping Theorem* (OST) from probability theory to establish *sufficient* conditions for the soundness for analysis of probabilistic programs. However, the classic OST turns out to be *not* suitable for higher-moment analysis. We extend OST with a *new* sufficient condition that allows us to prove Thm. 4.4. We discuss the details of our extended OST in a companion paper [39]; in this work, we focus on the derivation system for central moments.

4.3 An Algorithm for Checking Soundness Criteria

Termination Analysis. We reuse our system for automatically deriving higher moments, which we developed in §3.3 and §3.4, for checking if $\mathbb{E}[T^{md}] < \infty$ (Thm. 4.4(i)). To reason about termination time, we assume that every program statement increments the cost accumulator by one. For example, the inference rule (Q-Sample) becomes

$$\frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \qquad Q = \langle 1, 1, \cdots, 1\rangle \otimes \mathbb{E}_{x\sim\mu_D}[Q']}{\Delta \vdash \{\Gamma; Q\}\ x \sim D\ \{\Gamma'; Q'\}}$$

However, we cannot apply Thm. 4.4 for the soundness of the termination-time analysis, because that would introduce a circular dependence. Instead, we use a different proof technique to reason about $\mathbb{E}[T^{md}]$, taking into account the *monotonicity* of the runtimes. Because upper-bound analysis of higher moments of runtimes has been studied by Kura et al. [24], we skip the details, but include them in the technical report [38].

Boundedness of $\|Y_n\|_\infty$, $n \in \mathbb{Z}^+$. To ensure that the condition in Thm. 4.4(ii) holds, we check if the analyzed program satisfies the *bounded-update* property: every (deterministic or probabilistic) assignment to a program variable updates the variable with a change bounded by a constant C almost surely. Then the absolute value of every program variable at evaluation step n can be bounded by $C \cdot n = O(n)$. Thus, a polynomial up to degree $\ell \in \mathbb{N}$ over program variables can be bounded by $O(n^\ell)$ at evaluation step n. As observed by Wang et al. [41], bounded updates are common in practice.

5 Tail-Bound Analysis

One application of our central-moment analysis is to bound the probability that the accumulated cost deviates from some given quantity. In this section, we sketch how we produce the tail bounds shown in Fig. 1(c).

There are a lot of *concentration-of-measure* inequalities in probability theory [12]. Among those, one of the most important is *Markov's inequality*:

Proposition 5.1. *If X is a nonnegative random variable and $a > 0$, then $\mathbb{P}[X \ge a] \le \frac{\mathbb{E}[X^k]}{a^k}$ for any $k \in \mathbb{N}$.*

Recall that Fig. 1(b) presents upper bounds on the raw moments $\mathbb{E}[tick] \le 2d + 4$ and $\mathbb{E}[tick^2] \le 4d^2 + 22d + 28$ for the cost accumulator *tick*. With Markov's inequality, we

derive the following tail bounds:

$$\mathbb{P}[tick \geq 4d] \leq \frac{\mathbb{E}[tick]}{4d} \leq \frac{2d+4}{4d} \xrightarrow{d\to\infty} \frac{1}{2}, \tag{8}$$

$$\mathbb{P}[tick \geq 4d] \leq \frac{\mathbb{E}[tick^2]}{(4d)^2} \leq \frac{4d^2+22d+28}{16d^2} \xrightarrow{d\to\infty} \frac{1}{4}. \tag{9}$$

Note that (9) provides an asymptotically more precise bound on $\mathbb{P}[tick \geq 4d]$ than (8) does, when d approaches infinity.

Central-moment analysis can obtain an even more precise tail bound. As presented in Fig. 1(b), our analysis derives $\mathbb{V}[tick] \leq 22d + 28$ for the variance of *tick*. We can now employ concentration inequalities that involve variances of random variables. Recall *Cantelli's inequality*:

Proposition 5.2. *If X is a random variable and $a > 0$, then $\mathbb{P}[X - \mathbb{E}[X] \geq a] \leq \frac{\mathbb{V}[X]}{\mathbb{V}[X]+a^2}$ and $\mathbb{P}[X - \mathbb{E}[X] \leq -a] \leq \frac{\mathbb{V}[X]}{\mathbb{V}[X]+a^2}$.*

With Cantelli's inequality, we obtain the following tail bound, where we assume $d \geq 2$:

$$\begin{aligned} &\mathbb{P}[tick \geq 4d] = \mathbb{P}[tick - (2d+4) \geq 2d-4] \\ &\leq \mathbb{P}[tick - \mathbb{E}[tick] \geq 2d-4] \leq \frac{\mathbb{V}[tick]}{\mathbb{V}[tick] + (2d-4)^2} \\ &= 1 - \frac{(2d-4)^2}{\mathbb{V}[tick] + (2d-4)^2} \leq \frac{22d+28}{4d^2+6d+44} \xrightarrow{d\to\infty} 0. \end{aligned} \tag{10}$$

For all $d \geq 15$, (10) gives a more precise bound than both (8) and (9). It is also clear from Fig. 1(c), where we plot the three tail bounds (8), (9), and (10), that the asymptotically most precise bound is the one obtained via variances.

In general, for higher central moments, we employ *Chebyshev's inequality* to derive tail bounds:

Proposition 5.3. *If X is a random variable and $a > 0$, then $\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathbb{E}[(X-\mathbb{E}[X])^{2k}]}{a^{2k}}$ for any $k \in \mathbb{N}$.*

In our experiments, we use Chebyshev's inequality to derive tail bounds from the fourth central moments. We will show in Fig. 8 that these tail bounds can be more precise than those obtained from both raw moments and variances.

6 Implementation and Experiments

Implementation. Our tool is implemented in OCaml, and consists of about 5,300 LOC. The tool works on imperative arithmetic probabilistic programs using a CFG-based IR [37]. The language supports recursive functions, continuous distributions, unstructured control-flow, and local variables. To infer the bounds on the central moments for a cost accumulator in a program, the user needs to specify the order of the analyzed moment, and a maximal degree for the polynomials to be used in potential-function templates. Using APRON [20], we implemented an interprocedural numeric analysis to infer the logical contexts used in the derivation. We use the off-the-shelf solver Gurobi [26] for LP solving.

Evaluation setup. We evaluated our tool to answer the following three research questions:

1. How does the raw-moment inference part of our tool compare to existing techniques for expected-cost bound analysis [29, 41]?
2. How does our tool compare to the state of the art in tail-probability analysis (which is based only on higher *raw* moments [24])?
3. How scalable is our tool? Can it analyze programs with many recursive functions?

For the first question, we collected a broad suite of challenging examples from related work [24, 29, 41] with different loop and recursion patterns, as well as probabilistic branching, discrete sampling, and continuous sampling. Our tool achieved comparable precision and efficiency with the prior work on expected-cost bound analysis [29, 41]. The details are included in the technical report [38].

For the second question, we evaluated our tool on the complete benchmarks from Kura el al. [24]. We also conducted a case study of a timing-attack analysis for a program provided by DARPA during engagements of the STAC program [14], where central moments are more useful than raw moments to bound the success probability of an attacker. We include the case study in the technical report [38].

For the third question, we conducted case studies on two sets of synthetic benchmark programs:

- coupon-collector programs with N coupons ($N \in [1, 10^3]$), where each program is implemented as a set of tail-recursive functions, each of which represents a state of coupon collection, i.e., the number of coupons collected so far; and
- random-walk programs with N consecutive one-dimensional random walks ($N \in [1, 10^3]$), each of which starts at a position that equals the number of steps taken by the previous random walk to reach the ending position (the origin). Each program is implemented as a set of non-tail-recursive functions, each of which represents a random walk. The random walks in the same program can have different transition probabilities.

The largest synthetic program has nearly 16,000 LOC. We then ran our tool to derive an upper bound on the fourth (resp., second) central moment of the runtime for each coupon-collector (resp., random-walk) program.

The experiments were performed on a machine with an Intel Core i7 3.6GHz processor and 16GB of RAM under macOS Catalina 10.15.7.

Results. Some of the evaluation results to answer the second research question are presented in Tab. 1. The programs (1-1) and (1-2) are coupon-collector problems with a total of two and four coupons, respectively. The programs (2-1) and (2-2) are one-dimensional random walks with integer-valued and real-valued coordinates, respectively. We omit three other programs here but include the full results in the technical report [38]. The table contains the inferred upper bounds on the moments for runtimes of these programs, and

Table 1. Upper bounds on the raw/central moments of runtimes, with comparison to Kura et al. [24]. "T/O" stands for timeout after 30 minutes. "N/A" means that the tool is not applicable. "-" indicates that the tool fails to infer a bound. Entries with more precise results or less analysis time are marked in bold. Full results are included in the technical report [38].

program	moment	this work		Kura et al. [24]	
		upper bnd.	time (sec)	upper bnd.	time (sec)
(1-1)	2^{nd} raw	**201**	0.019	**201**	**0.015**
	3^{rd} raw	**3829**	**0.019**	**3829**	0.020
	4^{th} raw	**90705**	**0.023**	**90705**	0.027
	2^{nd} central	**32**	**0.029**	N/A	N/A
	4^{th} central	**9728**	**0.058**	N/A	N/A
(1-2)	2^{nd} raw	**2357**	1.068	3124	**0.037**
	3^{rd} raw	**148847**	1.512	171932	**0.062**
	4^{th} raw	**11285725**	1.914	12049876	**0.096**
	2^{nd} central	**362**	**3.346**	N/A	N/A
	4^{th} central	**955973**	**9.801**	N/A	N/A
(2-1)	2^{nd} raw	**2320**	**0.016**	**2320**	11.380
	3^{rd} raw	**691520**	**0.018**	-	16.056
	4^{th} raw	**340107520**	**0.021**	-	23.414
	2^{nd} central	**1920**	**0.026**	N/A	N/A
	4^{th} central	**289873920**	**0.049**	N/A	N/A
(2-2)	2^{nd} raw	**8375**	**0.022**	**8375**	38.463
	3^{rd} raw	**1362813**	**0.028**	-	73.408
	4^{th} raw	**306105209**	**0.035**	-	141.072
	2^{nd} central	**5875**	**0.029**	N/A	N/A
	4^{th} central	**447053126**	**0.086**	N/A	N/A

the running times of the analyses. We compared our results with Kura et al.'s inference tool for raw moments [24]. Our tool is as precise as, and sometimes more precise than the prior work on all the benchmark programs. Meanwhile, our tool is able to infer an upper bound on the raw moments of degree up to four on all the benchmarks, while the prior work reports failure on some higher moments for the random-walk programs. In terms of efficiency, our tool completed each example in less than 10 seconds, while the prior work took more than a few minutes on some programs. One reason why our tool is more efficient is that we always reduce higher-moment inference with non-linear polynomial templates to efficient LP solving, but the prior work requires semidefinite programming (SDP) for polynomial templates.

Besides raw moments, our tool is also capable of inferring upper bounds on the central moments of runtimes for the benchmarks. To evaluate the quality of the inferred central moments, Fig. 8 plots the upper bounds of tail probabilities on runtimes T obtained by Kura et al. [24], and those by our central-moment analysis. Specifically, the prior work uses Markov's inequality (Prop. 5.1), while we are also able to apply Cantelli's and Chebyshev's inequality (Props. 5.2 and 5.3) with central moments. Our tool outperforms the prior work on programs (1-1) and (1-2), and derives better

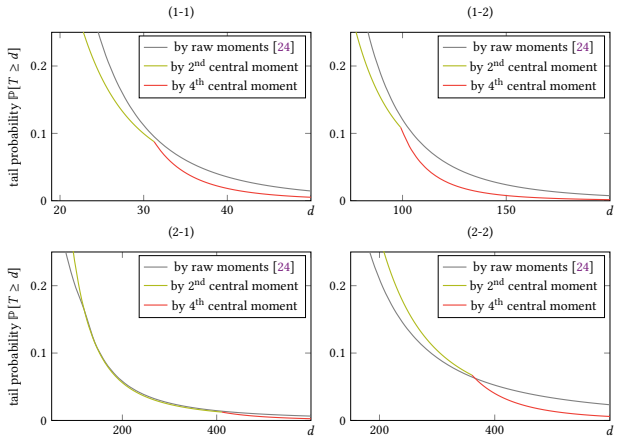
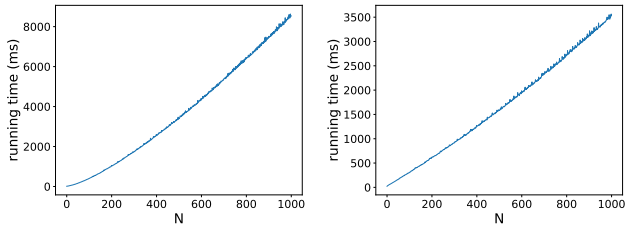


Figure 8. Upper bound of the tail probability $\mathbb{P}[T \geq d]$ as a function of d, with comparison to Kura et al. [24]. Each gray line is the minimum of tail bounds obtained from the raw moments of degree up to four inferred by Kura et al. [24]. Green lines and red lines are the tail bounds given by 2^{nd} and 4^{th} central moments inferred by our tool, respectively. We include the plots for other programs in the technical report [38].

tail bounds when d is large on program (2-2), while it obtains similar curves on program (2-1).

Scalability. In Fig. 9, we demonstrate the running times of our tool on the two sets of synthetic benchmark programs; Fig. 9a plots the analysis times for coupon-collector programs as a function of the independent variable N (the total number of coupons), and Fig. 9b plots the analysis times for random-walk programs as a function of N (the total number of random walks). The evaluation statistics show that our tool achieves good scalability in both case studies: the runtime is almost a linear function of the program size, which is the number of recursive functions for both case studies. Two reasons why our tool is scalable on the two sets of programs are (i) our analysis is compositional and uses function summaries to analyze function calls, and (ii) for a fixed set of templates and a fixed diameter of the call graph, the number of linear constraints generated by our tool grows linearly with the size of the program, and the LP solvers available nowadays can handle large problem instances efficiently.

Discussion. Higher central moments can also provide more information about the *shape* of a distribution, e.g., the *skewness* (i.e., $\frac{\mathbb{E}[(T-\mathbb{E}[T])^3]}{(\mathbb{V}[T])^{3/2}}$) indicates how lopsided the distribution of T is, and the *kurtosis* (i.e., $\frac{\mathbb{E}[(T-\mathbb{E}[T])^4]}{(\mathbb{V}[T])^2}$) measures the heaviness of the tails of the distribution of T. We used our tool to analyze two variants of the random-walk program (2-1). The two random walks have different transition



(a) Coupon Collector **(b)** Random Walk

Figure 9. Running times of our tool on two sets of synthetic benchmark programs. Each figure plots the runtimes as a function of the size of the analyzed program.

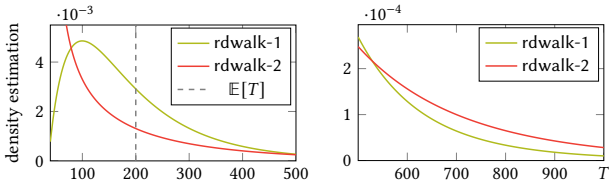


Figure 10. Density estimation for the runtime T of two variants rdwalk-1 and rdwalk-2 of (2-1).

probabilities and step length, but they have the same expected runtime $\mathbb{E}[T]$. Tab. 2 presents the skewness and kurtosis derived from the moment bounds inferred by our tool. A positive skew indicates that the mass of the distribution is concentrated on the *left*, and larger skew means the concentration is more *left*. A larger kurtosis, on the other hand, indicates that the distribution has *fatter* tails. Therefore, as the derived skewness and kurtosis indicate, the distribution of the runtime T for rdwalk-2 should be more left-leaning and have fatter tails than the distribution for rdwalk-1. Density estimates for the runtime T, obtained by simulation, are shown in Fig. 10.

Table 2. Skewness & kurtosis.

program	skewness	kurtosis
rdwalk-1	2.1362	10.5633
rdwalk-2	2.9635	17.5823

Our tool can also derive *symbolic* bounds on higher moments. The table below presents the inferred upper bounds on the variances for the random-walk benchmarks, where we replace the concrete inputs with symbolic pre-conditions.

program	pre-condition	upper bnd. on the variance
(2-1)	$x \geq 0$	$1920x$
(2-2)	$x \geq 0$	$2166.6667x + 1541.6667$

7 Conclusion

We have presented an automatic central-moment analysis for probabilistic programs that support general recursion and continuous distributions, by deriving symbolic interval bounds on higher raw moments for cost accumulators. We have proposed *moment semirings* for compositional reasoning and *moment-polymorphic recursion* for interprocedural analysis. We have proved soundness of our technique using a recent extension to the Optional Stopping Theorem. The effectiveness of our technique has been demonstrated with our prototype implementation and the analysis of a broad suite of benchmarks, as well as a tail-bound analysis.

In the future, we plan to go beyond arithmetic programs and add support for more datatypes, e.g., Booleans and lists. We will also work on other kinds of uncertain quantities for probabilistic programs. Another research direction is to apply our analysis to higher-order functional programs.

Acknowledgments

This article is based on research supported, in part, by a gift from Rajiv and Ritu Batra; by ONR under grants N00014-17-1-2889 and N00014-19-1-2318; by DARPA under AA contract FA8750-18-C0092; and by the NSF under SaTC award 1801369, SHF awards 1812876 and 2007784, and CAREER award 1845514. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the sponsoring agencies.

We thank Jason Breck and John Cyphert for formulating, and providing us with an initial formalization of the problem of timing-attack analysis.

References

[1] Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *Computer Aided Verif. (CAV'16)*. https://doi.org/10.1007/978-3-319-41528-4_3

[2] Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *European Symp. on Programming (ESOP'18)*. https://doi.org/10.1007/978-3-319-89884-1_5

[3] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal Certification of Code-Based Cryptographic Proofs. In *Princ. of Prog. Lang. (POPL'09)*. https://doi.org/10.1145/1594834.1480894

[4] Ezio Bartocci, Laura Kovács, and Miroslav Stankovič. 2019. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops. In *Automated Tech. for Verif. and Analysis (ATVA'19)*. https://doi.org/10.1007/978-3-030-31784-3_15

[5] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2018. How long, O Bayesian network, will I sample thee?. In *European Symp. on Programming (ESOP'18)*. https://doi.org/10.1007/978-3-319-89884-1_7

[6] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *Int. Conf. on Functional Programming (ICFP'16)*. https://doi.org/10.1145/2951913.2951942

[7] Olivier Bouissou, Eric Goubault, Sylvie Putot, Aleksandar Chakarov, and Sriram Sankaranarayanan. 2016. Uncertainty Propagation Using Probabilistic Affine Forms and Concentration of Measure Inequalities. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'16)*. https://doi.org/10.1007/978-3-662-49674-9_13

[8] Quentin Carbonneaux, Jan Hoffmann, Thomas Reps, and Zhong Shao. 2017. Automated Resource Analysis with Coq Proof Objects. In *Computer Aided Verif. (CAV'17)*. https://doi.org/10.1007/978-3-319-63390-9_4

[9] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verif. (CAV'13)*. 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 *Computer Aided Verif. (CAV'16)*. https://doi.org/10.1007/978-3-319-41528-4_1

[11] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. In *Princ. of Prog. Lang. (POPL'16)*. https://doi.org/10.1145/2837614.2837639

[12] Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press. https://doi.org/10.1017/CBO9780511581274

[13] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Princ. of Prog. Lang. (POPL'15)*. https://doi.org/10.1145/2676726.2677001

[14] Dustin Fraze. 2015. Space/Time Analysis for Cybersecurity (STAC). Available on https://www.darpa.mil/program/space-time-analysis-for-cybersecurity.

[15] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521 (May 2015). https://doi.org/10.1038/nature14541

[16] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic Programming. In *Future of Softw. Eng. (FOSE'14)*. https://doi.org/10.1145/2593882.2593900

[17] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. 2019. Aiming Low Is Harder: Induction for Lower Bounds in Probabilistic Program Verification. *Proc. ACM Program. Lang.* 4, POPL (December 2019). https://doi.org/10.1145/3371105

[18] Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics. In *Asian Symp. on Prog. Lang. and Systems (APLAS'10)*. https://doi.org/10.1007/978-3-642-17164-2_13

[19] Martin Hofmann and Steffen Jost. 2003. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *Princ. of Prog. Lang. (POPL'03)*. https://doi.org/10.1145/604131.604148

[20] Bertrand Jeannet and Antoine Miné. 2009. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *Computer Aided Verif. (CAV'09)*. https://doi.org/10.1007/978-3-642-02658-4_52

[21] Xia Jiang and Gregory F. Cooper. 2010. A Bayesian spatio-temporal method for disease outbreak detection. *J. American Medical Informatics Association* 17, 4 (July 2010). https://doi.org/10.1136/jamia.2009.000356

[22] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run–Times of Probabilistic Programs. In *European Symp. on Programming (ESOP'16)*. https://doi.org/10.1007/978-3-662-49498-1_15

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

[24] Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probability for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'19)*. https://doi.org/10.1007/978-3-030-17465-1_8

[25] Zhifei Li and Jason Eisner. 2009. First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests. In *Empirical Methods in Natural Language Processing (EMNLP'09)*. https://dl.acm.org/doi/10.5555/1699510.1699517

[26] Gurobi Optimization LLC. 2020. Gurobi Optimizer Reference Manual. Available on https://www.gurobi.com.

[27] Guido Manfredi and Yannick Jestin. 2016. An Introduction to ACAS Xu and the Challenges Ahead. In *Digital Avionics Systems Conference (DASC'16)*. https://doi.org/10.1109/DASC.2016.7778055

[28] Annabelle K. McIver and Carroll C. Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer Science+Business Media, Inc. https://doi.org/10.1007/b138392

[29] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'18)*. https://doi.org/10.1145/3192366.3192394

[30] Van Chan Ngo, Mario Dehesa-Azuara, Matthew Fredrikson, and Jan Hoffmann. 2017. Verifying and Synthesizing Constant-Resource Implementations with Types. In *Symp. on Sec. and Privacy (SP'17)*. https://doi.org/10.1109/SP.2017.53

[31] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Logic in Computer Science (LICS'16)*. https://doi.org/10.1145/2933575.2935317

[32] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. https://dl.acm.org/doi/book/10.5555/528623

[33] Reuven Y. Rubinstein and Dirk P. Kroese. 2016. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc. https://doi.org/10.1002/9781118631980

[34] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths. In *Prog. Lang. Design and Impl. (PLDI'13)*. https://doi.org/10.1145/2491956.2462179

[35] Anne Schreuder and Luke Ong. 2019. Polynomial Probabilistic Invariants and the Optional Stopping Theorem. https://arxiv.org/abs/1910.12634

[36] Robert Endre Tarjan. 1985. Amortized Computational Complexity. *SIAM J. Algebraic Discrete Methods* 6, 2 (August 1985). https://doi.org/10.1137/0606031

[37] Di Wang, Jan Hoffmann, and Thomas Reps. 2018. PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'18)*. https://doi.org/10.1145/3192366.3192408

[38] Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. https://arxiv.org/abs/2001.10150

[39] Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Expected-Cost Analysis for Probabilistic Programs and Semantics-Level Adaption of Optional Stopping Theorems. https://arxiv.org/abs/2103.16105

[40] Di Wang, David M. Kahn, and Jan Hoffmann. 2020. Raising Expectations: Automating Expected Cost Analysis with Types. *Proc. ACM Program. Lang.* 4, ICFP (August 2020). https://doi.org/10.1145/3408992

[41] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost Analysis of Nondeterministic Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'19)*. https://doi.org/10.1145/3314221.3314581