Automated Concentration Bound Analysis for Probabilistic Recurrence Relations

Hongfei Fu, Yican Sun, Krishnendu Chatterjee, Amir Goharshady

Automated Concentration Bound Analysis for Probabilistic Recurrence Relations

Beating Karp's Cookbook after Three Decades

HONGFEI FU*, Shanghai Jiao Tong University
YICAN SUN, Peking University
KRISHNENDU CHATTERJEE, IST Austria
AMIR KAFSHDAR GOHARSHADY, The Hong Kong University of Science and Technology

Probabilistic recurrence relations (PRRs) are a standard formalism to analyze the runtime of randomized algorithms. In this work, we consider the classical problem of obtaining concentration bounds on the runtime of randomized algorithms modeled as PRRs. More specifically, given a time limit κ, we study the probability $\Pr[T \geq \kappa]$ that the runtime T of the algorithm reaches or exceeds the time limit. The key research question in concentration bound analysis is to find an upper-bound function $\mathit{upper}(n) \geq \Pr[T \geq \kappa]$ such that $\mathit{upper}(n)$ converges rapidly to zero as n tends to infinity. The classical approach to this problem is the "cookbook" method by Karp (J. ACM 1994). For the past three decades, finding an approach to improve Karp's classical bounds has been a major open problem.

In this work, we first demonstrate a method to obtain such upper-bound functions from a specific family of supermartingale functions through the well-known Optional Stopping Theorem (OST). We then provide automated algorithmic approaches to synthesize such supermartingale functions in exponential and polynomial form. Our algorithms are based on the Taylor expansion and tight approximations of the expectation of moments that involve logarithmic terms. Compared with Karp's method, our algorithmic approaches find significantly tighter concentration bounds for several classical randomized algorithms such as QuickSort, QuickSelect and DiameterComputation. Moreover, our algorithmic approaches can also find concentration bounds for several randomized algorithms in computational geometry which adopt techniques such as randomized incremental construction. To the best of our knowledge, no previous method was applicable in these cases. Finally, we present experimental results demonstrating the practical versatility and efficiency of our approach.

1 INTRODUCTION

Randomized algorithms. Randomized algorithms [Karp 1991; Motwani and Raghavan 1995] are a class of algorithms that adopt randomization techniques, such as randomized pivoting [Hoare 1961a,b] and randomized incremental construction [Clarkson et al. 1993]. As a central topic in theoretical computer science, many randomized algorithms have been devised in various application domains, such as sorting and selection [Hoare 1961a,b], computational geometry [Clarkson et al. 1993], distributed systems [Motwani and Raghavan 1995, Chapter 3][Kleinberg and Tardos 2006, Chapter 13], graph theory [Karger 1993], and data streaming [Morris 1978]. Their advantage in comparison with deterministic algorithms is two-fold: first, randomized algorithms typically have simpler design while their expected performance often matches or outperforms the deterministic counterparts (e.g. QuickSort [Hoare 1961a] vs. MergeSort [Cormen et al. 2009, Chapter 2]); second, for various problems, especially when it comes to distributed settings, deterministic algorithms are impossible and the only solutions are through randomization [Motwani and Raghavan 1995, Chapter 3][Kleinberg and Tardos 2006, Chapter 13][Morris 1978].

*Due to different academic cultural standards, Mainland Chinese co-authors are listed before other co-authors (Austria, Hong Kong). No comparison of contribution between the two groups is implied. Hongfei Fu and Yican Sun contributed equally.

Probabilistic recurrence relations. Probabilistic Recurrence Relations (PRRs) are the probabilistic extension of classical recurrence relations and are widely used in the runtime analysis of recursive randomized algorithms. The probabilities and random variables in these recurrences respectively model random samplings and the random size of an instance passed to the recursive procedures in a randomized algorithm. The runtime of various randomized algorithms, such as QuickSort [Hoare 1961a], QuickSelect [Hoare 1961b], and DiameterComputation [Motwani and Raghavan 1995, Chapter 9], can be captured by PRRs. The runtime complexity analysis of a randomized algorithm is therefore reduced to that of its corresponding PRR. Thus, formal analysis of PRRs has become a central and classical problem in the analysis of randomized algorithms and has also received a renewed attention in recent years [Bazzi and Mitter 2003; Chatterjee et al. 2017a; Chaudhuri and Dubhashi 1997; Dubhashi and Panconesi 2009; Karp 1994; McDiarmid and Hayward 1996; Motwani and Raghavan 1995; Tassarotti 2017; Tassarotti and Harper 2018]. In formal analysis of PRRs, the two fundamental problems are *Expected Runtime Analysis* and *Concentration Bound Analysis*.

Expected runtime analysis. Expected runtime analysis aims to derive tight upper bounds on the expected runtime complexity of a given PRR [Motwani and Raghavan 1995]. The expected runtime is a key performance criterion that examines for how long the randomized algorithm will execute in expectation. This is important since many randomized algorithms perform poorly in the worst case but are efficient on average. For example, the well-known QuickSort algorithm [Hoare 1961a] takes $\Omega(n^2)$ time in the worst case but has an expected runtime of $O(n \log n)$. Compared with the classical runtime analysis of non-probabilistic recurrences that can typically be solved using the Master theorem [Cormen et al. 2009, Chapter 4][Akra and Bazzi 1998], expected runtime analysis of PRRs is much more intricate and there is no straightforward solution similar to Master theorem, except for certain special cases [Bazzi and Mitter 2003]. Instead, the expected runtime of PRRs is obtained either through the traditional methods of Harmonic series and generating functions via computer algebra [Flajolet et al. 1991; Zimmermann and Zimmermann 1989] or through automated guess-and-check algorithms and approximation of integrals [Chatterjee et al. 2017a].

Concentration bound analysis. Concentration bound analysis [Dubhashi and Panconesi 2009; Karp 1991] considers upper bounds on the probability that a given PRR does not terminate within a prescribed time limit. The goal is to show that this probability converges rapidly to zero as the time limit tends to infinity. The most classical approach to this problem is that of [Karp 1994]. Since then, finding automated approaches for obtaining tighter bounds than Karp's method has been a key open problem. Moreover, ad-hoc methods have been devised to improve these bounds over a specific recurrence, such as QuickSort [McDiarmid and Hayward 1996].

Comparison of the two analyses. Compared with the expected runtime analysis, concentration bound analysis is a more refined problem. It examines the probability that the runtime of a given PRR deviates significantly from its expected value. Unsurprisingly, methods used for concentration bound analysis are typically much more involved than expected runtime analysis. For instance, expected runtime analysis allows a template-based approach through guess-and-check functions [Chatterjee et al. 2017a], while there is no such template-based approach for the concentration bound analysis in the literature. Furthermore, in some cases, such as QuickSort, it is possible to manually derive the exact expected runtime of probabilistic recurrences by applying the generating-function method, but no such approach is known when it comes to concentration bounds.

Karp's cookbook. The classical method to obtain concentration bounds was proposed in [Karp 1994] where a "cookbook" formula similar to the Master theorem is proposed. This approach requires the solution to the expected runtime of a deterministic version from the PRR of concern, and then outputs a coarse concentration bound for the original PRR by a fixed mathematical formula. Based on Karp's method, the work [Chaudhuri and Dubhashi 1997] proposed an approach that weakens several conditions required in [Karp 1994] and obtains bounds that are slightly

worse than [Karp 1994]. For the QuickSort algorithm, the work [McDiarmid and Hayward 1996] performed an involved ad-hoc manual analysis to derive the asymptotically optimal concentration bound. Recently, there has been a renewed interest in this problem. The work [Tassarotti and Harper 2018] mechanized the proof for concentration bound of QuickSort in the interactive theorem prover Coq [Bertot and Castéran 2004].

Challenges and gaps. Although concentration bound analysis for PRRs is a classical problem that has been studied for around thirty years, there are still several key challenges. First, while the bounds in [Karp 1994] were known to be coarse (as exemplified by [McDiarmid and Hayward 1996] and the results of this work), it has been a key open problem for almost three decades to find general approaches that (a) are automated and (b) can significantly improve these classical results. All previous improvements lacked automation and were obtained on a fixed benchmark, i.e. they consider only a specific single recurrence, such as QuickSort. Second, in several application domains, such as computational geometry, the preprocessing time in each recurrence equation is randomized rather than deterministic, and the analysis involves an acyclic system of PRRs rather than a single equation. Unfortunately, existing approaches all assumed that the preprocessing time is deterministic and are therefore inapplicable. Third, all existing results consider concentration bounds that are *exponentially* decreasing and other forms of concentration bounds have not been thoroughly investigated.

Our contribution. In this work, we consider the problem of automated concentration bound analysis for PRRs. Our main contributions are as follows:

(1) We propose a sound approach based on the novel concept of Taylor Supermartingale Function Sequences (TSFS). We show that the existence of such functions for a given PRR leads to a concentration bound on its runtime. The main theoretical tools we use here are Markov's inequality (Theorem 3.2) and Optional Stopping Theorem (Theorem 3.3).
(2) We develop algorithmic approaches to synthesize a TSFS in exponential and polynomial form for a given PRR. Our main tools in this synthesis process are Taylor expansion and tight approximations of expectation of moments with logarithmic terms.
(3) Compared with Karp's cookbook method, our algorithmic approaches derive much tighter concentration bounds over classical randomized algorithms such as QuickSort, QuickSelect, RandomizedSearch and DiameterComputation. For example, in case of QuickSelect, we find a bound for $\Pr[T \geq 20 \cdot n]$ that is 10^6 times tighter. Moreover, for QuickSort and RandomizedSearch, our bounds become arbitrarily tighter than Karp's as n grows. See Section 6 for details.
(4) Furthermore, our algorithmic approaches can handle randomized algorithms that are beyond the scope of [Karp 1994] and other existing methods such as [Chaudhuri and Dubhashi 1997; Tassarotti 2017]. To be more precise, our algorithmic approaches can handle PRRs with randomized preprocessing phase in a randomized algorithm. In contrast, other approaches require the preprocessing phase to be deterministic. Moreover, our algorithmic approaches can handle *systems* of probabilistic recurrences such as those arising from the SeidelLP [Seidel 1991] and the SmallestEnclosingDisk problem [Welzl 1991]. To the best of our knowledge, no existing method can handle concentration bounds for systems of probabilistic recurrences.

Key novel aspects. The key novelties of our approach are two-fold. First, we establish the novel notion of Taylor Supermartingale Function Sequences (TSFS) for solving concentration bounds of PRRs. Second, to synthesize a TSFS in exponential or polynomial form, we consider the novel idea of Taylor expansion and over-approximation of expectation of moments with logarithmic terms in our algorithmic approaches. The novel aspect is that while these techniques are quite simple, they surprisingly lead to first significant improvements in classical decades-old bounds for fundamental and well-studied randomized algorithms. Our approach can even handle classical

randomized algorithms that are beyond the reach of all previous methods.

2 PRELIMINARIES

We first review some necessary concepts in probability theory in Section 2.1. Then, in Section 2.2, we introduce *Systems of Probabilistic Recurrence Relations* (SPRRs) and formally define our problem.

2.1 Background Notions in Probability Theory

Throughout the paper, we denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the sets of non-negative integers, integers and real numbers, respectively. Below, we present only a brief description of the concepts needed for this work. We refer to standard textbooks (e.g. [Billingsley 1995; Rosenthal 2006; Williams 1991]) for a more detailed treatment.

Discrete probability distributions. A *discrete probability distribution* over a countable set U is a function $\eta : U \to [0, 1]$ such that $\sum_{u \in U} \eta(u) = 1$. The *support* of η is defined as $supp(\eta) := \{u \in U \mid \eta(u) > 0\}$. Informally, $\eta(u)$ is the probability that a sampling from the distribution η returns u.

Probability spaces. A *probability space* is a triple $(\Omega, \mathcal{F}, \Pr)$ where Ω is a non-empty set (called the *sample space*), $\mathcal{F}$ is a *σ-algebra* over Ω (i.e. a collection of subsets of Ω that contains the empty set $\emptyset$ and is closed under complementation and countable union) and $\Pr$ is a *probability measure* on $\mathcal{F}$, i.e. a function $\Pr\colon \mathcal{F} \to [0, 1]$ such that (i) $\Pr(\Omega) = 1$, and (ii) for all pairwise-disjoint set-sequences $A_1, A_2, \cdots \in \mathcal{F}$ (i.e. $A_i \cap A_j = \emptyset$ whenever $i \neq j$) it holds that $\sum_{i=1}^{\infty} \Pr(A_i) = \Pr\left(\bigcup_{i=1}^{\infty} A_i\right)$. Elements of $\mathcal{F}$ are conventionally called *events*. An event $A \in \mathcal{F}$ holds *almost-surely* (a.s.) if $\Pr(A) = 1$.

Random variables. A *random variable* X from a probability space $(\Omega, \mathcal{F}, \Pr)$ is an $\mathcal{F}$-measurable function $X\colon \Omega \to \mathbb{R} \cup \{-\infty, +\infty\}$, i.e. a function such that for all $d \in \mathbb{R} \cup \{-\infty, +\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) < d\}$ lies in $\mathcal{F}$.

Expectation. The *expected value* of a random variable X from a probability space $(\Omega, \mathcal{F}, \Pr)$, denoted by $\mathbb{E}[X]$, is defined as the Lebesgue integral of X w.r.t. the probability measure $\Pr$, i.e., $\mathbb{E}[X] := \int X \,\mathrm{d}\Pr$. The precise definition of Lebesgue integral is rather technical and therefore omitted. See [Williams 1991, Chapter 5] for a formal definition. In the case that $ran\, X = \{d_0, d_1, \ldots\}$ ($ran\, X$ is the range of the random variable X when viewed as a function) is countable, then we have $\mathbb{E}[X] = \sum_{k=0}^{\infty} d_k \cdot \Pr(X = d_k)$.

Filtrations. A *filtration* of a probability space $(\Omega, \mathcal{F}, \Pr)$ is an infinite sequence $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ of sigma-algebras over Ω such that $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$ for all $n \in \mathbb{N}$. Intuitively, a filtration models the information $\mathcal{F}_n$ available at a specific time point $n \geq 0$.

Conditional expectation. Let X be any random variable from a probability space $(\Omega, \mathcal{F}, \Pr)$ such that $\mathbb{E}[|X|] < \infty$ (i.e. X is integrable). Then, given any sub-sigma-algebra $\mathcal{G} \subseteq \mathcal{F}$, there exists a random variable from $(\Omega, \mathcal{F}, \Pr)$, denoted by $\mathbb{E}[X \mid \mathcal{G}]$, such that:

(E1) $\mathbb{E}[X \mid \mathcal{G}]$ is $\mathcal{G}$-measurable, and
(E2) $\mathbb{E}[|\mathbb{E}[X \mid \mathcal{G}]|] < \infty$, and
(E3) for all $A \in \mathcal{G}$, we have $\int_A \mathbb{E}[X \mid \mathcal{G}] \,\mathrm{d}\Pr = \int_A X \,\mathrm{d}\Pr$.

The random variable $\mathbb{E}[X \mid \mathcal{G}]$ is called the *conditional expectation* of X given $\mathcal{G}$, and is a.s. unique in the sense that if Y is another random variable satisfying (E1)–(E3), then $\Pr(Y = \mathbb{E}[X \mid \mathcal{G}]) = 1$. We refer to [Williams 1991, Chapter 9] for details. The intuition is that $\mathbb{E}[X \mid \mathcal{G}]$ represents the expectation of X when assuming the information in $\mathcal{G}$.

Discrete-time stochastic processes. A *discrete-time stochastic process* is an infinite sequence $\Gamma = \{X_n\}_{n \in \mathbb{N}}$ of random variables where the random variables X_n are all from the same probability space $(\Omega, \mathcal{F}, \Pr)$. The process Γ is *adapted to* a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ if for all $n \in \mathbb{N}$, X_n is $\mathcal{F}_n$-measurable. Intuitively, the random variable X_n measures the value at the n-th step of the process Γ.

Stopping times. Given a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ with respect to a probability space $(\Omega, \mathcal{F}, \Pr)$, a *stopping time* ρ is a random variable $\rho\colon \Omega \to \{0, 1, 2, 3, \cdots, \infty\}$, such that for every $n \in \{0, 1, 2, \cdots, \infty\}$, the event $\{\omega \mid \rho(\omega) \leq n\}$ is measurable in $\mathcal{F}_n$. In other words, $\{\omega \mid \rho(\omega) \leq n\} \in \mathcal{F}_n$.

Martingales and supermartingales. A discrete-time stochastic process $\Gamma = \{X_n\}_{n\in\mathbb{N}}$ adapted to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ is a *martingale* (resp. *supermartingale*) if for every $n \in \mathbb{N}$, $\mathbb{E}[|X_n|] < \infty$ and it holds a.s. that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] = X_n$ (resp. $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \leq X_n$). Intuitively, a martingale (resp. supermartingale) is a discrete-time stochastic process in which for an observer who has seen the values of $X_0, \ldots, X_n$, the expected value at the next step, i.e. $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n]$, is equal to (resp. no more than) the last observed value X_n. Also, note that in a martingale, the observed values for $X_0, \ldots, X_{n-1}$ do not matter given that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] = X_n$. In contrast, in a supermartingale, the only requirement is that $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \leq X_n$ and hence $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n]$ may depend on $X_0, \ldots, X_{n-1}$. Also, note that $\mathcal{F}_n$ might contain more information than just the observations of X_i's. We refer to [Williams 1991, Chapter 10] for a deeper treatment.

Example 2.1. Consider an unbiased and discrete random walk, in which we start at a position X_0, and at each second walk one step to either left or right with equal probability. Let X_n denote our position after n seconds. It is easy to verify that $\mathbb{E}(X_{n+1}|X_0, \ldots, X_n) = \frac{1}{2}(X_n - 1) + \frac{1}{2}(X_n + 1) = X_n$. Hence, this random walk is a martingale. Note that by definition, every martingale is also a supermartingale. As another example, consider the classical gambler's ruin: a gambler starts with Y_0 dollars of money and bets continuously until he loses all of his money. If the bets are unfair, i.e. the expected value of his money after a bet is less than its expected value before the bet, then the sequence $\{Y_n\}_{n\in\mathbb{N}_0}$ is a supermartingale. In this case, Y_n is the gambler's total money after n bets. On the other hand, if the bets are fair, then $\{Y_n\}_{n\in\mathbb{N}_0}$ is a martingale. See Appendix A for another illustrative example of martingales. □

2.2 Systems of Probabilistic Recurrence Relations

In this work, we consider the formal analysis of *Systems of Probabilistic Recurrence Relations* (SPRRs). An SPRR is a collection of probabilistic recurrence equations that describe the running-time behavior of a randomized algorithm. Each recurrence equation corresponds to a procedure in the algorithm.

Systems of Probabilistic Recurrence Relations (SPRRs). An SPRR Δ is a finite collection of probabilistic recurrence relations $eq_1, \ldots, eq_m$, which models the running time of a randomized algorithm with m procedures, for which each recurrence equation eq_r $(1 \leq r \leq m)$ corresponds to a procedure in the algorithm (which we shall denote by $proc_r$). Each eq_r $(1 \leq r \leq m)$ takes the following form:

$$T_r(n) = \begin{cases} Cost_r(n) + \left[\sum_{j=1}^{S_r} T_r(h_{r,j}(n))\right] + \left[\sum_{j=1}^{O_r} T_{q_{r,j}}(g_{r,j}(n))\right] & \text{if } n \geq c_r + 1 \\ 0 & \text{if } n \leq c_r \end{cases}. \qquad (eq_r)$$

Here, the variable n is a non-negative integer modeling the size of the input passed to procedure $proc_r$ and c_r is a non-negative integer constant below which the procedure $proc_r$ halts immediately. The rest of the notation is as follows:

- $T_r(n)$ is a random variable modeling the running time of $proc_r$ on an input of size n.
- $Cost_r(n)$ is a random variable that models the preprocessing time of $proc_r$ on input size n.
- S_r is a constant that represents the number of procedure calls in $proc_r$ to $proc_r$ itself. Among these procedure calls, each $h_{r,j}(n)$ represents the random size of the instance passed to the j-th recursive procedure call (to $proc_r$ itself).
- O_r is a constant that represents the number of procedure calls in $proc_r$ to procedures other than $proc_r$. We denote the callee in the j-th such call by $q_{r,j}$. Note that $q_{r,j} \neq r$. Moreover, $g_{r,j}(n)$ is a random variable associated with input size n that models the parameter size passed

to the j-th call.

Since $h_{r,j}(n)$ and $g_{r,j}(n)$ both model input sizes to procedure calls, we assume that they observe discrete probability distributions over non-negative integers.

Intuitive description of SPRRs. Informally, given an initial input of size n^* and a prescribed main procedure $proc_{r^*}$, an SPRR runs as follows. Initially, the stack contains only the main procedure $proc_{r^*}$ with the initial input size n^*. Then, the following steps are performed:

Step 1. If the current stack is emty, then the SPRR halts, otherwise it goes to Step 2.

Step 2. The SPRR pops the top of the current stack to get the current procedure $proc_r$ and the current input size n.

Step 3. If $n \leq c_r$ then the SPRR terminates the current procedure immediately and goes back to Step 1, otherwise it goes to Step 4.

Step 4. The SPRR runs the preprocessing stage that takes a randomized amount $Cost_r(n)$ of execution time, and then goes to Step 5.

Step 5. The SPRR pushes to the top of the stack the S_r procedure calls to $proc_r$ itself with their corresponding input sizes $h_{r,j}(n)$ $(1 \leq j \leq S_r)$ and the O_r procedure calls to $proc_{q_{r,j}}$'s with their input sizes $g_{r,j}(n)$ $(1 \leq j \leq O_r)$, and then goes back to Step 1.

Note that in Step 5, we consider an arbitrarily-fixed order for pushing the procedural calls to the stack, since we are only interested in the total execution time. From the intuition above, it is straightforward to see that the random variables $T_r(n)$ $(1 \leq r \leq m)$ satisfy the recurrence equations of the SPRR.

Below we present several examples that model classical randomized algorithms as SPRRs. Due to space limitations, detailed descriptions of the algorithms are relegated to Appendix B.

Example 2.2 (QuickSelect). Consider the problem of finding the d-th smallest element in an unordered array of n distinct elements. A classical randomized algorithm for solving this problem is QuickSelect [Hoare 1961b] that provides a simple randomized variant matching the best-known $O(n)$ deterministic runtime. Since there is only one recursive procedure in QuickSelect, we model the algorithm as the following SPRR with one recurrence equation (where we have $m = 1$, $T_1 = T$, $Cost_1(n) = n$, $S_1 = 1$ and $O_1 = 0$):

$$T(n) = n + T(h(n)) \quad .$$

Here, $T(n)$ represents the number of comparisons performed by QuickSelect over an input of size n, and $h(n)$ is the random variable that captures the size of the remaining array that has to be searched recursively. It can be derived that $h(n) = \max\{i, n-1-i\}$ where i is sampled uniformly from $\{0, \ldots, n-1\}$. □

Example 2.3 (QuickSort). Consider the classical problem of sorting an array of n distinct elements. A well-known randomized algorithm for solving this problem is QuickSort [Hoare 1961a]. Just as before, there is only one recursive procedure in QuickSort. Thus, we model the algorithm as the following SPRR with one recurrence equation (where we have $m = 1$, $T_1 = T$, $Cost_1(n) = n$, $S_1 = 2$ and $O_1 = 0$):

$$T(n) = n + T(h_1(n)) + T(h_2(n)) \quad .$$

Here, $h_1(n)$ and $h_2(n)$ are random variables that capture the sizes of the two sub-arrays. It is easy to see that $h_1(n) = i$ and $h_2(n) = n - 1 - i$ where i is uniformly sampled from $\{0, \ldots, n-1\}$. □

Example 2.4 (SeidelLP). Consider a linear programming problem in d dimensions, i.e. with d variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d)$ and n constraints $\mathbf{a}_1^T \cdot \mathbf{x} \leq b_1, \ldots, \mathbf{a}_n^T \cdot \mathbf{x} \leq b_n$. Suppose the objective is to minimize $\mathbf{c}^T \cdot \mathbf{x}$. SeidelLP [Motwani and Raghavan 1995, Chapter 9.10] is a well-known randomized incremental algorithm that handles this problem. To model the algorithm by an SPRR, we set up a

system with d equations $eq_1, eq_2, \ldots, eq_d$, where eq_r corresponds to the r-dimensional recursive subinstances of the algorithm. We have $c_r = r$ and for every $n \geq r + 1$:

$$T_r(n) = Cost_r(n) + T_r(h_{r,1}(n)) + T_{r-1}(g_{r,1}(n)) \tag{1}$$

Where $h_{r,1}(n) = n - 1$. Moreover, $Cost_r$ and $g_{r,1}$ are dependent and jointly observe the following discrete distribution: with probability $\frac{r}{n}$, $Cost_r(n) = r \cdot n$ and $g_{r,1}(n) = n - 1$, and with probability $1 - \frac{r}{n}$, $Cost_r(n) = r$ and $g_{r,1}(n) = 0$.

Semantics of SPRRs. We now present formal semantics that treat each random variable $T_r(n)$ as the *accumulated cost* of a stochastic process derived from the underlying SPRR. Below, we fix an SPRR $\Delta = \{eq_1, \cdots, eq_m\}$ where each recurrence equation eq_r takes the form (eq_r) as specified on Page 5. We represent a procedure call σ as an ordered pair (r, n) of non-negative integers where r represents the r-th procedure and n is the size of the parameter passed to this procedure call. We first formally define the notion of *stacks* and *stack transition functions*.

Stacks. A *stack* β is a finite word $\sigma_1 \ldots \sigma_k$ ($k \geq 0$) such that for each $1 \leq j \leq k$, the symbol $\sigma_j = (r_j, n_j)$ is a procedure call. The word β is organized such that σ_1 corresponds to the top of the stack and σ_k is the bottom. We use ε to denote the empty stack. For a stack $\beta = (r, n)\beta'$, we define $Cost(\beta) := Cost_r(n)$. Intuitively, $Cost(\beta)$ is the preprocessing time of the call at the top of the stack.

Stack transition functions. A stack transition function describes the stochastic effect of popping the procedure call at the top of the stack and pushing all its sub-procedure calls. Formally, the *stack transition function* is a function κ that maps every procedure call $\sigma = (r, n)$ to the discrete probability distribution $\kappa(\sigma)$ over stacks such that for each stack β, $\kappa(\sigma)(\beta)$ is the probability that exactly all the procedure calls in β are pushed to the stack while respecting the order they appear in the word β[*]. The exact form of κ is completely determined by the summations $\sum_{j=1}^{S_r} T_r(h_{rj}(n))$ and $\sum_{j=1}^{O_r} T_{q_{rj}}(g_{rj}(n))$ in (eq_r) so that it can be calculated exactly from the probability distributions of the random variables $h_{rj}(n)$ and $g_{rj}(n)$. However, we do not need to provide a more detailed notation in order to present our results, which only depend on the conceptual notion of a stack transition function.

Based on stacks and the stack transition function, we now define the semantics of SPRRs through discrete-time Markov chains [Baier and Katoen 2008, Chapter 10].

Markov chain semantics of SPRRs. The semantics of an SPRR Δ is a discrete-time Markov chain $(S, \mathbf{P})$. The set S of Markov chain states is the set of all stacks. For any two stacks β, β', the transition probability $\mathbf{P}(\beta, \beta')$ is determined by the following two cases:

Case 1: $\beta = \varepsilon$. Then $\mathbf{P}(\beta, \varepsilon) := 1$ and $\mathbf{P}(\beta, \beta') := 0$ for all other stacks $\beta' \neq \varepsilon$.

Case 2: $\beta = \sigma\beta''$ with the top procedure call σ. Then $\mathbf{P}(\beta, \beta') := 0$ if there is no stack γ such that $\beta' = \gamma\beta''$, and $\mathbf{P}(\beta, \beta') := \kappa(\sigma)(\gamma)$ if $\beta' = \gamma\beta''$ for some (unique) stack γ.

In other words, if the current stack is empty, i.e. the whole SPRR is terminated, which corresponds to Case 1, then the SPRR always stays at the empty stack. Otherwise (Case 2), the SPRR pops the top procedure call of the current stack and stochastically pushes sub-procedure calls back to the top of the stack. Given the Markov chain $(S, \mathbf{P})$ of an SPRR $\Delta$and an initial stack β^* [†], the probability space $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ for Δ is the one induced by the Markov chain of Δ, see [Baier and Katoen 2008, Chapter 10]) for details. A brief description of the construction of the probability space is also provided in Appendix C. We denote the expectation under the probability space $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ as $\mathbb{E}_{\Delta,\beta^*}$, and omit the subscript Δ whenever it is clear from the context. Below we introduce several random variables under $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ that will be used later.

[*]Recall that we fix an arbitrary order that is followed when these procedure calls are pushed to the stack.

[†]Note that β^* is typically a single main procedure call σ^*.

Random stack $next_\beta$. We denote by $next_\beta$ a random stack that observes the discrete probability distribution $\mathbf{P}(\beta, -)$ and represents the next stack stochastically determined by the current stack β. Note that a random stack consists of a collection of random variables that describe all the information of the procedure calls in the stack.

Random traces. From the Markov chain semantics, a random execution of an SPRR with a main procedure call induces an infinite sequence of random stacks $\widehat{\beta}_0, \widehat{\beta}_1, \ldots$ such that the random stack $\widehat{\beta}_0$ corresponds to the (deterministic) main procedure call and each random stack $\widehat{\beta}_k$ represents the stack at the k-th step of an execution, so that for each $k \geq 0$, $\widehat{\beta}_{k+1}$ is sampled from $next_{\widehat{\beta}_k}$. We call $\widehat{\beta}_0, \widehat{\beta}_1, \ldots$ a *random trace*.

Top-stack random variables. For each $k \geq 0$, we define a pair $\widehat{\sigma}_k = (\widehat{r}_k, \widehat{n}_k)$ of random variables such that $\widehat{r}_k$ (resp. $\widehat{n}_k$) represents the procedure (resp. the parameter size passed to the procedure) on the top of the stack at the k-th step of a run. Formally, given any infinite sequence of stacks $\omega = \beta_0\beta_1 \ldots$, if $\beta_k = \sigma\beta'$ for some procedure call $\sigma = (r, n)$ and stack β' then we have $\widehat{r}_k(\omega) := r$ and $\widehat{n}_k(\omega) := n$. If the k-th stack is empty, we define $\widehat{r}_k = \widehat{n}_k = \bot$.

Stopping time τ. We define the stopping time random variable τ as the first time point when the SPRR reaches the empty stack, i.e. $\tau(\omega) := \inf\{k \mid \beta_k = \varepsilon\}$ for every run $\omega = \{\beta_j\}_{j\in\mathbb{N}}$ of stacks (where $\inf \emptyset := \infty$). It is worth noting that the stopping time τ is different from the total execution time of the underlying randomized algorithm since it does not take the execution time $Cost_r(n)$ of the preprocessing stage into account. We now define the notion of accumulated cost for SPRRs.

Accumulated cost. We first introduce a random variable C_k for each $k \geq 0$ that represents the accumulated cost in a run of Δ until the k-th step. We treat the time spent on the preprocessing stage as cost so that the total accumulated cost until a run of Δ reaches empty stack corresponds directly to the total execution time of the underlying randomized algorithm. Thus, for every run ω, we define $C_k(\omega) := \sum_{j=0}^{k-1} Cost_{\widehat{r}_j(\omega)}(\widehat{n}_j(\omega))$, where $Cost_r(\bot) := 0$. The *accumulated cost* is simply defined as C_τ. From the definition, it is straightforward to see that the accumulated cost C represents the randomized execution time of the underlying randomized algorithm.

We are now formally define the concentration bound analysis problem over SPRRs, which is the central problem in our paper.

Concentration bound analysis of SPRRs. Consider an SPRR Δ and a main procedure call $\sigma^* = (r^*, n^*)$. Given a symbolic expression κ over n^*, the goal is to infer an upper-bound on the probability that the accumulated cost C_τ is no smaller than κ. Formally, our task is to infer an upper-bound on

$$\Pr_{\Delta,\sigma^*}(C_\tau \geq \kappa) \tag{2}$$

where κ is a polynomial over the symbolic input size n^* and $\ln n^*$ with symbolic coefficients. Intuitively, this corresponds to bounding the probability that the runtime of the randomized algorithm exceeds the bound κ. Moreover, we would like the upper-bound to decrease dramatically and rapidly tend to zero as κ increases.

3 THE GENERAL THEORETICAL APPROACH

In this section, we develop sound approaches for solving the concentration bound analysis problem over SPRRs. We first recall some basic results from probability theory, then demonstrate the theoretical foundation of our approaches, and finally present our main theorems under practical considerations. Below we fix an SPRR with m recurrence equations where the r-th recurrence equation eq_r is in the form (eq_r).

3.1 Basic Results from Probability Theory

We first recall the well-known Markov's inequality. Intuitively, this inequality provides a way to obtain concentration bounds for a non-negative random variable.

Theorem 3.1 (Markov's inequality [Williams 1991, Chapter 6]). *For any non-negative random variable X and real number $a > 0$, we have that* $\Pr(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$.

Using the following lemma, we can apply Markov's inequality to a random variable that is not necessarily non-negative:

Lemma 3.2. *Consider any random variable X and monotonically non-decreasing function $f : \mathbb{R} \to [0, \infty)$. For any real number d such that $f(d) > 0$, we have* $\Pr(X \geq d) \leq \mathbb{E}[f(X)]/f(d)$.

Proof. Since f is monotonically non-decreasing, we have $\Pr(X \geq d) \leq \Pr(f(X) \geq f(d))$ as $X \geq d$ implies $f(X) \geq f(d)$. By Markov's inequality (Theorem 3.1), we obtain $\Pr(f(X) \geq f(d)) \leq \mathbb{E}[f(X)]/f(d)$. The result follows. □

Our approaches also depend on the classical Optional Stopping Theorem, which is well-known in martingale theory. For simplicity, we only present a minimal part of the theorem, which is required for our method:

Theorem 3.3 (Optional stopping theorem [Williams 1991, Theorem 10.10]). *For any discrete-time supermartingale $\Gamma = \{X_n\}_{n \in \mathbb{N}}$ adapted to a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ where every random variable X_n is non-negative, and any stopping time ρ w.r.t the filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ that is almost surely finite (i.e.* $\Pr(\rho < \infty) = 1$*), we have* $\mathbb{E}[X_\rho] \leq \mathbb{E}[X_0]$.

3.2 Taylor Supermartingale Function Sequences and Our Main Theorem

The core concept in our theoretical foundations is the novel notion of Taylor Supermartingale Function Sequences (TSFS). These sequences lead to concentration bounds of SPRRs through Markov's inequality and Optional Stopping Theorem. Note that, by Lemma 3.2, to solve the concentration bound problem for the accumulated cost random variable C, it suffices to choose a monotonically non-decreasing function f and then bound the expected value of $f(C)$. In this work, we consider a specific class of functions, called T-functions, as our choices for the monotonically non-decreasing function f in Lemma 3.2.

The Class $\mathbb{T}$ of T-functions. We consider a special class of functions, called *Taylor functions* (*T-functions*), that have Taylor expansions and non-negative derivatives everywhere on $[0, \infty)$. To be more precise, we define the *T-function* class $\mathbb{T}$ as the set of all functions $f : [0, \infty) \to \mathbb{R}$ that (i) have non-negative derivatives $f^{(k)}$ for every order $k \geq 0$, and (ii) can be expanded into Taylor series at every non-negative real number. More precisely, a function $f : [0, \infty) \to \mathbb{R}$ falls in the class $\mathbb{T}$ (i.e. is a T-function) iff the following conditions hold:

- (C1) for every order $k \geq 0$, the k-th order derivative function $f^{(k)}$ exists and is non-negative (i.e. $f^{(k)}(x) \geq 0$ for every real number $x \geq 0$);
- (C2) for any reals $x, y \geq 0$, we have $f(x) = \sum_{k=0}^{\infty} \frac{(x-y)^k}{k!} \cdot f^{(k)}(y)$, i.e. the function f can be expanded into its Taylor series at any non-negative real number y.

Note that the function class $\mathbb{T}$ includes many exponential functions (e.g. $f(x) = \exp(t \cdot x)$ where $t > 0$) and polynomial functions (e.g. $f(x) = d \cdot x^k$ where $d, k > 0$ and k is an integer). Our approach combines the ideas of T-functions, Markov's inequality (Theorem 3.2) and Optional Stopping Theorem (Theorem 3.3) to bound the concentration probability $\Pr_{\Delta,\sigma^*}(C \geq \kappa)$ in (2) by first choosing a suitable T-function f and then bounding all its k-th order derivatives $f^{(k)}$.

Theorem 3.4 (Main Theorem). *Let Δ be a given SPRR. Consider a T-function $f \in \mathbb{T}$ and an infinite sequence Ψ of functions $\Psi = \Psi_0, \Psi_1, \Psi_2, \ldots$ such that each Ψ_k maps a stack β of Δ to a non-negative*

real value $\Psi_k(\beta)$*. Suppose that the sequence* Ψ *satisfies the following conditions:*

(A1) *for every* $k \geq 0$*, we have* $\Psi_k(\varepsilon) \geq f^{(k)}(0)$*;*

(A2) *for every stack* β *and* $k \geq 0$*, we have* $\Psi_k(\beta) \geq \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{k+j}(next_\beta)}{j!} \cdot Cost^j(\beta)\right]$.

Then, for every stack β *and* $k \geq 0$*, it holds that* $\Psi_k(\beta) \geq \mathbb{E}_\beta\left[f^{(k)}(C)\right]$.

Intuitively, (A1) states that Ψ_k is an upper-bound for the k-th order derivative of a T-function f at the empty stack, for which the empty stack corresponds to the zero point at $f^{(k)}$; (A2) specifies that the expectation of expanding the function Ψ_k into its Taylor series over the next probability distribution $next_\beta$ and the preprocessing time $Cost(\beta)$ of the current stack β is no more than its current value Ψ_k. In the sequel, we refer to conditions (A1) and (A2) as *Taylor Supermartingale* (TS) conditions, and to an infinite sequence Ψ of functions $\Psi = \Psi_0, \Psi_1, \Psi_2, \ldots$ that fulfill the TS conditions as a *Taylor Supermartingale Function Sequence* (TSFS). We now provide a sketch of our proof for this theorem. For a detailed proof, see Appendix D.1.

Proof Sketch for Theorem 3.4. Fix any stack β and consider the random trace $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2, \ldots$ for the SPRR Δ with the initial stack $\widehat{\beta}_0 = \beta$. For each $k \geq 0$, we define $\mathcal{G}_k$ as the smallest σ-algebra that makes the random variables in $\widehat{\beta}_0, \ldots, \widehat{\beta}_k$ and $C_0, \ldots, C_{k-1}$ measurable. It is straightforward to observe that $\{\mathcal{G}_k\}_{k\geq 0}$ is a filtration. For each $k, i \geq 0$, we define the random variable $Y_{k,i} := \sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_k)}{j!} \cdot C_k^j$ (where we assume $0^0 := 1$). We prove that for every $k, i \geq 0$, the following properties hold:

(a) for every $k, i \geq 0$, $Y_{k,i}$ is non-negative;

(b) $Y_{\tau,i} \geq f^{(i)}(C_\tau)$;

(c) For every $i \geq 0$, $Y_{k,i}$ forms a supermartingale with respect to the filtration $\{\mathcal{G}_k\}_{k\geq 0}$.

First, note that Property (a) follows directly from the definition of $Y_{k,i}$. Second, as $\widehat{\beta}_\tau = \varepsilon$, Property (b) can be deduced from (A1) as follows: $Y_{\tau,i} = \sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_\tau)}{j!} \cdot C_\tau^j \geq \sum_{j=0}^{\infty} \frac{f^{(i+j)}(0)}{j!} \cdot C_\tau^j = f^{(i)}(C_\tau)$. We now prove Property (c).

$$
\begin{aligned}
\mathbb{E}[Y_{k+1,i} \mid \mathcal{G}_k] &= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{j!} \cdot C_{k+1}^j \,\middle|\, \mathcal{G}_k\right] \text{ (By the definition of } Y_{k+1,i}) \\
&= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{j!} \cdot (C_k + Cost(\widehat{\beta}_k))^j \,\middle|\, \mathcal{G}_k\right] \text{ (By the definition of } C_{k+1}) \\
&= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{j!} \cdot \sum_{l=0}^{j} \binom{j}{l} \cdot C_k^l \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \text{ (By the Binomial Theorem)} \\
&= \mathbb{E}\left[\sum_{j=0}^{\infty} \sum_{l=0}^{j} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{l!(j-l)!} \cdot C_k^l \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \left(\text{By } \binom{j}{l} = \tfrac{j!}{l!\cdot(j-l)!}\right) \\
&= \mathbb{E}\left[\sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \text{ (By rearrangement of summation)} \\
&= \sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \text{ (By } (*) \text{ below)} \\
&\leq \sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \Psi_{i+l}(\widehat{\beta}_k) = Y_{k,i} \text{ (By the condition (A2) and } (\star) \text{ in the conditions (E1)–(E3))}
\end{aligned}
$$

The $(*)$ includes properties of conditional expectations such as linearity, positivity, cMON and 'taking out what is known' as in [Williams 1991, Chapter 9.7, Page 88] (which we extend to non-negative random variables). The omitted details at $(\star)$ follow the basic definition of conditional expectation, i.e. (E1)–(E3), to relate $\Psi_{i+j}(\widehat{\beta}_{k+1})$ and $\Psi_{i+j}(next_{\widehat{\beta}_k})$. Then, by applying the Optional Stopping Theorem (Theorem 3.3), we have that $\mathbb{E}[Y_{\tau,i}] \leq \mathbb{E}[Y_{0,i}]$ for every $i \geq 0$. The result follows from the fact that $Y_{\tau,i} \geq f^{(i)}(C_\tau)$ (see Property (b)) and $Y_{0,i} = \Psi_i(\beta)$. □

Application of Theorem 3.4. Theorem 3.4 provides a way to bound the expected value $\mathbb{E}_\beta[f(C_\tau)]$ by $\Psi_0(\beta)$. Combined with Lemma 3.2, we obtain a general procedure for deriving concentration bounds for SPRRs as follows:

- We find an appropriate T-function $f \in \mathbb{F}$.
- We find a TSFS $\Psi = \Psi_0, \Psi_1, \Psi_2, \ldots$ to obtain the bound $\Psi_k(\beta)$ for $\mathbb{E}_\beta\left[f^{(k)}(C_\tau)\right]$.
- We apply Lemma 3.2 to derive the concentration bound $\Pr_{\sigma^*}(C_\tau \geq \kappa) \leq \frac{\mathbb{E}_{\sigma^*}[f(C_\tau)]}{f(\kappa)} \leq \frac{\Psi_0(\sigma^*)}{f(\kappa)}$.

However, the general procedure above is infeasible to realize algorithmically, since one does not fix the form of a T-function f and the TS conditions (A1) and (A2) are defined for all stacks and an infinite sequence of functions. In other words, one would need to consider an infinite number of stacks and functions. To overcome this difficulty, in the following we consider practical versions of Theorem 3.4 for exponential and polynomial T-functions (Theorems 3.5 and 3.6). Based on these, we then develop our algorithmic approaches in Section 5.

3.3 Composition Theorems for Exponential and Polynomial T-Functions

Instead of applying Theorem 3.4 directly, we consider simplified versions (called *composition theorems*) of Theorem 3.4 over SPRRs. In many scenarios, concentration bounds are exponentially- and polynomially-decreasing. Hence, we focus on exponential and polynomial T-functions since they correspond to these two common types of concentration bounds. In detail, we devise composition theorems for exponential and polynomial T-functions respectively. Informally, the composition theorems reduce the synthesis of an infinite length TSFS into synthesizing a finite number of functions. In Section 5, we will illustrate our algorithmic approach based on these two theorems. Below, we will introduce our two composition theorems for exponential and polynomial T-functions respectively.

The Composition Theorem for exponential T-functions is stated in Theorem 3.5. Intuitively, the theorem considers $2 \cdot m$ functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$ where functions f_r and t_r correspond to the r-th recurrence equation eq_r in the input SPRR, and imposes constraints on these functions. If all the constraints are satisfied, then one could compose from these $2 \cdot m$ functions a TSFS that fulfills the TS conditions (A1) and (A2) w.r.t some exponential T-function $f(x) = \exp(t \cdot x)$ $(t > 0)$. Thus, the theorem allows our algorithm to focus on the synthesis of only a finite number of functions, i.e. $2 \cdot m$ functions, over the recurrence equations of the input SPRR.

Theorem 3.5 (Composition Theorem For Exponentially Decreasing Bounds). *Consider $2 \cdot m$ functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$ both from $\mathbb{N}$ into $[0, \infty)$. Suppose that the following condition holds for every $1 \leq r \leq m$ and $1 \leq n \leq n^*$:*

$$\exp(t_r(n) \cdot f_r(n)) \geq \mathbb{E}\left[\exp\left(t_r(n) \cdot \left(Cost(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n))\right)\right)\right] \quad (\ddagger)$$

Then, the following sequence $\{\Psi_i\}_{i \geq 0}$ is a TSFS w.r.t the exponential T-function $f(x) := \exp(t_\star \cdot x)$:

$$\Psi_i(\beta) := t_\star^i \cdot \exp\left(t_\star \cdot \sum_{j=1}^{k} f_{r_i}(n_i)\right) \text{ for any } i \geq 0 \text{ and stack } \beta = (r_1, n_1) \ldots (r_k, n_k)$$

where $t_\star := \min_{1 \leq r \leq m} \min_{1 \leq n \leq n^*} t_r(n)$ is the minimum value over all function values.

Proof Sketch. We will verify conditions (A1) and (A2) in our main theorem (Theorem 3.4). (A1) directly follows from the non-negativity of $f_1, f_2, \ldots, f_m$'s. To verify (A2), firstly, due to the convexity of $t_r(n)$, we could apply condition (‡) to the top element of β, and then by the definition of $next_\beta$, we conclude that (A2) holds. Please refer to Appendix D for details. □

Thus, once we synthesize appropriate functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$ that satisfy the conditions specified in the theorem, we could construct a TSFS that leads to an exponentially-decreasing concentration bound by Theorem 3.4.

We now illustrate our Composition Theorem for polynomial T-functions. Again, the theorem reduces the synthesis of TS functions derived from some polynomial T-function into the synthesis of a finite number of functions. In detail, the theorem considers $m \cdot (k+1)$ functions $f_{r,i}$ ($1 \leq r \leq m$, $0 \leq i \leq k$) for which each $f_{r,i}(n)$ informally represent an over-approximation of $\mathbb{E}[T_r(n)^i]$ (i.e. the expectation of the higher moment $T_r(n)^i$), and imposes two groups of constraints for these functions. One group of constraints is set up for every recurrence equation (eq_r) in the given SPRR and every degree $1 \leq i \leq k$, and the other group of constraints are boundary conditions for $f_{r,0}$ for each recurrence equation (eq_r) in the SPRR. If all the constraints are satisfied, then our theorem can construct a TSFS with respect to a polynomial T-function $f(x) = x^k$.

Theorem 3.6 (Composition Theorem for Polynomially Decreasing Bounds). *Let Δ be an SPRR with m recurrence equations and k be a parameter that represents the degree of desired monomials. Consider $m \cdot (k+1)$ functions $f_{r,i}$ ($1 \leq r \leq m$, $0 \leq i \leq k$), each of which maps natural numbers to non-negative reals. Suppose that for every $1 \leq r \leq m$ and $1 \leq i \leq k$, the following constraints are satisfied:*

$$\forall n.\{1 \leq n \leq n^*\} \Rightarrow \left\{ \frac{f_{r,i}(n)}{i!} \geq \mathbb{E}\left[\sum_{\mathcal{I}} \frac{Cost_r(n)^j}{j!} \cdot \prod_{u=1}^{S_r} \frac{f_{r,p_u}(h_{r,u}(n))}{p_u!} \cdot \prod_{u=1}^{O_r} \frac{f_{q_{r,u},e_u}(g_{r,u}(n))}{e_u!} \right] \right\} \tag{3}$$

$$\forall n.\{1 \leq n \leq n^*\} \Rightarrow \{f_{r,0}(n) \geq 1\} \tag{4}$$

where $\mathcal{I} := \left[0 \leq j, p_1, \ldots, p_{S_r}, e_1, \ldots, e_{O_r} \leq i, \sum_{u=1}^{S_r} p_u + \sum_{u=1}^{O_r} e_u + j = i \right]$. Then, the following sequence of functions $\Psi_0, \Psi_1, \ldots$ satisfies condition (A1)–(A2) of Theorem 3.4, with respect to the T-function $f(x) = x^k$, given β represented by $\beta = (r_1, n_1) \ldots (r_t, n_t)$:

- *For $0 \leq i \leq k$, we define: $\Psi_i(\beta) := k! \cdot \sum_{0 \leq p_1, \cdots, p_t \leq k-i, \sum_{u=1}^{t} p_u = k-i} \prod_{u=1}^{t} \frac{f_{r_u,p_u}(n_u)}{p_u!}$.*
- *For $i \geq k+1$, we define: $\Psi_i(\beta) := 0$.*

Proof Sketch. We verify conditions (A1) and (A2) in our main theorem (Theorem 3.4). (A1) is straightforward by non-negativity of $f_{i,j}$'s. To verify (A2), we first apply condition (3) to the top element of β, and then through the definition of $next_\beta$, we conclude that (A2) holds. Please refer to Appendix D for details. □

4 AN ILLUSTRATION OF OUR ALGORITHMIC METHOD ON QUICKSELECT

Before presenting our synthesis algorithms in detail, we first illustrate our approach over a representative example, namely the QuickSelect SPRR of Example 2.2. A formal description of our algorithms will be provided in Section 5. Consider the QuickSelect SPRR in Example 2.2. We aim to derive an exponentially decreasing concentration bound for this example.

Upper-bound on expected runtime. Suppose that we are provided with a function $f(n) := 6 \cdot n$ that serves as an over-approximation of $\mathbb{E}[T(n)]$. This function can be automatically obtained using previous approaches for expected runtime analysis, such as [Chatterjee et al. 2017a]. In this work, we assume that such upper-bounds are given as part of the input.

Synthesis goal. Since the only recurrence equation in this example is

$$T(n) = n + T(h(n)),$$

in order to apply our composition theorem for exponential concentration bounds (Theorem 3.5), it suffices to synthesize a function $t(n)$ that satisfies the following condition:

$$\forall n \quad (1 \leq n \leq n^*) \Rightarrow \mathbb{E}[\exp(t(n) \cdot (n + f(h(n))))] \leq \exp(t(n) \cdot f(n)).$$

Intuition behind the constraint. Intuitively, the LHS represents the expectation of $\exp(t(n) \cdot f(n))$ after the one-step expansion of the recurrence equation. Therefore, the inequality above encodes a supermartingale condition for the exponential term $\exp(t(n) \cdot f(n))$.

Rephrasing the inequality. By dividing both sides of the condition by $\exp(t(n) \cdot f(n))$, we obtain this equivalent formulation:

$$\forall n \quad (1 \leq n \leq n^*) \Rightarrow \mathbb{E}[\exp(t(n) \cdot X(n))] \leq 1$$

where $X(n) \coloneqq n + 6 \cdot h(n) - 6 \cdot n$ is a random variable.

Setting up a template. Following the intuition above, our approach chooses a template for $t(n)$ and sets $t(n) \coloneqq \lambda/n$ where λ is an unknown coefficient that should be synthesized. Now the condition becomes:

$$\forall n.(1 \leq n \leq n^*) \Rightarrow \mathbb{E}[\exp((\lambda/n) \cdot (n + 6 \cdot h(n) - 6 \cdot n))] \leq 1.$$

Intuitively, we use $t(n)$ as a scaling factor that allows us to bound the range of $t(n) \cdot X(n)$ by constants.

Binary searching. Next, our approach performs a binary search to find the optimal value of λ. The binary search is performed with the help of a verification procedure that, given a guess for λ, verifies whether the condition above holds.

Verification procedure. Suppose that our current guess is $\lambda = 1$. To verify whether the condition holds for $\lambda = 1$, our approach expands the exponential term in the expectation above through Taylor's expansion and over-approximates the remainder. To over-approximate the remainder, our algorithm firstly needs to bound $t(n) \cdot X(n)$ by constants.

Synthesizing the Bound D. Our approach firstly bounds $X(n)$ by $B(n) \coloneqq 3.67 \cdot n$, i.e., we have that $-B(n) \leq X(n) \leq B(n)$. By plugging in $t(n) = 1/n$, we have that the bound D for $t(n) \cdot X(n)$, i.e. $-3.67 \leq t(n) \cdot X(n) \leq 3.67$ This leads to the following strengthened condition that soundly approximates the original one:

$$1 + \sum_{i=1}^{L} \frac{n^{-i}}{i!} \cdot \mathbb{E}\left[X(n)^i\right] + \frac{n^{-(L+1)} \cdot \exp(3.67)}{(L+1)!} \cdot \mathbb{E}\left[X(n)^{L+1}\right] \leq 1. \tag{5}$$

In this inequality, L is the depth of the Taylor expansion and is always odd.

Over-approximating expectation terms. To solve this, we still need to over-approximate the higher moment $\mathbb{E}\left[X(n)^i\right]$. We introduce a moment approximation algorithm that can derive tight upper and lower approximations up to any prescribed precision. Suppose we are to over-approximate $\mathbb{E}[X(n)^2]$. Our approach first expands $\mathbb{E}[X(n)^2]$ by applying Newton's binomial theorem to the definition of $X(n)$:

$$\mathbb{E}[X(n)^2] = 36 \cdot \mathbb{E}[h(n)^2] - 60 \cdot n \cdot \mathbb{E}[h(n)] + 37 \cdot n^2.$$

Now, it suffices to over-approximate $\mathbb{E}[h(n)^2]$ and under-approximate $\mathbb{E}[h(n)]$. To over-approximate $\mathbb{E}[h(n)^2]$, our approach first expands this expectation as a summation that iterates over all of the possible elements in the support of $h(n)$'s distribution. Recall that $h(n) = \max\{i, n-1-i\}$ where i is uniformly distributed over $\{0, \ldots, n-1\}$. Then, it soundly removes floors and ceilings from expressions such as $\lfloor \frac{n}{2} \rfloor$. As expected, in the removal of floors and ceilings, our approach distinguishes between even and odd n. Below, we present the even case:

$$\mathbb{E}[h(n)^2] \leq \frac{1}{n} \cdot \left(\sum_{i=n/2}^{n-1} i^2 + \sum_{i=n/2}^{n-1} i^2 \right).$$

Next, by applying our moment approximation algorithm that derives upper and lower bounds for any summation of the form $\sum_{i=0}^{n-1} i^a \ln^b i$ up to any prescribed precision, our approach obtains that (i) $\mathbb{E}[h(n)^2]$ is exactly $\frac{7}{12} \cdot n^2 - \frac{3}{4} \cdot n + \frac{1}{6}$, and (ii) $\mathbb{E}[h(n)]$ is exactly $\frac{3}{8} \cdot n - \frac{1}{4}$. As a consequence, it over-approximate $\mathbb{E}[X(n)^2]$ as $13 \cdot n^2 + 3 \cdot n + 6$.

Checking the strengthened inequality. Finally, after approximating all the higher moments that appear in Equation (5), our algorithm decides that the inequality in (5) holds for the guess $\lambda = 1$

and returns true[‡].

Final results. Our approach then continues with the binary search and obtains an optimal value of $\lambda^* \approx 1.364$ for λ. Thus, it successfully synthesizes $t(n) := \frac{1.364}{n}$. By plugging $t(n)$ back into our core theorem for exponential bounds (Theorem 3.5), our approach synthesizes the following concentration bound:

$$\Pr[T(n) \geq \alpha \cdot n] \leq \exp(-1.364 \cdot \alpha + 8.184).$$

Comparison with Karp's cookbook. In comparison, [Karp 1994] provides $\Pr(T(n) \geq \alpha \cdot n) \leq (3/4)^{\alpha-4}$. Thus, our approach produces a much tighter bound than Karp's method. For $\alpha = 20$, our method synthesizes the bound $5.089 \cdot 10^{-9}$, while Karp's method outputs 0.01. In other words, our method beats [Karp 1994] by a ratio of $1.984 \cdot 10^6$.

Extension to polynomially decreasing bounds. Our approach is not limited to exponential concentration bounds. If we consider the constraint in Theorem 3.6 instead of Theorem 3.5, then every step of the algorithm above can be adapted with minimal changes in order to synthesize a polynomially decreasing concentration bound. See Section 5.4 for details. As an example, our approach is able to synthesize the following bound for SeidelLP with dimension $d = 2$:

$$\Pr[C_\tau \geq \alpha \cdot 8 \cdot n^*] \leq 9.33 \cdot \alpha^{-7}$$

In the constrast, since the problem involves stochastic cost fucntion and a system of probabilistic recurrence relations, it is beyond the theoretical framework of Karp's method.

5 ALGORITHMIC APPROACHES

In this section, we provide algorithmic approaches for automatic synthesis of a TSFS for exponential and polynomial T-functions. Our algorithmic approaches are based on Composition Theorems (Theorems 3.5 and 3.6) for exponential and polynomial T-functions. Additionally, we make extra assumptions on our input SPRRs. While these assumptions mean that our algorithmic approaches are not as general as the theoretical foundation presented in the previous section, they nevertheless enable us to obtain efficient algorithms that can handle real-world SPRRs and obtain tighter concentration bounds. We first recall the syntax of SPRRs.

SPRRs. Recall that an SPRR Δ consists of m recurrence equations for which the r-th equation ($1 \leq r \leq m$) in the SPRR is of the following format:

$$T_r(n) = \begin{cases} Cost_r(n) + \left[\sum_{j=1}^{S_r} T_r(h_{r,j}(n))\right] + \left[\sum_{j=1}^{O_r} T_{q_{r,j}}(g_{r,j}(n))\right] & \text{if } n \geq c_r + 1 \\ 0 & \text{if } n \leq c_r \end{cases} \qquad (eq_r)$$

where T_r represents the r-th procedure, c_r is the non-negative integer constant for the threshold of termination of procedure T_r, $Cost_r(n)$ is the random amount of preprocessing time, $T_r(h_{r,j}(n))$'s ($1 \leq j \leq S_r$) are the S_r procedure calls to T_r itself with random sizes $h_{r,j}(n)$, and $T_{q_{r,j}}(g_{r,j}(n))$'s ($1 \leq j \leq O_r$) are the procedure calls to procedures $T_{q_{r,j}}$ other than T_r with random sizes $g_{r,j}(n)$. We assume that the m-th procedure is the main procedure and model the initial input size by n^*.

Since many randomized algorithms have expected runtimes in the form of polynomials in the input size n and its logarithm $\ln n$, we consider concentration bounds that also involve these expressions. In this work, we refer to these polynomial-like expressions as *pseudo-polynomials*.

Pseudo-polynomials. A (univariate) *pseudo-polynomial* (in the input-size variable n) is simply a polynomial over n and $\ln n$. Formally, it is a function of the form $p(n) = \sum_{i,j \in \mathbb{Z}} d_{ij} \cdot n^i \cdot \ln^j n$ where $d_{i,j}$'s are real numbers. We also call an expression of the form $d_{ij} \cdot n^i \cdot \ln^j n$ a *pseudo-monomial*. Our goal is then to automatically derive exponentially and polynomially decreasing concentration

[‡]Checking the inequality is not straightforward. We also provide a dedicated procedure for this step. The details are removed in order to keep this illustration high-level.

bounds in the form of $\Pr_{\Delta,\sigma^*}(C_\tau \geq \kappa)$ where the expression κ is assumed to be a pseudo-monomial.

Assumptions on the SPRR structure. While our general theory (Section 3) is applicable to all SPRRs, for the sake of obtaining more efficient algorithms, we assume the input SPRRs in this section satisfy the following conditions:

- *Non-mutuality*: We prohibit mutual recursion and assume that the call graph among our procedures is acyclic, except for potentially the recursive calls made by a procedure to itself. Formally, for every $1 \leq r \leq m$ and $1 \leq j \leq O_r$, we assume that $q_{r,j}$ is in $\{1, \ldots, r-1\}$.
- *Ordinariness*: We assume that the recursive calls are ordinary. In other words, in each procedure that calls itself recursively, we assume that either there is only one such call, or there are two such calls which correspond to a divide-and-conquer approach and the sum of sizes in the two calls are exactly $n-1$. Formally, we either have $S_r = 1$, or $S_r = 2$ and $h_{r,1}(n) + h_{r,2}(n) = n-1$. Note that most real-world SPRRs are ordinary. For example, QuickSelect has only one recursive call to itself, whereas QuickSort fits into our divide-and-conquer category, as it makes two recursive calls to itself and the sum of sizes passed to the two calls is $n-1$.

Assumptions on probability distributions. Any algorithmic approach to concentration bounds for SPRRs has to handle probability distributions in the recurrences. Aspects of a recurrence that can be probabilistic include the preprocessing cost function $Cost_r(n)$ and the sizes passed to recursive function calls, namely $h_{r,j}(n)$ and $g_{r,j}(n)$. In the sequel, we restrict our attention to only those distributions that are used in our real-world SPRRs.

Formally, we assume that $Cost_r(n)$ is sampled from a discrete distribution with a finite support whose every element and the probability assigned to each element in this distribution is a pseudo-polynomial with rational coefficients.

As mentioned before, if $S_r = 2$, then $h_{r,2}(n) = n - 1 - h_{r,1}(n)$. Hence, we only have to specify probability distributions for $g_{r,j}(n)$'s and $h_{r,1}(n)$. In the sequel, we assume that the random variable $h_{r,1}(n)$ is indepedent with others. For $h_{r,1}(n)$, our algorithm supports three types of distribution:

- A discrete distribution with a finite support whose every element is eithor an affine expression with natural coefficients, or in the form $\lfloor (n+b)/2 \rfloor$ for some integer b. For example, the support may contain $n-1$, 2, $\lfloor \frac{n}{2} \rfloor$ or $\lfloor \frac{n-1}{2} \rfloor$. The probability assigned to each element is a pseudo-polynomial.
- A uniform distribution over $\{0, 1, \ldots, n-1\}$,
- $\max\{i, n-i-1\}$ where i is distributed uniformly over $\{0, 1, \ldots, n-1\}$.

For $g_{r,j}(n)$'s, we suppose that they observe as the same type of discrete distributions as the first case for $h_{r,1}(n)$.

We first present some prerequisites of our algorithms in Sections 5.1 and 5.2 and then proceed with the main algorithms in Sections 5.3 and 5.4.

5.1 Approximations by Pseudo-Polynomials

Approximating higher-order moments. In several of our algorithmic steps in the sequel, we will need to approximate the higher-order moments of $f(i)$, where f is a pseudo-polynomial and either i observes a uniform distribution or $i = \max\{j, n-1-j\}$ and j observes a uniform distribution. To achieve this, we need to tightly approximate the summation

$$\sum_{i=0}^{n-1} i^a \cdot \ln^b i \tag{6}$$

by a pseudo-polynomial. We now provide a short overview of how this approximation can be performed. For more details, see Appendix E. Our approximation method is MomentApprox(L_T, a, b, n) to approximate (6). It approximates the summation through integrals and Taylor expansions. The

parameters a and b represent the degrees for i and $\ln i$, respectively, and L_T is a parameter chosen by the user that represents the degree of the approximation. To avoid corner cases, our algorithm assumes $n \geq \exp(b/L_T) + 1$. The algorithm's goal is to synthesize pseudo-polynomial upper and lower bounds for (6) with constant additive error. To approximate (6), our algorithm uses its corresponding integral $\int_d^n x^a \cdot \ln^b x \, \mathrm{d}x$, since the latter could be automatically computed as a pseudo-polynomial through integration by parts. Then, our algorithm would measure the difference between the summation and integral through Taylor's expansion with degree $a + L_T$ at each term. The algorithm controls the additive error for minor terms using the Riemann Zeta function.

Approximating logarithms of affine expressions. We now present a proposition that helps us approximate $\ln^v(a \cdot x + b)$ by a pseudo-polynomial:

Proposition 5.1. *For every $a, b, x > 0$, the following inequalities hold:*

$$\ln a + \ln x \leq \ln(a \cdot x + b) \leq \ln a + \ln x + \tfrac{b}{a \cdot x}$$

Moreover, for every $a > 0, b < 0$, and $x \geq \frac{-2 \cdot b}{a}$, we have:

$$\ln a + \ln x + \tfrac{2 \cdot b}{a \cdot x} \leq \ln(a \cdot x + b) \leq \ln a + \ln x + \tfrac{b}{a \cdot x}.$$

A proof is provided in Appendix G. Using this proposition, to approximate $\ln^v(a \cdot x + b)$, we can first approximate $\ln(a \cdot x + b)$ by the proposition above and then calculate its v-th power (and expand it by applying the binomial theorem).

5.2 Checking Non-positivity of Pseudo-polynomials

In both of our algorithms (Sections 5.4 and 5.3), we need to check whether a given pseudo-polynomial P is non-positive for every $n \geq n_0$. We now provide a sketch of our algorithm for handling this problem. A detailed description is provided in Appendix F.

Our algorithm first obtains an equivalent form $P_1(n)$ of $P(n)$ by dividing everything by the term with the largest order. Then, it naively tries to iterate over every $n \geq n_0$ and evaluate $P_1(n)$. This naive approach will of course take infinite time. Thus, we propose a condition for ending the iteration. The condition considers an over-approximation $P_2(n)$ of $P_1(n)$ by removing all *non-constant* terms with negative coefficients. It also tries to find a constant n_e such that $P_2(n)$ decreases monotonically for every $n \geq n_e$. Intuitively, this means that if $P_2(n_1) \leq 0$ for some $n_1 \geq n_e$, then $P_2(n) \leq 0$ for every $n \geq n_1$. Combining this with the fact that P_2 is an over-approximation of P_1, we can conclude that $P_1(n)$ and $P(n)$ are also non-positive for every $n \geq n_1$. Hence, we can safely terminate the iteration after n_1. Note that our algorithm is complete in the sense that if $P(n)$ is always non-positive, the algorithm will eventually terminate and report this property.

5.3 Our Algorithm for Synthesizing Exponentially Decreasing Bounds

In this section, we provide a detailed explanation of the algorithm used in Section 4 to obtain concentration bounds for QuickSelect. While the main ideas are the same as in Section 4, we now present them in a more general format, so that they are applicable to a wider family of SPRRs. To facilitate switching between the concrete example and the abstract explanation of the algorithm, our paragraph names in this section correspond to those of Section 4.

Sketch of the algorithm. Our algorithmic approach to exponentially decreasing bounds is based on Theorem 3.5. Recall that to apply this theorem, it suffices to synthesize $2 \cdot m$ functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$ that satisfy the condition in Equation (‡) below. Then, the bound can easily be computed by relying on the T-function $f(x) = \exp(t_\star \cdot x)$ where $t_\star := \min_{1 \leq r \leq m} \min_{1 \leq n \leq n^*} t_r(n)$. We assume that the t_r's and f_r's are in a pseudo-monomial form and use a template-based method for our synthesis. As we will see, the core of the problem is to find a value λ_r, which is the coefficient in template of t_r, for each equation $1 \leq r \leq m$ in our SPRR. Since the SPRR is assumed to be

non-mutual, we can find these values in order from λ_1 to λ_m. For each r, our goal is to find the largest possible λ_r that satisfies (‡). Therefore, if we have a verification algorithm that, given $t_1, \ldots, t_r$ and $f_1, \ldots, f_r$ decides whether (‡) is satisfied, then we can find λ_r by a simple binary search. Below, we present the details of each step separately.

Intuition behind the condition (‡). Let us consider Theorem 3.5 again. We are looking for $2 \cdot m$ functions $f_1, \ldots, f_m, t_1, \ldots, t_m : \mathbb{N} \to [0, \infty)$ such that:

$$\exp(t_r(n) \cdot f_r(n)) \geq \mathbb{E}\left[\exp\left(t_r(n) \cdot \left(Cost_r(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n))\right)\right)\right] \qquad (‡)$$

Intuitively, f_r is an over-approximation of the expected running time of the r-th procedure and t_r is a scaling factor. Note that the f_r's can be obtained by previous automated methods for bounding expected runtimes, such as the one in [Chatterjee et al. 2017a]. Hence, our focus is on synthesizing the t_r's.

Rephrasing the condition. If we define

$$X_r(n) := -f_r(n) + Cost_r(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n)),$$

then (‡) can simply be written as

$$\mathbb{E}\left[\exp(t_r(n) \cdot X_r(n))\right] \leq 1.$$

As we will see in the sequel, we need to reason about the range of X_r in our verification algorithm.

Setting up templates. In the first step, the algorithm sets up a monomial template for every t_r and f_r. In other words, it assumes that each of the functions is of the following form:

- $t_r(n) = \lambda_r \cdot n^{-a_r} \cdot \ln^{-b_r} n$
- $f_r(n) = w_r \cdot n^{u_r} \cdot \ln^{v_r} n$

We assume that the values of a_r, b_r, u_r, v_r, and w_r are given and the goal is to synthesize λ_r for every $1 \leq r \leq m$. This does not affect the generality of our approach, since u_r, v_r and w_r can be automatically synthesized by previous methods that focus on bounding expected runtimes, such as [Chatterjee et al. 2017a], and a_r and b_r are typically small integers and one can simply try all possible combinations in which they are bounded by a small threshold c. Moreover, the algorithm also sets up a template for the range of X_r :

- $B_r(n) = \gamma_r \cdot n^{p_r} \cdot \ln^{q_r} n$

In other words, we assume that the range of $X_r(n)$ is bounded by a pseudo-polynomial function $B_r(n)$. Since X_r does not depend on any of the t_j's, computing the constants γ_r, p_r and q_r can be automated. However, this is not the main focus of our algorithm. Hence, we have relegated the details to Appendix H.

Binary searching. Our goal is to synthesize t_r's that satisfy $\mathbb{E}\left[\exp(t_r(n) \cdot X_r(n))\right] \leq 1$ for every r. Each t_r is of the form $t_r(n) = \lambda_r \cdot n^{-a_r} \cdot \ln^{-b_r} n$, where the only unknown is λ_r. Note that our SPRR is non-mutual. Therefore, the algorithm first synthesizes λ_1, then uses it to synthesize λ_2 and so on. When synthesizing λ_r, we can assume that $\lambda_1, \ldots, \lambda_{r-1}$ are already known. The synthesis process itself is quite simple. The algorithm relies on a verification procedure that, given values for $\lambda_1, \ldots, \lambda_r$, decides whether (‡) holds. Using this verification procedure, the algorithm performs a binary search to find the largest possible value for λ_r.

Verification procedure. This is the most involved part of our algorithm. Given the values for $\lambda_1, \ldots, \lambda_r$, the goal is to decide whether $\mathbb{E}\left[\exp(t_r(n) \cdot X_r(n))\right] \leq 1$ for every $n \geq c_r + 1$. It is well-known that there exists $\xi \in [0, x]$ such that

$$\exp(x) = 1 + \sum_{i=1}^{L} \frac{x^i}{i!} + \frac{\exp(\xi)}{(L+1)!} x^{L+1}$$

If $-B \leq x \leq B$ and L is an odd number, then x^{L+1} is always non-negative. Hence, we have:

$$\exp(x) \leq 1 + \sum_{i=1}^{L} \frac{x^i}{i!} + \frac{\exp(B)}{(L+1)!} x^{L+1}.$$

Let us apply this inequality to $\exp(t_r(n) \cdot X_r(n))$. Suppose that $-D_r \leq t_r(n) \cdot X_r(n) \leq D_r$ for some $D_r > 0$. Then, we have:

$$\exp(t_r(n) \cdot X_r(n)) \leq 1 + \sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot X_r(n)^i + \frac{\exp(D_r) \cdot t_r(n)^{L+1}}{(L+1)!} \cdot X_r(n)^{L+1}.$$

Now, we can strengthen our target inequality $\mathbb{E}\left[\exp(t_r(n) \cdot X_r(n))\right] \leq 1$ to:

$$\mathbb{E}\left[1 + \sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot X_r(n)^i + \frac{\exp(D_r) \cdot t_r(n)^{L+1}}{(L+1)!} \cdot X_r(n)^{L+1}\right] \leq 1,$$

which is equivalent to:

$$\sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot \mathbb{E}\left[X_r(n)^i\right] + \frac{\exp(D_r) \cdot t_r(n)^{L+1}}{(L+1)!} \cdot \mathbb{E}\left[X_r(n)^{L+1}\right] \leq 0.$$

by subtracting 1 on both side. Our verification procedure tries to establish the last inequality. To achieve this goal, our algorithm performs three steps:

(1) Synthesize the bound D_r.
(2) Over-approximate each expectation term $\mathbb{E}\left[X_r(n)^i\right]$ for large enough n by a pseudo-polynomial expression $A_{r,i}(n)$.[§]
(3) Check whether the strengthened inequality

$$\sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot A_{r,i}(n) + \frac{\exp(B_r) \cdot t_r(n)^{L+1}}{(L+1)!} \cdot A_{r,L+1}(n) \leq 0$$

holds for every $n \geq \max\{\ell, c_r + 1\}$.

Step (1). Synthesizing a bound D_r. Recall that $-B_r(n) \leq X_r(n) \leq B_r(n)$ where $B_r(n) = \gamma_r \cdot n^{p_r} \cdot \ln^{q_r} n$. Our goal is to bound $t_r(n) \cdot X_r(n)$. Let

$$V(n) := B_r(n) \cdot t_r(n) = \gamma_r \cdot \lambda_r \cdot n^{p_r - a_r} \cdot \ln^{q_r - b_r} n.$$

It is easy to see that we can use the maximum of $V(n)$ as D_r. Due to the monotonicity of pseudo-monomials, the algorithm considers the following cases:

- If $p_r > a_r$ or $p_r = a_r$ and $q_r > b_r$, then the algorithm outputs $D_r = +\infty$.
- If $p_r \leq a_r$ and $q_r \leq b_r$, then $V(n)$ reaches its maximum at $n = c_r + 1$ and the algorithm outputs $D_r = V(c_r + 1)$.
- If $p_r \leq a_r - 1$ and $q_r \geq b_r + 1$, then $V(n)$ reaches its maximum at $\max\left\{c_r + 1, \frac{q_r - b_r}{a_r - p_r}\right\}$. Hence, the algorithm outputs $D_r = V\left(\max\left\{c_r + 1, \frac{q_r - b_r}{a_r - p_r}\right\}\right)$.

Step (2). Over-approximating expectation terms. In this step, the algorithm over-approximates each expectation term $\mathbb{E}\left[X_r(n)^i\right]$ by a pseudo-polynomial expression $A_{r,i}(n)$. Recall that

$$X_r(n) = Cost_r(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n)) - f_r(n).$$

The algorithm expands $\mathbb{E}\left[X_r(n)^i\right]$ by simply applying the binomial theorem to the RHS of the equality above. This leads to:

$$\mathbb{E}\left[X_r(n)^i\right] = \sum_{\mathcal{I}} i! \cdot \frac{(-1)^b \cdot f_r(n)^b}{b!} \cdot \frac{\mathbb{E}\left[Cost_r(n)^j\right]}{j!} \cdot \mathbb{E}\left[\prod_{u=1}^{S_r} \frac{f_r^{pu}(h_{r,u}(n))}{p_u!}\right] \cdot \prod_{u=1}^{O_r} \mathbb{E}\left[\frac{f_{q_{r,u}}^{qu}(g_{r,u}(n))}{q_u!}\right]$$

[§]Since the approximation does not hold for small n, our algorithm first finds an integer ℓ such that the approximation is sound for every $n \geq \ell$ and then simply iterates over $c_r + 1 \leq n \leq \ell$, evaluates the expression and trivially verifies that it satisfies the condition.

where $\mathcal{I}$ denotes $\left[0 \le b, j, p_1, \ldots, p_{S_r}, q_1, \ldots, q_{O_r} \le i, b + j + \sum_{u=1}^{S_r} p_u + \sum_{u=1}^{O_r} q_u = i\right]$. Each term is treated separately. In order to obtain an over-approximation for the LHS, the algorithm over/under-approximates each term of the sum on the RHS based on the parity of b in $(-1)^b$.

For each term, the algorithm has to estimate (i) $\mathbb{E}\left[Cost_r(n)^j \cdot \prod_{u=1}^{O_u} \frac{f_{q_{r,u}}^{q_u}(g_{r,u}(n))}{q_u!}\right]$, and (ii) $\mathbb{E}\left[\prod_{u=1}^{S_r} \frac{f_r^{p_u}(h_{r,u}(n))}{p_u!}\right]$. To approximate (i), our algorithm performs the following steps: (Recall that all random variables in (i) observe discrete distributions):

- It iterates over every possible combination of values for $Cost_r(n)$ and the $g_{r,u}(n)$'s in their respective discrete distributions and approximates $Cost_r(n)^j$ and each $f_{q_{r,u}}(g_{r,u})(n)^{q_u}$ by a pseudo-polynomial
  - $Cost_r(n)^j$ can represented exactly and straightforwardly by a pseudo-polynomial.
  - Whenever an element in the support involves $\lfloor (n+b)/2 \rfloor$, our algorithm will distinguish between even and odd n, and transform the floor/ceiling term by an affine expression precisely.
  - Now, each $g_{r,u}(n)$ is in the form of an affine expression $p \cdot n + q$ ($p, q \in \mathbb{Q}$). Plugging this into the template of f, the algorithm derives a combination of terms of the form $\ln^v(p \cdot n + q)$. It then applies Proposition 5.1 and approximates these terms by pseudo-polynomials. Note that if $q < 0$, the proposition only applies for $n \ge \frac{-2 \cdot q}{p}$. In this case, the algorithm sets $\ell := \max\{\ell, \frac{-2 \cdot q}{p}\}$ and manually checks every $c_r + 1 \le n \le \ell$ in order to handle the corner case.

We now focus on how the algorithm estimates (ii). The treatment depends on whether $S_r = 1$ or $S_r = 2$, as well as the probability distributions. Based on the assumptions made at the beginning of this section, the algorithm considers the following six cases:

- $S_r = 1$
  - $h_{r,1}(n)$ has a discrete distribution, we follow exactly the same procedure for approximating (i).
  - $h_{r,1}(n)$ is distributed uniformly: In this case, the algorithm needs to approximate
    $$\frac{1}{n} \cdot \sum_{i=0}^{n-1} i^{u_r \cdot p_1} \ln^{v_r \cdot p_1} i$$
    Relying on MomentApprox($L_T, u_r \cdot p_1, v_r \cdot p_1, n$), the algorithm computes pseudo-polynomial bounds for this term. Note that MomentApprox requires the assumption $n \ge 1 + \lceil \exp(\frac{v_r \cdot p_1}{L_T}) \rceil$, thus the algorithm sets $\ell := \max\{\ell, 1 + \lceil \exp(\frac{v_r \cdot p_1}{L_T}) \rceil\}$ and handles each $c_r + 1 \le n \le \ell$ by evaluation.
  - $h_{r,1}(n) = \max\{i, n - 1 - i\}$ where i is uniformly distributed: In this case, the algorithm should approximate
    $$\frac{1}{n} \cdot \left(\sum_{i=\lceil (n-1)/2 \rceil}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i + \sum_{i=\lfloor (n-1)/2 \rfloor}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \right).$$
    It distinguishes between even and odd n to remove floors and ceils, and then handles the two summations separately and exactly as in the previous case, applying MomentApprox. Also we need to incorporate corner cases and updates ℓ.
- $S_r = 2$ and $h_{r,1}(n) + h_{r,2}(n) = n - 1$: In all the cases below, the algorithm over-estimates $h_{r,2}(n)^{a_r} \cdot \ln^{b_r} h_{r,2}(n)$ as $h_{r,2}(n)^{a_r} \cdot \ln^{b_r} n$. This is sound since it leads to an over-approximation of $X_r(n)$ and thus strengthens the condition.
  - $h_{r,1}(n)$ has a discrete distribution. This case is handled similarly to when $S_r = 1$. The only difference is that now the algorithm has to consider both recursive calls and should consider:
    $$(p \cdot n + q)^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1}(p \cdot n + q) \cdot (n - 1 - p \cdot n - q)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_2} n.$$

– $h_{r,1}(n)$ is distributed uniformly: In this case, the algorithm has to approximate

$$\frac{1}{n} \cdot \sum_{i=0}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n.$$

To handle this expression, the algorithm first applies binomial expansion and then uses MomentApprox as in the case where $S_r = 1.$, and considers corner cases similar to previous cases.

– $h_{r,1}(n) = \max\{i, n-i-1\}$ where i is distributed uniformly: The algorithm should approximate

$$\frac{1}{n} \cdot \left(\sum_{i=\lceil (n-1)/2 \rceil}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n + \sum_{i=\lfloor (n-1)/2 \rfloor}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n \right).$$

It considers each sum separately and applies binomial expansion followed by MomentApprox as in the previous case.

Step (3). Checking the strengthened inequality. Finally, after finding the over-approximations in the previous steps, the algorithm considers the strengthened inequality

$$\sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot A_{r,i}(n) + \frac{\exp(B_r) \cdot t_r(n)^{L+1}}{(L+1)!} \cdot A_{r,L+1}(n) \leq 0$$

and computes it symbolically. The goal of this step is to decide whether this inequality holds for every $n \geq \max\{\ell, c_r + 1\}$. This step is achieved as outlined in Section 5.2 and Appendix F.

5.4 Our Algorithm for Synthesizing Polynomially Decreasing Bounds

Sketch of the algorithm. Our algorithm for synthesis of polynomial bounds is quite similar to the algorithm in Section 5.3, except that it considers a different set of constraints, i.e. the constraints of Theorem 3.6. To apply this theorem, we would need to synthesize $m \cdot (k+1)$ functions $f_{1,0}, f_{1,1}, \ldots, f_{1,k}, f_{2,0}, f_{2,1}, \ldots, f_{2,k}, \ldots, f_{m,0}, f_{m,1}, \ldots, f_{m,k}$ that satisfy the following constraints for all $1 \leq r \leq m$, $1 \leq i \leq k$ and $1 \leq n \leq n^*$:

$$\frac{f_{r,i}(n)}{i!} \geq \mathbb{E}\left[\sum_{\mathcal{I}} \frac{Cost_r(n)^j}{j!} \cdot \prod_{u=1}^{S_r} \frac{f_{r,p_u}(h_{r,u}(n))}{p_u!} \cdot \prod_{u=1}^{O_r} \frac{f_{q_{r,u},e_u}(g_{r,u}(n))}{e_u!} \right] \tag{7}$$

$$f_{r,0}(n) \geq 1 \tag{8}$$

where $\mathcal{I} := \left[0 \leq j, p_1, \ldots, p_{S_r}, e_1, \ldots, e_{O_r} \leq i, \sum_{u=1}^{S_r} p_u + \sum_{u=1}^{O_r} e_u + j = i \right]$. As in the previous algorithm, we start by setting up a template for each function $f_{r,i}$. The core of the problem is then to find values for coefficients $\lambda_{r,i}$ that appear in the templates for $f_{r,i}$. Given the non-mutuality of the input SPRR, we can solve the $\lambda_{r,i}$'s in lexicographic order. As before, we design a verification algorithm that gets values for $\lambda_{1,0}, \lambda_{1,1}, \ldots, \lambda_{1,k}, \lambda_{2,0}, \ldots, \lambda_{r,i}$ as input and decides whether (7) is violated. We then rely on this verification procedure to find each $\lambda_{r,i}$ using binary search.

Intuition behind the functions and templates. In the constraints above, each $f_{r,i}(n)$ intuitively serves as an over-approximation of $\mathbb{E}[T_r(n)^i]$. This, together with the fact that most randomized algorithms considered in this work have pseudo-polynomial runtimes, naturally leads to the idea of establishing pseudo-monomial templates for each $f_{r,i}$.

Setting up templates. In the first step, the algorithm sets up a pseduo-monomial template for each $f_{r,i}$. In other words, it assumes that each $f_{r,i}$ is in the following form:

- $f_{r,i}(n) = \lambda_{r,i} \cdot n^{u_{r,i}} \cdot \ln^{v_{r,i}} n.$

As in Section 5.3, we assume that $u_{r,i}$ and $v_{r,i}$ are given and focus on synthesizing $\lambda_{r,i}$. Note that $\lambda_{r,i}$ must be positive. As discussed at the end of Section 3, our algorithm will synthesize the bound $\lambda_{r,i} \cdot \kappa^{-k} \cdot (n^*)^{u_{r^*,k}} \cdot (\ln n^*)^{v_{r^*,k}}$. Given that κ is a pseudo-polynomial over n^*, this bound is polynomially decreasing.

Binary searching. Given that the SPRR is assumed to be non-mutual, we can solve the $\lambda_{r,i}$'s in lexicographic order $\lambda_{1,0}, \lambda_{1,1}, \ldots, \lambda_{1,k}, \lambda_{2,0}, \ldots, \lambda_{2,k}, \ldots, \lambda_{m,k}$. Assuming that we have a verification procedure (details below) that, given values for $\lambda_{1,0}, \ldots, \lambda_{r,i}$, decides whether (7) holds, the algorithm performs a binary search on $\lambda_{r,i}$. Since no upper-bound is known apriori, it first keeps doubling its guess until an upper-bound is found and then proceeds with classical binary search.

Verification procedure. As input, our verification procedure gets a guessed value for $\lambda_{r,i}$, as well as fixed values for all $\lambda_{r',i'}$'s that come lexicographically before it. It should then check whether they satisfy

$$\frac{f_{r,i}(n)}{i!} \geq \mathbb{E}\left[\sum_{\mathcal{I}} \frac{Cost_r(n)^j}{j!} \cdot \prod_{u=1}^{S_r} \frac{f_{r,p_u}(h_{r,u}(n))}{p_u!} \cdot \prod_{u=1}^{O_r} \frac{f_{q_{r,u},e_u}(g_{r,u}(n))}{e_u!}\right].$$

where $\mathcal{I} := \left[0 \leq j, p_1, \ldots, p_{S_r}, e_1, \ldots, e_{O_r} \leq i, \sum_{u=1}^{S_r} p_u + \sum_{u=1}^{O_r} e_u + j = i\right]$. The process is similar to that of Section 5.3. The algorithm iterates over every summation term (in $\mathcal{I}$) and over-approximates the term by a pseudo-polynomial. It then checks whether the strengthened inequality obtained by replacing each term with its over-approximation holds. Since checking this inequality is done in the exact same manner as in Section 5.3, we only focus on the over-approximation step.

Over-approximations. Based on linearity of expectation, for each summation term, the algorithm has to over-approximate (i) $\mathbb{E}\left[Cost_r(n)^j \cdot \prod_{u=1}^{O_u} \frac{f_{q_{r,u},e_u}(g_{r,u}(n))}{e_u!}\right]$, and (ii) $\mathbb{E}\left[\prod_{u=1}^{S_r} \frac{f_{r,p_u}(h_{r,u}(n))}{p_u!}\right]$, by pseudo-polynomials. (i) is approximated using the same procedure as in Section 5.3. To approximate (ii), the algorithm considers six cases, based on the number of recursive calls to the r-th procedure and the distribution of $h_{r,1}(n)$.

- $S_r = 1$
  - $h_{r,1}(n)$ follows a discrete distribution, we follow the same procedure for approximating (i).
  - $h_{r,1}(n)$ is distributed uniformly, The algorithm considers $\frac{1}{n} \cdot \sum_{i=0}^{n-1} i^{u_r \cdot p_1} \ln^{v_r \cdot p_1} i$. We could also follow the same procedure as we do for our algorithm in exponentially decreasing bounds.
  - $h_{r,1}(n) = \max\{i, n-i-1\}$, where i is distributed uniformly: The algorithm considers

    $$\frac{1}{n} \cdot \left(\sum_{i=\lceil (n-1)/2 \rceil}^{n-1} i^{u_r \cdot p_1} \ln^{v_r \cdot p_1} i + \sum_{i=\lfloor (n-1)/2 \rfloor}^{n-1} i^{u_r \cdot p_1} \ln^{v_r \cdot p_1} i\right)$$

    and applies the method in the previous case to each sum separately.
- $S_r = 2$: In the cases below, we always have $h_{r,2}(n) = n - 1 - h_{r,1}(n)$. Moreover, the algorithm over-approximates $h_{r,2}(n)^{a_r} \cdot \ln^{b_r} h_{r,2}(n)$ by $h_{r,2}(n)^{a_r} \cdot \ln^{b_r} n$.
  - $h_{r,1}(n)$ follows a discrete distribution: this follows exactly the same procedure as we do for our algorithm in exponentially decreasing bounds.
  - $h_{r,1}(n)$ is distributed uniformly: In this case, the algorithm should over-approximate

    $$\frac{1}{n} \cdot \sum_{i=0}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n.$$

    The over-approximation is obtained by first applying a binomial expansion and then applying MomentApprox to each term.
  - $h_{r,1}(n) = \max\{i, n-i-1\}$, where i is distributed uniformly: In this case, the algorithm combines the ideas from previous cases. It has to find an upper-bound for

    $$\frac{1}{n} \cdot \left(\sum_{i=\lceil (n-1)/2 \rceil}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n + \sum_{i=\lfloor (n-1)/2 \rfloor}^{n-1} i^{u_r \cdot p_1} \cdot \ln^{v_r \cdot p_1} i \cdot (n-i-1)^{u_r \cdot p_2} \cdot \ln^{v_r \cdot p_1} n\right).$$

    It considers each sum separately and first applies a binomial expansion. Then, it applies the MomentApprox procedure to each term.

Checking the strengthened inequality. Finally, the verification algorithm checks the strengthened inequality obtained by replacing every term in (7) by its over-approximation as a pseudo-polynomial. The process is the same as in Section 5.3 and relies on the algorithm of Section 5.2

Benchmark	SPRR	Base case threshold(s)
L1Diameter	$T(n) = n + T(\text{Uniform}(n))$	$c = 1$
L2Diameter	$T(n) = n \cdot \ln n + T(\text{Uniform}(n))$	$c = 1$
QuickSelect	$T(n) = n + T(\text{MUniform}(n))$	$c = 1$
SortSelect	$T_1(n) = n + T_1(\text{MUniform}(n))$ $T_2(n) = T_1(n) + T_2(\lceil (n-1)/2 \rceil) + T_2(\lfloor (n-1)/2 \rfloor)$	$c_1 = 1, c_2 = 1$
QuickSort	$T(n) = n + T(h_1(n)) + T(n - 1 - h_1(n))$ where $h_1(n)$ is sampled from $\text{Uniform}(n)$	$c = 1$
RandSearch	$T(n) = 1 + T(\text{MUniform}(n))$	$c = 1$
Channel	$T(n) = 1 + T\left(\left\{\frac{1}{e} : n-1;\ 1 - \frac{1}{e} : n\right\}\right)$	$c = 1$
Robot	$T(n) = \{0.5 : -0.05;\ 0.5 : 0.05\} + T(n-1)$	$c = 0$
SeidelLP in dimension d	$T_1(n) = n$ $T_r(n) = \left\{\frac{r}{n} : r \cdot n + T_{r-1}(n-1);\ 1 - \frac{r}{n} : r\right\} + T_r(n-1)$ $(2 \le r \le d)$	$c_r = r$ for every $1 \le r \le d$
Disk in dimension d	$T_1(n) = n$ $T_r(n) = 1 + T_r(n-1) + T_{r-1}(\{\frac{r}{n} : n-1;\ 1 - \frac{r}{n} : 0\})$ $(2 \le r \le d)$	$c_r = r$ for every $1 \le r \le d$

Table 1. Our Benchmark SPRRs. We use the notation $\{p_1 : expr_1, \ldots, p_k : expr_k\}$ to denote that with probability p_i the value is $expr_i$. $\text{MUniform}(n)$ denotes $\max\{i, n-1-i\}$ where i is uniformly distributed.

(Appendix F).

6 EXPERIMENTAL RESULTS

In this section, we provide experimental results over SPRRs obtained from various classical randomized algorithms. These results demonstrate that our approach is able to obtain much tighter bounds than [Karp 1994] in practice. Moreover, it can also handle a wider family of SPRRs.

Benchmarks. We evaluated the two algorithmic approaches of Section 5 over classical randomized algorithms such as QuickSort (Example 2.3), QuickSelect (Example 2.2), DiameterComputation [Motwani and Raghavan 1995, Chapter 9], RandomizedSearch [McConnell 2001, Chapter 9], ChannelConflictResolution [Kleinberg and Tardos 2006, Chapter 13], SeidelLP (Example 2.4) and SmallestEnclosingDisk [Welzl 1991]. We also considered a manually-crafted algorithm, SortingSelect, that sorts an array using a divide-and-conquer method in which in each iteration the median element is obtained by QuickSelect and used as a pivot. Due to space constraints, a detailed description of these algorithms is provided in Appendix B. Table 1 provides a summary of their SPRRs. The initial procedure r^* is always the last procedure in the SPRR and the initial input size is denoted by n^*.

Implementation and environment. We implemented our algorithms in C++ used the miracl[Ltd. 2018] library for high precision computation in MomentApprox. For QuickSort, QuickSelect, DiameterComputation, RandomizedSearch, Robot and SortingSelect, we ran our algorithm of Section 5.3, which synthesizes exponentially decreasing concentration bounds. For SeidelLP and SmallestEnclosingDisk, we ran the algorithm of Section 5.4 which synthesizes polynomially decreasing bounds. For all examples, we expand the Taylor series to depth 10 for applying MomentApprox. And for expanding the exponential template into Taylor series, we choose depth 15 for QuickSelect and DiameterComputation with $L1$ case, and depth 9 for the others. All results were obtained on an Ubuntu 18.04 machine with an 8 core (i7-7900x) processor and 40 GB of RAM.

Benchmark	Task	$f(n)$	$D(n)$	$t_i(n)$	Our symbolic bound	Time (s)	Karp's bound
L1Diameter	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$2.5 \cdot n$	$1.5 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.008 \cdot (2.5 - \alpha))$	2.28	$(\frac{1}{2})^{\alpha-2}$
		$3 \cdot n$	$2 \cdot n$		$\exp(1.540 \cdot (3.0 - \alpha))$	2.16	
		$3.5 \cdot n$	$2.5 \cdot n$		$\exp(1.774 \cdot (3.5 - \alpha))$	2.08	
L2Diameter	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*]$	$2.5 \cdot n \cdot \ln n$	$1.5 \cdot n \cdot \ln n$	$\frac{\lambda}{n \cdot \ln n}$	$\exp(1.008 \cdot (2.5 - \alpha))$	3.09	$(\frac{1}{2})^{\alpha-2}$
		$3 \cdot n \cdot \ln n$	$2 \cdot n \cdot \ln n$		$\exp(1.338 \cdot (3.0 - \alpha))$	3.10	
		$3.5 \cdot n \cdot \ln n$	$2.5 \cdot n \cdot \ln n$		$\exp(1.189 \cdot (3.5 - \alpha))$	3.10	
QuickSelect	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$5.0 \cdot n$	$3 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.007 \cdot (5.0 - \alpha))$	1.92	$(\frac{3}{4})^{\alpha-4}$
		$6 \cdot n$	$3.67 \cdot n$		$\exp(1.364 \cdot (6.0 - \alpha))$	1.98	
		$7 \cdot n$	$4.33 \cdot n$		$\exp(1.112 \cdot (7.0 - \alpha))$	1.95	
SortSelect	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*]$	$5.0 \cdot n$	$3 \cdot n$	$\frac{\lambda_1}{n}$	$\exp(1.007 \cdot (15.0 - \alpha) \cdot \ln n^*)$	1.96	-
		$15 \cdot n \cdot \ln n$	$4.20 \cdot n$				
		$6 \cdot n$	$3.67 \cdot n$	$\frac{\lambda_2}{n}$	$\exp(1.364 \cdot (18.0 - \alpha) \cdot \ln n^*)$	1.98	
		$18 \cdot n \ln n$	$5.24 \cdot n$				
		$7 \cdot n$	$4.33 \cdot n$		$\exp(1.112 \cdot (21.0 - \alpha) \cdot \ln n^*)$	2.14	
		$21 \cdot n \cdot \ln n$	$6.27 \cdot n$				
QuickSort	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^* + \beta \cdot n^*]$	$6 \cdot n \cdot \ln n$	$1.08 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.123 \cdot ((6 - \alpha) \cdot \ln n^* - \beta))$	2.54	$\exp(\frac{1-\alpha}{2})$
		$7 \cdot n \cdot \ln n$	$1.43 \cdot n$		$\exp(1.038 \cdot ((7 - \alpha) \cdot \ln n^* - \beta))$	2.57	
		$8 \cdot n \cdot \ln n$	$1.78 \cdot n$		$\exp(0.944 \cdot ((8 - \alpha) \cdot \ln n^* - \beta))$	2.53	
RandSearch	$\Pr[C_\tau \geq \alpha \cdot \ln n^*]$	$4.0 \cdot \ln n$	1.78	λ	$(n^*)^{0.446 \cdot (4.0 - \alpha)}$	2.53	$(n^*)^{1-0.287 \cdot \alpha}$
		$4.3 \cdot \ln n$	1.98		$(n^*)^{0.441 \cdot (4.3 - \alpha)}$	2.53	
		$4.5 \cdot \ln n$	2.12		$(n^*)^{0.432 \cdot (4.5 - \alpha)}$	2.52	
Channel	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$3 \cdot n$	2	λ	$\exp(0.887 \cdot (3.0 - \alpha) \cdot n^*)$	0.49	$\exp(\frac{\alpha-e}{e})$
		$4 \cdot n$	3		$\exp(0.964 \cdot (3.0 - \alpha) \cdot n^*)$	0.44	
		$5 \cdot n$	4		$\exp(0.975 \cdot (3.0 - \alpha) \cdot n^*)$	0.45	
Robot	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$0.1 \cdot n$	0.15	λ	$\exp(22.65 \cdot (0.1 - \alpha) \cdot n^*)$	0.11	-
		$0.2 \cdot n$	0.25		$\exp(14.48 \cdot (0.2 - \alpha) \cdot n^*)$	0.11	
		$0.3 \cdot n$	0.35		$\exp(10.16 \cdot (0.3 - \alpha) \cdot n^*)$	0.11	

Table 2. Symbolic Exponentially Decreasing Concentration Bounds. For each benchmark, we provide results based on several different over-approximations $f(n)$ of the expected runtime.

Benchmark	Parameters	$f(n)$	$d(n)$	Our concrete bound	Karp's concrete bound	Ratio
L1Diameter	$\alpha = 6$	$2.5 \cdot n$	$1.5 \cdot n$	0.029	0.0625	2.13
	$\alpha = 8$	$3.5 \cdot n$	$2.5 \cdot n$	$3.41 \cdot 10^{-4}$	0.015	45.81
	$\alpha = 10$	$3 \cdot n$	$2 \cdot n$	$2.079 \cdot 10^{-5}$	0.0039	187.84
L2Diameter	$\alpha = 6$	$3.5 \cdot n$	$2.5 \cdot n$	0.051	0.0625	1.21
	$\alpha = 8$	$2.5 \cdot n$	$1.5 \cdot n$	0.0039	0.015	3.97
	$\alpha = 10$	$3 \cdot n$	$2 \cdot n$	$8.51 \cdot 10^{-5}$	0.0039	45.87
QuickSelect	$\alpha = 12$	$5 \cdot n$	$3 \cdot n$	$8.641 \cdot 10^{-4}$	0.1002	115.85
	$\alpha = 16$	$7 \cdot n$	$4.33 \cdot n$	$4.503 \cdot 10^{-5}$	0.0317	703.85
	$\alpha = 20$	$6 \cdot n$	$3.67 \cdot n$	$5.089 \cdot 10^{-9}$	0.0101	$1.984 \cdot 10^{6}$
QuickSort	$\alpha = 7, \beta = 6$	$6 \cdot n \cdot \ln n$	$1.079 \cdot n$	$\exp(-1.123 \cdot \ln n^* + 6.738)$	0.050	∞ (as $n^* \to \infty$)
	$\alpha = 8, \beta = 1$	$7 \cdot n \cdot \ln n$	$1.426 \cdot n$	$\exp(-1.038 \cdot \ln n^* + 1.038)$	0.03	
	$\alpha = 9, \beta = 3$	$8 \cdot n \cdot \ln n$	$1.772 \cdot n$	$\exp(-0.944 \cdot \ln n^* + 2.832)$	0.018	
RandSearch	$\alpha = 6.5$	$4 \cdot \ln n$	1.772	$(n^*)^{-1.113}$	$(n^*)^{-0.869}$	
	$\alpha = 6.8$	$4.3 \cdot \ln n$	1.98	$(n^*)^{-1.104}$	$(n^*)^{-0.956}$	
	$\alpha = 7.0$	$4.5 \cdot \ln n$	2.11	$(n^*)^{-1.084}$	$(n^*)^{-1.013}$	
Channel	$\alpha = 5$	$3 \cdot n$	3	$\exp(-1.773 \cdot n^*)$	0.432	
	$\alpha = 6$	$4 \cdot n$	4	$\exp(-1.928 \cdot n^*)$	0.300	
	$\alpha = 7$	$5 \cdot n$	5	$\exp(-1.951 \cdot n^*)$	0.207	

Table 3. Concrete Exponentially Decreasing Concentration Bounds.

Experimental results. Tables 2–4 illustrate our experimental results over the benchmarks.

- *Exponentially decreasing concentration bounds:* Table 2 shows our symbolic exponential bounds, which were obtained using the algorithm of Section 5.3. It also provides Karp's bounds for the same SPRRs, demonstrating that our bounds are much tighter. To further illustrate this point, we provide concrete bounds (with fixed values for the parameters α and β) in Table 3. As shown in this table, our bounds consistently beat previous results. Notably, in some cases, our bound gets infinitely tighter than Karp's bound as n^* grows. Additionally, in two cases (SortSelect and Robot), Karp's method is not applicable. This is because they either involve more than one recurrence equation or have randomized preprocessing times.

Benchmark	$\mathbb{E}[T_d(n^*)]$	Task	degree k	Symbolic bound	Time(s)
SeidelLP($d=2$)	$8 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 8 \cdot n^*]$	$k=5$	$3.15 \cdot \alpha^{-5}$	0.12
			$k=6$	$5.24 \cdot \alpha^{-6}$	0.21
			$k=7$	$9.33 \cdot \alpha^{-7}$	0.22
SeidelLP($d=3$)	$36 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 36 \cdot n^*]$	$k=5$	$3.30 \cdot \alpha^{-5}$	0.25
			$k=6$	$5.55 \cdot \alpha^{-6}$	0.43
			$k=7$	$10.01 \cdot \alpha^{-7}$	0.70
Disk($d=3$)	$10 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 10 \cdot n^*]$	$k=3$	$1.487 \cdot \alpha^{-3}$	0.15
			$k=4$	$2.125 \cdot \alpha^{-4}$	0.37
			$k=5$	$3.318 \cdot \alpha^{-5}$	1.75
Disk($d=4$)	$41 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 41 \cdot n^*]$	$k=2$	$1.137 \cdot \alpha^{-2}$	0.21
			$k=3$	$1.439 \cdot \alpha^{-3}$	0.28
			$k=4$	$2.000 \cdot \alpha^{-4}$	3.40

Table 4. Polynomially Decreasing Concentration Bounds.

These examples show that our approach handles a wider set of SPRRs than previous methods.

- *Polynomially decreasing concentration bounds:* Table 4 shows our experimental results using the algorithm of Section 5.4 to obtain polynomially decreasing bounds. In all benchmarks, we let $u_{r,i} = v_{r,i} = i$ in our templates for every $1 \leq r \leq m$ and $1 \leq i \leq k$. To the best of our knowledge, we are providing the first algorithm for synthesizing polynomial bounds. As such, there are no previous methods to compare against. Moreover, previous methods for exponential bounds do not yield any results on these benchmarks. We believe this is probably because the best possible concentration bounds are polynomially decreasing.

Efficiency. As shown in Tables 2 and 4, our algorithms are extremely efficient in practice and can synthesize much tighter bounds in a matter of a few seconds. In our experiments, the maximum runtime was only 3.40 seconds, while the average runtime was 1.38 seconds.

7 RELATED WORKS

In this section, we compare our approach with the most relevant results in the literature.

Expected runtime analysis. Expected runtime analysis of probabilistic recurrences has been widely considered in e.g. [Bazzi and Mitter 2003; Chatterjee et al. 2017a; Flajolet et al. 1991; Zimmermann and Zimmermann 1989]. However, methods for expected runtime analysis cannot be adapted to concentration bound analysis, since it is a much more intricate problem.

Karp's cookbook. For the concentration bound analysis of probabilistic recurrences, the most well-known and related result is the classical method of [Karp 1994], where it is proven that concentration bounds of a wide class of probabilistic recurrences can be directly computed by a mathematical formula. Compared with [Karp 1994], our algorithmic approaches can derive much tigher concentration bounds over probabilistic recurrences for classical randomized algorithms including QuickSort, QuickSelect, RandomizedSearch and DiameterComputation. Furthermore, our algorithmic approaches can handle systems of probabilistic recurrences with randomized preprocessing time, such as SeidelLP and SmallestEnclosingDisk, which are beyond the scope of Karp's method and other existing approaches such as [Chaudhuri and Dubhashi 1997; McDiarmid and Hayward 1996; Tassarotti 2017; Tassarotti and Harper 2018].

Previous results on concentration bounds. The work [Chaudhuri and Dubhashi 1997] weakens several conditions required by [Karp 1994], but obtains bounds that are theoretically worse than [Karp 1994]. Moreover, [Chaudhuri and Dubhashi 1997] cannot handle probabilistic recurrences with randomized preprocessing time and systems of probabilistic recurrences. Thus, the comparison between our approach and [Karp 1994] carries over to [Chaudhuri and Dubhashi

1997]. The work [McDiarmid and Hayward 1996] performs an involved ad-hoc manual analysis to derive the asymptotically optimal concentration bound for QuickSort. In contrast, our approach is automated for a wide class of probabilistic recurrences and derives a concentration bound for the QuickSort that is slightly worse than [McDiarmid and Hayward 1996]. The work [Tassarotti and Harper 2018] mechanizes the proof of the concentration bound for QuickSort in Coq, thus it is incomparable to our approach. Finally, the recent results in [Tassarotti 2017] introduce a new approach for solving work and span runtime of parallel randomized algorithms. The method of [Tassarotti 2017] is manual and can neither handle randomized preprocessing time nor systems of probabilistic recurrences. Moreover, our automated approach can derive much tighter concentration bounds for QuickSort against this method: our automatic approach generate the concentration bound $\Pr[T(n) \geq 11 \cdot n^* \ln n^* + 12 \cdot n^*] \leq \exp(-5.615 \ln n^* - 13.476)$, while this method outputs $\Pr[T(n) \geq 11 \cdot n^* \ln n^* + 12 \cdot n^*] \leq \exp(-0.24 \cdot \ln n^* - 1.47)$.

Non-probabilistic Recurrence Relations. There many results on the formal analysis of classical non-probabilistic recurrence relations in the literature. For example, [Wegbreit 1975] considered solving recurrence relations through difference equations and generating functions. [Zimmermann and Zimmermann 1989] also considered transforming them into difference equations. Grobauer [Grobauer 2001] uses generating recurrence relations from DML for the worst-case analysis. The COSTA project [Albert et al. 2009, 2008, 2007] transforms Java bytecode into recurrence relations and solves them through ranking functions. The PURRS tool [Bagnara et al. 2005] addresses finite linear recurrences (with bounded summation), and some restricted linear infinite recurrence relations (with unbounded summation). Recently, new approaches are proposed for solving non-probabilistic recurrence relations through compositional approaches [Farzan and Kincaid 2015; Kincaid et al. 2017], abstract interpretation [Kincaid et al. 2018] and templates [Breck et al. 2020]. Since these approaches do not consider probability, they cannot be applied to the formal analysis of probabilistic recurrence relations.

Probabilistic programs. There are also many relevant results in formal analysis of probabilistic programs, such as martingale-based approaches of [Chatterjee and Fu 2017; Chatterjee et al. 2016, 2018a,b, 2017b; Fioriti and Hermanns 2015; Fu and Chatterjee 2019; Huang et al. 2018, 2019; Kura et al. 2019; McIver et al. 2017; Wang et al. 2020b,a, 2019]. These approaches first synthesize linear/polynomial (ranking) supermartingales through constraint solving, and then use concentration inequalities to derive concentration bounds of probabilistic programs. Compared with these results, our approaches synthesize Taylor supermartingale functions in exponential and polynomial form by Taylor expansion and tight approximation of expectation of moments with logarithm, thus they are completely different, both in terms of the problem they solve and the employed techniques.

8 CONCLUSION

In this work, we considered the problem of formal analysis of concentration bounds of probabilistic recurrence relations. We first proposed a general theoretical approach through Markov's inequality and Optional Stopping Theorem. The core notion in our general approach is that of Taylor supermaringale function sequences, which establish supermartingale conditions through Taylor series. Then, we introduced two algorithmic approaches that synthesize exponential and polynomial supermartingales for deriving the concentration bounds. Our experimental results demonstrate that our algorithmic approaches can obtain much tighter concentration bounds than existing approaches and can also handle a wider variety of recurrence relations.

Since the two methods are rather dissimilar, an interesting future direction would be to explore the integration of our approaches and the classical cookbook method by Karp in order to obtain even tighter bounds. Another interesting direction is to automate the process of obtaining SPRRs from probabilistic programs and randomized algorithms, similar to how the COSTA project [Albert

et al. 2009, 2008, 2007] translates Java bytecode into non-probabilistic recurrences. While this work provides automated approaches for concentration bound analysis of SPRRs, the translation from a program to an SPRR is still performed manually. The SPRRs for classical algorithms are well-known and widely studied, but there is still no automated method to obtain SPRRs for arbitrary programs.

REFERENCES

Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL* 2, POPL (2018), 34:1–34:32.

Mohamad A. Akra and Louay Bazzi. 1998. On the Solution of Linear Recurrence Equations. *Comput. Optim. Appl.* 10, 2 (1998), 195–210.

Elvira Albert, Puri Arenas, Samir Genaim, Miguel Gómez-Zamalloa, German Puebla, Diana V. Ramírez-Deantes, Guillermo Román-Díez, and Damiano Zanardini. 2009. Termination and Cost Analysis with COSTA and its User Interfaces. *Electr. Notes Theor. Comput. Sci.* 258, 1 (2009), 109–121.

Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. 2008. Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. In *SAS 2008*. 221–237.

Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. 2007. Cost Analysis of Java Bytecode. In *ESOP 2007*. 157–172.

Roberto Bagnara, Andrea Pescetti, Alessandro Zaccagnini, and Enea Zaffanella. 2005. PURRS: Towards Computer Algebra Support for Fully Automatic Worst-Case Complexity Analysis. *CoRR* abs/cs/0512056 (2005). arXiv:cs/0512056 http://arxiv.org/abs/cs/0512056

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

Louay Bazzi and Sanjoy K. Mitter. 2003. The Solution of Linear Probabilistic Recurrence Relations. *Algorithmica* 36, 1 (2003), 41–57.

Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer.

P Billingsley. 1995. *Probability and Measure* (3rd ed.). Wiley.

Jason Breck, John Cyphert, Zachary Kincaid, and Thomas W. Reps. 2020. Templates and recurrences: better together. In *PLDI*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 688–702.

Krishnendu Chatterjee and Hongfei Fu. 2017. Termination of Nondeterministic Recursive Probabilistic Programs. *CoRR* abs/1701.02944 (2017).

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

Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Nastaran Okati. 2018a. Computational Approaches for Stochastic Shortest Path on Succinct MDPs. In *IJCAI 2018*. 4700–4707.

Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. 2017a. Automated Recurrence Analysis for Almost-Linear Expected-Runtime Bounds. In *CAV*. 118–139.

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018b. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *TOPLAS* 40, 2 (2018), 7:1–7:45.

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

Shiva Chaudhuri and Devdatt P. Dubhashi. 1997. Probabilistic Recurrence Relations Revisited. *Theoretical Computer Science* 181, 1 (1997), 45–56.

Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. 1993. Four Results on Randomized Incremental Constructions. *Comput. Geom.* 3 (1993), 185–212.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms (3. ed.)*. MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press.

Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *FMCAD*, Roope Kaivola and Thomas Wahl (Eds.). IEEE, 57–64.

Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL*. 489–501.

Philippe Flajolet, Bruno Salvy, and Paul Zimmermann. 1991. Automatic Average-Case Analysis of Algorithm. *Theor. Comput. Sci.* 79, 1 (1991), 37–109.

Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *VMCAI*. 468–490.

Bernd Grobauer. 2001. Cost Recurrences for DML Programs. In *ICFP*, Benjamin C. Pierce (Ed.). ACM, 253–264.

C. A. R. Hoare. 1961a. Algorithm 64: Quicksort. *Commun. ACM* 4, 7 (1961), 321.

C. A. R. Hoare. 1961b. Algorithm 65: find. *Commun. ACM* 4, 7 (1961), 321–322.

Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *APLAS*. 181–201.

Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2019. Modular verification for almost-sure termination of probabilistic programs. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 129:1–129:29.

David R. Karger. 1993. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *SODA*, Vijaya Ramachandran (Ed.). ACM/SIAM, 21–30.

Richard M. Karp. 1991. An introduction to randomized algorithms. *Discret. Appl. Math.* 34, 1-3 (1991), 165–201.

Richard M. Karp. 1994. Probabilistic Recurrence Relations. *Journal of the ACM* 41, 6 (1994), 1136–1150.

Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas W. Reps. 2017. Compositional recurrence analysis revisited. In *PLDI*, Albert Cohen and Martin T. Vechev (Eds.). ACM, 248–262.

Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.* 2, POPL (2018), 54:1–54:33.

Jon M. Kleinberg and Éva Tardos. 2006. *Algorithm design.* Addison-Wesley.

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *TACAS (LNCS)*, Tomás Vojnar and Lijun Zhang (Eds.), Vol. 11428. Springer, 135–153.

MIRACL Ltd. 2018. https://github.com/miracl/MIRACL.

Hosam Mahmoud. 2008. *Pólya urn models.* CRC press.

Jeffrey J. McConnell (Ed.). 2001. *The Analysis of Algorithms: An Active Learning Approach.* Jones & Bartlett Learning.

Colin McDiarmid and Ryan Hayward. 1996. Large Deviations for Quicksort. *Journal of Algorithms* 21, 3 (1996), 476–507.

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2017. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 33.

Robert H. Sr. Morris. 1978. Counting Large Numbers of Events in Small Registers. *Commun. ACM* 21, 10 (1978), 840–842.

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms.* Cambridge University Press.

Franco P. Preparata and David E. Muller. 1979. Finding the Intersection of n Half-Spaces in Time O(n log n). *Theor. Comput. Sci.* 8 (1979), 45–55. https://doi.org/10.1016/0304-3975(79)90055-0

Jeffrey S Rosenthal. 2006. *A First Look at Rigorous Probability Theory* (2nd ed.). World Scientific Publishing Company.

Raimund Seidel. 1991. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry* 6, 3 (1991), 423–434.

Joseph Tassarotti. 2017. Probabilistic Recurrence Relations for Work and Span of Parallel Algorithms. *CoRR* abs/1704.02061 (2017). http://arxiv.org/abs/1704.02061

Joseph Tassarotti and Robert Harper. 2018. Verified Tail Bounds for Randomized Programs. In *ITP*. 560–578.

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2020b. Tail Bound Analysis for Probabilistic Programs via Central Moments. *CoRR* abs/2001.10150 (2020). https://arxiv.org/abs/2001.10150

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

Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost analysis of nondeterministic probabilistic programs. In *PLDI*. 204–220.

Ben Wegbreit. 1975. Mechanical Program Analysis. *Commun. ACM* 18, 9 (1975), 528–539.

Emo Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science, Proceedings [on occasion of H. Maurer's 50th birthday] (LNCS)*, Hermann A. Maurer (Ed.), Vol. 555. Springer, 359–370.

David Williams. 1991. *Probability with Martingales.* Cambridge university press.

Paul Zimmermann and Wolf Zimmermann. 1989. *The automatic complexity analysis of divide-and-conquer algorithms.* Technical Report. INRIA.

A AN ILLUSTRATIVE EXAMPLE OF A MARTINGALE

Example A.1 (Pólya's Urn [Mahmoud 2008]). As a more interesting example, consider an urn that initially contains R_0 red and B_0 blue marbles ($R_0 + B_0 > 0$). At each step, we take one marble from the urn, chosen uniformly at random, look at its color and then add two marbles of that color to the urn. Let B_n, R_n and M_n respectively be the number of red, blue and all marbles after n steps. Also, let $\beta_n = \frac{B_n}{M_n}$ and $\rho_n = \frac{R_n}{M_n}$ be the proportion of marbles that are blue (resp. red) after n steps. Let $\mathcal{F}_n$ model the observations until the n-th step. The process described above leads to the following equations:

$$M_{n+1} = 1 + M_n,$$
$$\mathbb{E}\left[B_{n+1} \mid \mathcal{F}_n\right] = \mathbb{E}\left[B_{n+1} \mid B_1, \ldots, B_n\right] = \frac{B_n}{M_n} \cdot (B_n + 1) + \frac{R_n}{M_n} \cdot B_n,$$
$$\mathbb{E}\left[R_{n+1} \mid \mathcal{F}_n\right] = \mathbb{E}\left[R_{n+1} \mid B_1, \ldots, B_n\right] = \frac{R_n}{M_n} \cdot (R_n + 1) + \frac{B_n}{M_n} \cdot R_n.$$

Note that we did not need to care about observing R_i's, M_i's, β_i's or ρ_i's, because they can be uniquely computed in terms of B_i's. More generally, an observer can observe only B_i's, or only R_i's, or only β_i's or ρ_i's and can then compute the rest using this information. Based on the equations above, we have:

$$\mathbb{E}\left[\beta_{n+1} \mid \mathcal{F}_n\right] = \frac{B_n}{M_n} \cdot \frac{B_n + 1}{M_n + 1} + \frac{M_n - B_n}{M_n} \cdot \frac{B_n}{M_n + 1} = \frac{B_n}{M_n} = \beta_n,$$
$$\mathbb{E}\left[\rho_{n+1} \mid \mathcal{F}_n\right] = \frac{R_n}{M_n} \cdot \frac{R_n + 1}{M_n + 1} + \frac{M_n - R_n}{M_n} \cdot \frac{R_n}{M_n + 1} = \frac{R_n}{M_n} = \rho_n.$$

Hence, both $\{\beta_n\}_{n \in \mathbb{N}_0}$ and $\{\rho_n\}_{n \in \mathbb{N}_0}$ are martingales. Informally, this means that the expected proportion of blue marbles in the next step is exactly equal to their observed proportion in the current step. This might be counter-intuitive. For example, consider a state where 99% of the marbles are blue. Then, it is more likely that we will add a blue marble in the next state. However, this is mitigated by the fact that adding a blue marble changes the proportions much less dramatically than adding a red marble.

B EXAMPLES OF (SYSTEMS OF) PROBABILISTIC RECURRENCE RELATIONS

Example B.1 (QuickSelect). Consider the problem of finding the d-th smallest element in an unordered array of n distinct elements. A classical randomized algorithm for solving this problem is QuickSelect [Hoare 1961b]. It begins by choosing a pivot element u of the array uniformly at random. It then compares all the other $n - 1$ elements of the array with u and divides them into two parts: (i) those that are smaller than u and (ii) those that are larger. Suppose that there are d' elements in part (i). If $d' < d - 1$, then the algorithm recursively searches for the $(d - d' - 1)$-th smallest element of part (ii). If $d' = d - 1$, the algorithm terminates by returning u as the desired answer. Finally, if $d' > d - 1$, the algorithm recursively finds the d−th smallest element in part (i). Note that the classical median selection algorithm is a special case of QuickSelect. While there is an involved deterministic algorithm for solving the same problem, more involved linear-time non-randomized algorithms exist for the same problem, QuickSelect provides a simple randomized variant matching the best-known $O(n)$ deterministic runtime. Since there is only one recursive procedure in QuickSelect, we model the algorithm as the following SPRR with one recurrence equation (where we have $m = 1$, $T_1 = T$, $Cost_1(n) = n - 1$, $S_1 = 1$ and $O_1 = 0$):

$$T(n) = n + T(h(n)) \quad .$$

Here, $T(n)$ represents the number of comparisons performed by QuickSelect over an input of size n, and $h(n)$ is the random variable that captures the size of the remaining array that has to be

searched recursively. It can be derived that $h(n) = \max\{i-1, n-i\}$ where i is sampled uniformly from $\{1, \ldots, n\}$. □

Example B.2 (QuickSort). Consider the classical problem of sorting an array of n distinct elements. A well-known randomized algorithm for solving this problem is QuickSort. Similar to QuickSelect, QuickSort begins by choosing a pivot element u of the array uniformly at random. It then compares all the other $n-1$ elements of the array with u and divides them into two sub-arrays: (i) those that are smaller than u and (ii) those that are larger. The algorithm then runs recursively on both sub-arrays. Compared with its deterministic counterpart, MergeSort, QuickSort provides a simple randomized variant whose expected runtime matches the best-known deterministic runtime $O(n \log n)$.

Just as before, there is only one recursive procedure in QuickSort. Thus, we model the algorithm as the following SPRR with only one recurrence equation (where we have $m = 1$, $Cost_1(n) = n - 1$, $S_1 = 2$, $O_1 = 0$):

$$T(n) = n + T(h_1(n)) + T(h_2(n)) \quad .$$

Here, $h_1(n)$ and $h_2(n)$ are random variables that capture the sizes of the two sub-arrays. It is easy to see that $h_1(n) = i$ and $h_2(n) = n - 1 - i$ where i is uniformly sampled from $\{0, \ldots, n-1\}$. □

Example B.3 (SeidelLP). Consider the classic linear programming problem in d dimensions, with d variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d)$ and n constraints $\mathbf{a}_1^T \cdot \mathbf{x} \leq b_1, \ldots, \mathbf{a}_n^T \cdot \mathbf{x} \leq b_n$, and the objective function to be minimized is $\mathbf{c}^T \cdot \mathbf{x}$. SeidelLP is a famous randomized incremental algorithm for linear programming. The algorithm is as follows, it picks a constraint $\mathbf{a}_i^T \cdot \mathbf{x} \leq b_i$ from all n constraints uniformly at random and recursively solves the linear programming instance without the chosen constraint. Given the optimum x^* produced by recursion, the algorithm analyzes two cases:

Case 1. If x^* still satisfies the ignored constraint $\mathbf{a}_i \cdot \mathbf{x} \leq b_i$, then x^* is still optimal. The checking required d steps of computation.

Case 2. Otherwise, if x^* violates the ignored constraint, then the new optimum must lie in the hyperplane $\mathbf{a}_i \cdot \mathbf{x} = b_i$, in this case, we could project all other constraints onto this hyperplane and recursively solves the subinstance. Note that since the hyperplane $\mathbf{a}_i \cdot \mathbf{x} = b_i$ has dimension $d-1$, the subinstance is a linear programming in $d-1$ dimensions.

The recursion stops when the number of constraints n equals to the dimension d, where in this case the optimum could be directly computed. Note that the probability of Case 2 above is no more than $\frac{d}{n}$, since there is no more than d tight constraints in a d-dimensional LP instance.

To model the algorithm above by SPRRs, we could set up a system with d relations $eq_1, eq_2, \ldots, eq_d$, where eq_r corresponds to the r-dimensional recursion subinstaces in the algorithm. We have that $c_r = r$ and for every $n \geq r + 1$:

$$T_r(n) = Cost_r(n) + T_r(h_{r,1}(n)) + T_{r-1}(g_{r,1}(n)) \tag{9}$$

Where $Cost_r(n)$ observes as a discrete distribution: with probability $1 - \frac{r}{n}$, its value is d and with probability $\frac{r}{n}$, its value is $r \cdot n$, which corresponds the cost for checking and projection in the two cases above, $h_{r,1}(n)$ deterministically equals to $n-1$, and the $g_{r,1}(n)$ also observes as discrete distribution: with probability $1 - \frac{r}{n}$ its value is 0 and with probability $\frac{r}{n}$ its value is $n-1$, since in Case 1 discussed above, there is no need to recursively calculate a $d-1$-dimensional subinstance while in the Case 2 we need to solve such a subinstance with $n-1$ constraints.

Example B.4 (DiameterComputation). Consider the DiameterComputation algorithm [Motwani and Raghavan 1995, Chapter 9] to compute the diameter of an input finite set S of three-dimensional points. Depending on the metric, i.e. Eucledian or L_1, we obtain two different recurrence relations. For Euclidean, we have the following SPRR:

$$T(n) = n \cdot \ln n + T(h(n)) \tag{10}$$

with $c = 0$, and where $h(n)$ observes as Uniform(n).For the L_1 metric, the SPRR is as follows:

$$T(n) = n + T(h(n)) \tag{11}$$

with $c = 0$, and where $h(n)$ observes as Uniform(n). Note that here we use deterministic versions for the subroutine HalfspaceIntersection as required in DiameterComputation, see [Preparata and Muller 1979] for the Euclidean case and [Motwani and Raghavan 1995, Problem 9.6, Page 276] for the L_1 case. □

Example B.5 (RandomizedSearch). Consider Sherwood's RandomizedSearch algorithm (cf. [McConnell 2001, Chapter 9]). The algorithm checks whether an integer value d is present within the index range $[i, j]$ $(0 \leq i \leq j)$ in an integer array ar which is sorted in increasing order and is without duplicate entries. The algorithm outputs either the index of the item or -1 if d is not present in the index range $[i, j]$. The SPRR for this example contains only one recurrence equation:

$$T(n) = 1 + T(h(n)) \tag{12}$$

with $c = 0$ and $h(n) = \max\{i, n - 1 - i\}$, where i is uniformly distributed over $\{0, 1, \ldots, n - 1\}$. □

Example B.6 (SmallestEnclosingDisk). Consider the SmallestEnclosingDisk problem from computational geometry: given n points in a two-dimensional plane, the goal is to find a smallest circle that covers all the points. Note that any such circle can be identified by three input points that are on the circle. We call these the *canonical* points¶. To solve this problem, Welzl [Welzl 1991] has proposed a randomized algorithm that runs in linear expected time and searches for suitable canonical points. The algorithm SmallestEnclosingDisk is split into three procedures. The main procedure, $proc_3$, chooses an input point x uniformly at random. It then guesses that the chosen point is not canonical and tries to form a smallest enclosing disk by finding three canonical points amongst the other $n - 1$ points. If this fails, then the algorithm knows that x is canonical and calls $proc_2$, which is a procedure that gets one canonical point, in this case x, as input and tries to find two other canonical points. This leads to the following recurrence:

$$T_3(n) = 1 + T_3(n - 1) + T_2(g_{31}(n)) \tag{eq_3}$$

where $T_3(n)$ is the running time of the main procedure, $proc_3$, and $T_2(n)$ is the running time of $proc_2$. Moreover, $g_{31}(n)$ is the size of the random instance passed to $proc_2$ and is equal to zero with probability $\frac{n-3}{n}$, i.e. when the algorithm could successfully find three canonical points other than x, and equal to $n - 1$ with probability $\frac{3}{n}$, i.e. when x must be canonical.

The procedure $proc_2$ is in turn very similar to $proc_3$. It chooses a point uniformly at random and throws it out, trying to find the canonical points among the other $n - 1$. If this fails, it calls $proc_1$, which is a procedure that, given a fixed pair of canonical points, finds the final canonical point (or reports that it is impossible to do so).

$$T_2(n) = 1 + T_2(n - 1) + T_1(g_{21}(n)) \tag{eq_2}$$

As before, $g_{21}(n)$ is the random size of the instance passed to $proc_1$. It equals $n - 1$ with probability $\frac{2}{n}$ and zero with probability $\frac{n-2}{n}$. Finally, $proc_1$ is a simple linear search that can be modeled by this recurrence:

$$T_1(n) = 1 + T_1(n - 1) \tag{eq_1}$$

¶The canonical points are not necessarily unique.

Putting everything together, we obtain the following SPRR with three procedures to describe the runntime behavior of SmallestEnclosingDisk:

$$T_1(n) = 1 + T_1(n-1) \tag{eq_1}$$
$$T_2(n) = 1 + T_2(n-1) + T_1(g_{21}(n)) \tag{eq_2}$$
$$T_3(n) = 1 + T_3(n-1) + T_2(g_{31}(n)) \tag{eq_3}$$

□

Example B.7 (ChannelConflictResolution). We consider two network scenarios in which n clients are trying to get access to a network channel. This problem is also called Resource-Contention Resolution [Kleinberg and Tardos 2006, Chapter 13]. In this problem, if more than one client tries to access the channel, then no client can access it, and if exactly one client requests access to the channel, then the request is granted. In the concurrent setting, the clients share one variable, which is the number of clients which have not yet been granted access. Also in this scenario, once a client gets an access the client does not request for access again. For this problem, we obtain an over-approximating recurrence relation

$$T(n) = 1 + T(h(n)) \tag{13}$$

where $h(n)$ observes as a discrete distribution with probability $\frac{1}{e}$ to be $n-1$, and probability $1 - \frac{1}{e}$ to be n. □

Example B.8 (Sorting by QuickSelect). Consider a sorting algorithm which selects the median through the QuickSelect algorithm. The SPRR for this example is:

$$T_1(n) = n + T_1(h_1(n)) \tag{14}$$
$$T_2(n) = T_1(n) + T_2(\lceil (n-1)/2 \rceil) + T_2(\lfloor (n-1)/2 \rfloor) \tag{15}$$

where T_1 is the SPRR for QuickSelect (See. Example B.1). □

Example B.9 (Robot). Consider the execution of a Robot dead reckoning approach, as used in probabilistic robotics. Dead reckoning is an approach for position estimation starting from a known fixed position at time $t = 0$. In each time unit, there is 0.5 probability that the estimation error is -0.05, and 0.5 probability that the estimation error is $+0.05$. We use $T(n)$ to model the total accumulated error in n time units. Thus, the recurrence relation is:

$$T(n) = Cost(n) + T(n-1)$$

with $c = 0$. $Cost(n)$ observes as a discrete distribution with 0.5 probability of being -0.05 and 0.5 probability of being $+0.05$.

C FORMAL CONSTRUCTION OF THE PROBABILITY SPACE IN AN SPRR'S SEMANTICS

Given an initial stack β^*[‖], the probability space $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ for an SPRR Δ follows standard definitions (cf. [Baier and Katoen 2008, Chapter 10]) so that (i) the sample space Ω_Δ is defined as the set of all infinite sequences (conventionally called *runs*) of stacks, (ii) $\mathcal{F}_\Delta$ is the smallest sigma-algebra generated from the set of all *cylinder* sets for which a cylinder set is the set of all runs (of stacks) sharing some common finite prefix, and (iii) $\Pr_{\Delta,\beta^*}$ is the unique probability measure such that for every cylinder set A specified by the common finite prefix $\beta_0 \ldots \beta_k$, we have $\Pr_{\Delta,\beta^*}(A) = 0$ if $\beta_0 \neq \beta^*$, and $\Pr_{\Delta,\beta^*}(A) = \prod_{j=0}^{k-1} \mathbf{P}(\beta_j, \beta_{j+1})$ if $\beta_0 = \beta^*$.

[‖]Note that β^* is typically a single main procedure call σ^*.

D DETAILED PROOF IN SECTION 3

D.1 Full proof of Theorem 3.4

In [Agrawal et al. 2018], conditional expectation is extended to non-negative random variables that are not necessarily integrable. A random variable X is *non-negative* if $X(\omega) \geq 0$ for all elements ω in the sample space.

Theorem D.1 ([Agrawal et al. 2018, Proposition 3.1]). *Let X be any non-negative random variable from a probability space $(\Omega, \mathcal{F}, \Pr)$ and $\mathcal{G}$ be a sub-sigma-algebra of $\mathcal{F}$ (i.e., $\mathcal{G} \subseteq \mathcal{F}$). Then there exists a random variable $\mathbb{E}[X \mid \mathcal{G}]$ from $(\Omega, \mathcal{F}, \Pr)$ (called a conditional expectation of X w.r.t $\mathcal{G}$) that fulfills the conditions (E1) and (E3) above. Moreover, the conditional expectation is almost-surely unique, i.e., for any random varaibles Y, Z from $(\Omega, \mathcal{F}, \Pr)$ that both fulfill (E1) and (E3) we have that $\Pr(Y = Z) = 1$.*

To prove Theorem 3.4, we need the extension of conditional expectation to non-negative random variables that are not necessarily integrable, as is illustrated in Theorem D.1. One issue that arises in extending the traditional notion of conditional expectation is whether the basic properties for conditional expectation still hold for non-negative random variables that are not necessarily integrable. Below we show that several basic properties we utilize in the proof of Theorem 3.4 is still valid upon the extension.

Below we fix a probability space $(\Omega, \mathcal{F}, \Pr)$ and a sub-sigma-algebra $\mathcal{G} \subseteq \mathcal{F}$. The following properties hold for *non-negative* random variables X, Y, Z and $X_0, \ldots, X_n, \ldots$ from $(\Omega, \mathcal{F}, \Pr)$.

(E4) $\mathbb{E}[\mathbb{E}[X \mid \mathcal{G}]] = \mathbb{E}[X]$. This follows directly from (E3) where we take A to be the whole sample space Ω.

(E5) If X is $\mathcal{G}$-measurable, then $\mathbb{E}[X \mid \mathcal{G}] = X$ a.s. This follows from (E1), (E3) and the almost-sure uniqueness in Theorem D.1.

(E6) (**Linearity**) $\mathbb{E}[a \cdot X + b \cdot Y \mid \mathcal{G}] = a \cdot \mathbb{E}[X \mid \mathcal{G}] + b \cdot \mathbb{E}[Y \mid \mathcal{G}]$ a.s. for all $a, b \geq 0$. This follows from the linearity of integration.

(E7) (**Positivity**) $\mathbb{E}[X \mid \mathcal{G}] \geq 0$ (a.s.). This follows from exactly the same proof under '**A Positivity Result**' on [Williams 1991, Page 87].

(E8) (**cMON**) If we have $X_n \leq X_{n+1}$ (a.s.) for all $n \geq 0$ and $\lim_{n\to\infty} X_n = X$ (a.s.), then it holds that $\mathbb{E}[X_n \mid \mathcal{G}] \leq \mathbb{E}[X_{n+1} \mid \mathcal{G}]$ (a.s.) and $\lim_{n\to\infty} \mathbb{E}[X_n \mid \mathcal{G}] = \mathbb{E}[X \mid \mathcal{G}]$ (a.s.). This follows exactly the same lines under 'Proof of (e)' on [Williams 1991, Page 89].

(E9) ('**Taking out what is known**') If Z is $\mathcal{G}$-measurable, then $\mathbb{E}[Z \cdot X \mid \mathcal{G}] = Z \cdot \mathbb{E}[X \mid \mathcal{G}]$ a.s. The proof is exactly the same as under 'Proof of (j)' on [Williams 1991, Page 90] for the non-negative case.

Below we present the proof for our main theorem.

Theorem 3.4. Let Δ be a given SPRR. Consider a T-function $f \in \mathbb{T}$ and an infinite sequence Ψ of functions $\Psi = \Psi_0, \Psi_1, \Psi_2, \ldots$ such that each Ψ_k maps a stack β of Δ to its corresponding non-negative real value $\Psi_k(\beta)$. Suppose that the sequence Ψ satisfies the following conditions:

(A1) for every $k \geq 0$, we have $\Psi_k(\varepsilon) \geq f^{(k)}(0)$;

(A2) for every stack β and $k \geq 0$, we have $\Psi_k(\beta) \geq \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{k+j}(next_\beta)}{j!} \cdot Cost^j(\beta)\right]$.

Then for every stack β and $k \geq 0$, it holds that $\Psi_k(\beta) \geq \mathbb{E}_\beta\left[f^{(k)}(C)\right]$.

Full proof of Theorem 3.4. Fix an initial stack β^* and consider the random trace $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2, \ldots$ for the SPRR Δ with the initial stack $\widehat{\beta}_0 = \beta^*$. For each $k \geq 0$, we define $\mathcal{G}_k$ as the smallest sigma-algebra that makes the random variables in $\widehat{\beta}_0, \ldots, \widehat{\beta}_k$ and $C_0, \ldots, C_{k-1}$ measurable. It is straightforward to observe that $\{\mathcal{G}_k\}_{k\geq 0}$ is a filtration.

For each $k, i \geq 0$, we define the random variable

$$Y_{k,i} := \sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_k)}{j!} \cdot C_k^j$$

where we assume $0^0 := 1$. We prove that for every $k, i \geq 0$, the following properties hold:

(a) for every $k, i \geq 0$, $Y_{k,i}$ is non-negative;

(b) $Y_{\tau,i} \geq f^{(i)}(C_\tau)$;

(c) For every $i \geq 0$, we have $Y_{k,i}$ forms a supermartingale with respect to the filtration $\{\mathcal{G}_k\}_{k\geq 0}$.

First, we have that the property (a) follows directly from the definition of $Y_{k,i}$. Second, as $\widehat{\beta}_\tau = \varepsilon$, the property (b) can be deduced from (A1):

$$Y_{\tau,i} = \sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_\tau)}{j!} \cdot C_\tau^j \geq \sum_{j=0}^{\infty} \frac{f^{(i+j)}(0)}{j!} \cdot C_\tau^j = f^{(i)}(C_\tau).$$

Below we prove the property (c). By Theorem D.1, we can take a conditional expectation $\mathbb{E}[Y_{k+1,i} \mid \mathcal{G}_k]$ of $Y_{k+1,i}$ w.r.t $\mathcal{G}_k$ and derive that the following equalities hold (a.s.):

$$\begin{aligned}
\mathbb{E}[Y_{k+1,i} \mid \mathcal{G}_k] &= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{j!} \cdot C_{k+1}^j \,\middle|\, \mathcal{G}_k\right] \\
&= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{j!} \cdot (C_k + Cost(\widehat{\beta}_k))^j \,\middle|\, \mathcal{G}_k\right] \\
&= \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{i+j}(\widetilde{\beta}_{k+1})}{j!} \cdot \sum_{l=0}^{j} \binom{j}{l} \cdot C_k^l \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \\
&= \mathbb{E}\left[\sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \\
&= \text{§ By applying (E6)-(\textbf{Linearity}) and (E8)-(\textbf{cMON}) §} \\
&\quad \sum_{l=0}^{\infty} \mathbb{E}\left[\frac{C_k^l}{l!} \cdot \sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \\
&= \text{§ By applying (E9)-(`\textbf{Taking out what is known}') §} \\
&\quad \sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] .
\end{aligned}$$

Furthermore, we have that the following holds (a.s.):

$$\begin{aligned}
\mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] &= \text{§ By applying (E6)-(\textbf{Linearity}) and (E8)-(\textbf{cMON}) §} \\
&\quad \sum_{j=l}^{\infty} \mathbb{E}\left[\frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right] \\
&= \text{§ By enumerating all possible stacks for } \widehat{\beta}_k, \widehat{\beta}_{k+1} \\
&\quad \text{and values for } Cost(\widehat{\beta}_k) \text{§} \\
&\quad \sum_{j=l}^{\infty} \mathbb{E}\left[\sum_{\beta,\beta',c} \mathbf{1}_{(\widehat{\beta}_k, \widehat{\beta}_{k+1}, Cost(\widehat{\beta}_k)) = (\beta,\beta',c)} \cdot \frac{\Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \,\middle|\, \mathcal{G}_k\right] \\
&= \text{§ By applying (E6)-(\textbf{Linearity}) and (E8)-(\textbf{cMON}) §} \\
&\quad \sum_{j=l}^{\infty} \sum_{\beta} \mathbb{E}\left[\sum_{\beta',c} \mathbf{1}_{(\widehat{\beta}_k, \widehat{\beta}_{k+1}, Cost(\widehat{\beta}_k)) = (\beta,\beta',c)} \cdot \frac{\Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \,\middle|\, \mathcal{G}_k\right]
\end{aligned}$$

Consider any event $A \in \mathcal{G}_k$ such that A is the disjoint union of d cylinder sets defined by d distinct finite prefixes $\alpha_1, \ldots, \alpha_d$ for which each α_j $(1 \le j \le d)$ is a finite sequence of stacks with length k. We use $\mathbf{P}(\beta, (\beta', c))$ to denote the probability that (the next stack, the cost triggered) is equal to (β', c) given the current stack β. By the definition of integration, we have that

$$\begin{aligned}
& \int_A \sum_{\beta', c} \mathbf{1}_{(\widehat{\beta}_k, \widehat{\beta}_{k+1}, Cost(\widehat{\beta}_k)) = (\beta, \beta', c)} \cdot \frac{\Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \mathrm{dPr}_{\beta^*} \\
= & \sum_{j=1}^{d} \mathbf{1}_{\alpha_j \downarrow = \beta} \cdot \Pr_{\beta^*}\left(Cyl(\alpha_j)\right) \cdot \left[\sum_{\beta', c} \frac{\mathbf{P}(\beta, (\beta', c)) \cdot \Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \right] \\
= & \int_A \mathbf{1}_{\widehat{\beta}_k = \beta} \cdot \left[\sum_{\beta', c} \frac{\mathbf{P}(\beta, (\beta', c)) \cdot \Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \right] \mathrm{dPr}_{\beta^*}
\end{aligned}$$

where $Cyl(\alpha_j)$ is the cylinder set with the common prefix α_j and $\alpha_j\downarrow$ denotes the last stack of α_j. Thus by the definition of conditional expectation, we have that

$$\mathbb{E}[\sum_{\beta', c} \mathbf{1}_{(\widehat{\beta}_k, \widehat{\beta}_{k+1}, Cost(\widehat{\beta}_k)) = (\beta, \beta', c)} \cdot \frac{\Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \mid \mathcal{G}_k] = \mathbf{1}_{\widehat{\beta}_k = \beta} \cdot \left[\sum_{\beta', c} \frac{\mathbf{P}(\beta, (\beta', c)) \cdot \Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \right] \text{(a.s.)}.$$

It follows that

$$\begin{aligned}
& \mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k \right] \\
= & \sum_{j=l}^{\infty} \sum_{\beta} \mathbb{E}\left[\sum_{\beta', c} \mathbf{1}_{(\widehat{\beta}_k, \widehat{\beta}_{k+1}, Cost(\widehat{\beta}_k)) = (\beta, \beta', c)} \cdot \frac{\Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \,\middle|\, \mathcal{G}_k \right] \\
= & \sum_{j=l}^{\infty} \sum_{\beta} \mathbf{1}_{\widehat{\beta}_k = \beta} \cdot \left[\sum_{\beta', c} \frac{\mathbf{P}(\beta, (\beta', c)) \cdot \Psi_{i+j}(\beta')}{(j-l)!} \cdot c^{j-l} \right] \\
= & \sum_{j=l}^{\infty} \sum_{\beta} \mathbf{1}_{\widehat{\beta}_k = \beta} \cdot \mathbb{E}\left[\frac{\Psi_{i+j}(next_{\beta}) \cdot Cost(\beta)^{j-l}}{(j-l)!} \right] \\
= & \sum_{j=l}^{\infty} \mathbb{E}\left[\frac{\Psi_{i+j}(next_{\widehat{\beta}_k}) \cdot Cost(\widehat{\beta}_k)^{j-l}}{(j-l)!} \right] \\
= & \mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(next_{\widehat{\beta}_k}) \cdot Cost(\widehat{\beta}_k)^{j-l}}{(j-l)!} \right] \\
\le & \Psi_{i+l}(\widehat{\beta}_k) \text{ (a.s.)}.
\end{aligned}$$

Thus we have

$$\mathbb{E}[Y_{k+1,i} \mid \mathcal{G}_k] = \sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \mathbb{E}\left[\sum_{j=l}^{\infty} \frac{\Psi_{i+j}(\widehat{\beta}_{k+1})}{(j-l)!} \cdot Cost(\widehat{\beta}_k)^{j-l} \,\middle|\, \mathcal{G}_k\right]$$
$$\leq \sum_{l=0}^{\infty} \frac{C_k^l}{l!} \cdot \Psi_{i+l}(\widehat{\beta}_k) = Y_{k,i}$$

Then by Optional Stopping Theorem (Theorem 3.3), we have that $\mathbb{E}[Y_{\tau,i}] \leq \mathbb{E}[Y_{0,i}]$ for every $i \geq 0$. Finally, the theorem follows from the fact that $Y_{\tau,i} \geq f^{(i)}(C_\tau)$ (see property (b) above) and $Y_{0,i} = \Psi_i(\beta_0)$. □

D.2 Full Proof of Theorem 3.5

Proof. Since all $f_1, \ldots, f_m$ are non-negative, we have that $\Psi_i(\epsilon) \geq t_\star^i \geq f^{(i)}(0)$, thus (A1) is satisfied. We verify (A2) by the definition of $next_\beta$. If $n_1 \leq c_{r_1}$, then $next_\beta$ is deterministically equal to $(r_2, n_2) \cdots (r_k, n_k)$, and $Cost(\beta) = 0$, thus for every $i \geq 0$, we have

$$\Psi_i(\beta) = t_\star^i \cdot \exp\left(\left(t_\star \cdot \textstyle\sum_{j=1}^k f_{r_i}(n_i)\right)\right) \geq t_\star^i \cdot \exp\left(\left(t_\star \cdot \textstyle\sum_{j=2}^k f_{r_i}(n_i)\right)\right) = \mathbb{E}\left[\textstyle\sum_{j\geq 0} \frac{\Psi_{k+j}(next_\beta)}{j!} \cdot Cost(\beta)^j\right].$$

Otherwise (i.e., $n_1 > c_{r_1}$), the random stack $next_\beta$ consists of $(r_1, h_{r_1,1}(n_1)), \ldots, (r_1, h_{r_1,S_{r_1}}(n_1))$ and $(q_{r_1,1}, g_{r_1,1}(n_1)), \ldots, (q_{r_1,O_{r_1}}, g_{r_1,O_{r_1}}(n_1))$, followed by $(r_2, n_2) \ldots (r_k, n_k)$. Note that $Cost(\beta) = Cost_{r_1}(n_1)$. Thus, for every $i \geq 0$, we have:

$$\begin{aligned}
&\Psi_i(\beta) = t_\star^i \cdot \exp(t \cdot \textstyle\sum_{i=1}^k f_{r_i}(n_i)) = t_\star^i \cdot \exp(t_\star \cdot \sum_{i=2}^k f_{r_i}(n_i)) \cdot \exp(t_\star \cdot f_{r_1}(n_1)) \\
&\geq t_\star^i \cdot \exp(t_\star \cdot \textstyle\sum_{i=2}^k f_{r_i}(n_i)) \cdot \mathbb{E}\left[\exp\left(t_\star \cdot \left(Cost_{r_1}(n_1) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n_1)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n_1))\right)\right)\right] \\
&= \mathbb{E}\left[\textstyle\sum_{j\geq 0} t_\star^{i+j}/j! \cdot Cost_{r_1}(n_1)^j \cdot \exp(t_\star \cdot (\sum_{i=2}^k f_{r_i}(n_i) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n_1)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n_1))))\right] \\
&= \mathbb{E}\left[\textstyle\sum_{j\geq 0} \frac{\Psi_{i+j}(next_\beta)}{j!} \cdot Cost(\beta)^j\right]
\end{aligned}$$

where the third inequality holds by $t_\star \leq t_{r_1}(n_1)$ and the convexity of $\exp(\cdot)$. In detail, we define $X_r(n) \coloneqq Cost_r(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n)) - f_r(n)$ to be the exponent of (‡) after dividing the both sides by the left hand side. Then (‡) could be reformulated as $\mathbb{E}[\exp(t_r(n) \cdot X_r(n))] \leq 1$. Note that $e(t) \coloneqq \mathbb{E}[\exp(t \cdot X_r(n))]$ is a convex function of t, and $e(0) = 1$, thus, if $e(t_r(n)) \leq 1$, for any $t_\star \leq t_r(n)$, $e(t_\star) \leq 1$. Thus the lemma follows. □

D.3 Full Proof of Theorem 3.6

Proof. We firstly verify (A1). Since $\theta^{(i)}(0) = k!$ if and only if $i = k$, otherwise $\theta^{(i)}(0) = 0$. For $i \neq k$, the (A1) is satisfied by non-negativity of f's, for $i = k$, (A1) holds by (4). We then verify (A2), for $i \geq k+1$, since $\Psi_i(\beta) = 0$ for any stack β, the conditions naturally holds. For $0 \leq i \leq k$, consider $\beta \coloneqq (r_1, n_1) \ldots (r_t, n_t)$, if $n_1 \leq c_{r_1}$ then $next_\beta$ deterministically equal to $(r_2, n_2) \ldots (r_t, n_t)$ and $Cost(\beta) = 0$, hence, (A2) holds since $\Psi_i(\beta) \geq \Psi_i(next_\beta)$, otherwise, $next_\beta = (r_2, n_2)(r_3, n_3) \cdots (r_k, n_k)(r_1, h_{r_1,1}(n_1)) \cdots (r_1, h_{r_1,S_{r_1}}(n_1))$
$(q_{r_1,1}, g_{r_1,1}(n_1)) \cdots (q_{r_1,O_{r_1}}, g_{r_1,O_{r_1}}(n_1))$ and $Cost(\beta) = Cost_{r_1}(n_1)$. Hence, we have that:

$$\begin{aligned}
&\Psi_i(\beta) = k! \cdot \textstyle\sum_{0\leq p_1,\cdots,p_t\leq k-i, \sum_{u=1}^t p_u = k-i} \prod_{u=1}^t \frac{f_{r_u}^{(p_u)}(n_u)}{p_u!} \\
&= k! \cdot \textstyle\sum_{p_1=0}^{k-i} \frac{f_{r_1}^{(p_1)}(n_1)}{p_1!} \cdot \sum_{\star''} \prod_{u=2}^t \frac{f_{r_u}^{(p_u)}(n_u)}{p_u!} \\
&\geq k! \cdot \textstyle\sum_{p_1=0}^{k-i} \left(\sum_{\mathcal{I}'} \frac{Cost_r(n)^j}{j!} \cdot \prod_{u=1}^{S_r} \frac{f_r^{(p'_u)}(h_{r,u}(n))}{p'_u!} \cdot \prod_{u=1}^{O_r} \frac{f_{q_{r,u}}^{(e'_u)}(g_{r,u}(n))}{e'_u!}\right) \cdot \sum_{\mathcal{I}''} \prod_{u=2}^t \frac{f_{r_u}^{(p_u)}(n_u)}{p_u!} \\
&= \mathbb{E}\left[\textstyle\sum_{j=0}^{k-i} \frac{Cost_{r_1}(n_1)^j}{j!} \cdot \Psi_{i+j}(next_\beta)\right] = \mathbb{E}\left[\textstyle\sum_{j=0}^{k-i} \frac{\Psi_{i+j}(next_\beta)}{j!} \cdot Cost(\beta)^j\right]
\end{aligned}$$

where s

$$\mathcal{I}' := \left[0 \le j, p'_1, \ldots, p'_{S_r}, e'_1, \ldots, e'_{O_r} \le i, \sum_{u=1}^{S_r} p'_u + \sum_{u=1}^{O_r} e'_u + j = p_1\right]$$
$$\mathcal{I}'' := \left[0 \le p_2, \cdots, p_t \le k - i - p_1, \sum_{u=2}^{t} p_u = k - i - p_1\right]$$

Thus, (A2) holds. □

E DETALED DESCRIPTION OF ALGMOMENT(L_T, A, B, N): APPROXIMATING HIGH-ORDER MOMENTS WITH PSEUDO-POLYNOMIALS

In the verification procedure of our algorithm, to approximate the high-order moments of $f(i)$, where f is a pseudo-monomial and i observes as an Uniform() or MUniform() distribution, we need to tightly approximate the summation $\sum_{i=0}^{n-1} i^a \cdot \ln^b i$ with pseudo-polynomials. In this part, we devise an algorithm AlgMoment(L_T, a, b, n) to solve this problem through integrals and taylor expansions, where a, b represents the degree for i and $\ln i$, and L_T represents the degree of approximation. i.e. our algorithm will approximate

$$\sum_{i=0}^{n-1} i^a \cdot \ln^b i \tag{16}$$

with approximation degree L_T.

The goal of this algorithm is to synthesize pseudo-polynomial upper and lower bounds for (18) with additive constant error. To briefly introduce the algorithm, to approximate (18), our algorithm naturally chooses to use its corresponding integral $\int_d^n x^a \cdot \ln^b x \, \mathrm{d}x$, since the latter could be automatically computed as a pseudo-polynomial through integration by parts. Then our algorithm would measure the difference between the summation and integral through Taylor's expansion in each term of summation with degree $a + L_T$, and controls the additive error for minor terms by Riemann-Zeta functions. To avoid corner cases, our algorithm assumes that $n \ge \lceil \exp(b/L_T) \rceil + 1$. Define $d := \lceil \exp(b/L_T) \rceil$. It firstly splits the summation into two parts:

$$\sum_{i=0}^{n-1} i^a \cdot \ln^b = \sum_{i=0}^{d-1} i^a \cdot \ln^b i + \sum_{i=d}^{n-1} i^a \cdot \ln^b i \tag{17}$$

where the first part could be calculated trivially. We now focus on how to approximate the second term.

$$\sum_{i=d}^{n-1} i^a \cdot \ln^b i \tag{18}$$

We will firstly introduce mathematical backgrounds of this algorithm, and then give an overview of this algorithm, and finally provide algorithmic details.

Background. Below we present mathematical backgrounds that our algorithm heavily relies on.

(Integration of $i^a \cdot ln^b i$.) There is an algorithm that given a, b, d, where a, b represents the degree of i and $\ln i$ and d is the lower bound in integration, and computes the integral $\int_d^n x^a \cdot \ln^b x \mathrm{d}x$ with pseudo-polynoimal. This directly follows from integration by parts.

(Derivatives of $i^a \cdot \ln^b i$.) There is an algorithm that given a, b, k and symbolically expands the k-th derivative of $i^a \cdot \ln^b i$, i.e. $(i^a \cdot \ln^b i)^{(k)}$ with pseudo-polynoimal. Below, we will use $\beta_{a,b,k,j}$ to represent the coefficient of $i^{a-k} \cdot \ln^j i$ in the k-th derivative of $i^a \cdot \ln^b i$. Formally, we have that:

$$(i^a \cdot \ln^b i)^{(k)} = \sum_{j=0}^{b} \beta_{a,b,k,j} \cdot i^{a-k} \cdot \ln^j i \tag{19}$$

This follows straightforwardly by the multiplication rule for derivatives.

(Monotonicity of $i^a \cdot \ln^b i$ for $a \le -1$ and $b \ge 0$.) . For every $a \le -1$ and $b \ge 0$, $i^a \cdot \ln^b i$ is monotonically increasing in $(0, \exp(-b/a)]$ and monotonically decreasing in $[\exp(-b/a), +\infty)$. It follows from the derivative fo $i^a \cdot \ln^b i$.

(Constant Approximation of $\sum_{i=d}^{n-1} i^a \cdot \ln^b i$ for $a \le -2$.) For every $a \le -2$ and $b \ge 0$, we could approximate the summation through b-th derivative of Riemman-Zeta function at $x = -a$, i.e. $\zeta^{(b)}(-a)$. Formally, we have that

$$0 \le \sum_{i=d}^{n-1} i^a \cdot \ln^b i \le \zeta^{(b)}(-a) - \sum_{i=0}^{d-1} i^{-a} \cdot \ln^b i. \tag{20}$$

This directly follows from the identity below:

$$\zeta^{(b)}(-a) = \sum_{i=0}^{\infty} i^a \cdot \ln^b i \ (a, b \in \mathbb{R}, a < -1)$$

Details. Below we will introduce details of AlgMoment. To approximate (18), Our algorithm naturally chooses to use its corresponding integral $\int_d^n x^a \cdot \ln^b x \, \mathrm{d}x$, since by Fact 1, the latter could be automatically computed as a pseudo-polynomial through integration by parts. The rest of our algorithm will carefully measure the difference between these two terms. To achieve this, our algorithm will do the following 3 steps.

Step 1. (Subtraction and Reorganization). Our algorithm will firstly subtract one from the other, and by splitting the integral into n parts and reorganizing the terms appropriately, we derive the formula below:

$$\int_d^n x^a \cdot \ln^b x \, \mathrm{d}x - \sum_{i=d}^{n-1} i^a \cdot \ln^b i \tag{21}$$

$$= \left(\sum_{i=d}^{n-1} \int_i^{i+1} x^a \cdot \ln^b x \, \mathrm{d}x\right) - \sum_{i=d}^{n-1} i^a \cdot \ln^b i \tag{22}$$

$$= \sum_{i=d}^{n-1} \int_0^1 \left((i+x)^a \ln^b(i+x) - i^a \ln^b i\right) \, \mathrm{d}x \tag{23}$$

Step 2. (Approximating the Integration). Then, we will focus on (23) and try to bound this term from below and above by pseudo-polynomials. In general, our algorithm will expand the term $(i+x)^a \ln^b(i+x) - i^a \ln^b i$ within the integral and and do substitutions. This is achieved through two substeps.

Step 2(a). (Expanding through Taylor's Expansion). Firstly, our algorithm expands this term through Taylor's Expansion with degree $a + L_T - 1$ and Lagrange's Remainder:

$$(i+x)^a \ln^b(i+x) - i^a \ln^b i = \sum_{k=1}^{a+L_T-1} \frac{(t^a \ln^b t)^{(k)}|_{t=i}}{k!} \cdot x^k + \frac{(t^a \ln^b t)^{(a+L_T)}|_{t=\xi_{x,i}}}{(a+L_T)!} \cdot x^{a+L_T} \tag{24}$$

where for each $\xi_{i,x}$ is bounded in $[0,1]$, i.e. $0 \le \xi_{i,x} \le 1$.

Step 2(b). (Substitution). In the following, our algorithm would do a chain of substitutions. Firstly, by plugging in (19), we derive an equivalent form of (24):

$$\sum_{k=1}^{a+L_T-1} \sum_{j=0}^{b} \frac{\beta_{a,b,k,j}}{k!} \cdot i^{a-k} \cdot \ln^j i \cdot x^k + \frac{\beta_{a,b,a+L_T,j}}{(a+L_T)!} \cdot \left(i + \xi_{i,x}\right)^{-L_T} \cdot \ln^b\left(i + \xi_{i,x}\right) \cdot x^{a+L_T} \tag{25}$$

Third, by plugging in (25) into (23) and changing the order of summation, we derive that:

$$\int_0^1 \left(\sum_{k=1}^{a+L_T-1} \sum_{j=0}^{b} \frac{\beta_{a,b,k,j}}{k!} \cdot \left(\sum_{i=d}^{n-1} i^{a-k} \cdot \ln^j i \right) \cdot x^k + \frac{\beta_{a,b,a+L_T,j}}{(a+L_T)!} \cdot \left(\sum_{i=d}^{n-1} \left(i+\xi_{i,x}\right)^{-L_T} \cdot \ln^b\left(i+\xi_{i,x}\right) \right) \cdot x^{a+L_T} \right) \mathrm{d}x$$

Finally, by the identity that $\int_0^1 x^k \mathrm{d}x = 1/(k+1)$. We derive that:

$$\sum_{k=1}^{a+L_T-1} \sum_{j=0}^{b} \frac{\beta_{a,b,k,j}}{(k+1)!} \cdot \underbrace{\left(\sum_{i=d}^{n-1} i^{a-k} \cdot \ln^j i\right)}_{SA} + \underbrace{\sum_{j=0}^{b} \frac{\beta_{a,b,a+L_T,j}}{(a+L_T+1)!} \cdot \left(\sum_{i=d}^{n-1} \left(i+\xi_{i,x}\right)^{-L_T} \cdot \ln^j\left(i+\xi_{i,x}\right)\right)}_{SB}$$

Step 3. (Bounding through Subinstances and Riemaman-Zeta Function). We will now focus on the equality above, note that in this formula, $\beta_{a,b,k,j}$ are constants, thus, it suffices to separately bound SA and SB from below and above by pseudo-polynomials.

Bounding *SA*. Note that bounding *SA* could be directly solved through the subinstace AlgMoment$(L_T, d, a-k, j)$.

Bounding *SB*. Bounding *SB* is more complicated, it is solved through three substeps. Below we will illustrate how to bound *SB* from above, the procedure of getting *SB*'s lower bound is similar.

- Firstly, we replace $\beta_{a,b,a+L_T,j}$ with $\max\{\beta_{a,b,a+L_T,j}, 0\}$, then it suffices to overapproximate the summation $\sum_{i=d}^{n-1} \left(i+\xi_{i,x}\right)^{-L_T} \cdot \ln^j\left(i+\xi_{i,x}\right)$.
- Then, by Fact 3 and our assumption that $d \geq \exp(b/L_T)$, we have that $\left(i+\xi_{i,x}\right)^{-L_T} \cdot \ln^j\left(i+\xi_{i,x}\right) \leq i^{-L_T} \cdot \ln^b i$.
- Finally, from Fact 4, we have that $\sum_{i=d}^{n-1} \left(i+\xi_{i,x}\right)^{-L_T} \cdot \ln^j\left(i+\xi_{i,x}\right) \leq \sum_{i=d}^{n-1} i^{-L_T} \cdot \ln^b i \leq \zeta^{(b)}(L_T) - \sum_{i=0}^{d-1} i^{-L_T} \cdot \ln^b i$. In summary, we could bound *SB* from above by:

$$\sum_{j=0}^{b} \frac{\max\{\beta_{a,b,a+L_T,j}, 0\}}{(a+L_T+1)!} \cdot \left(\zeta^{(b)}(L_T) - \sum_{i=0}^{d-1} i^{-L_T} \cdot \ln^b i\right)$$

and *SB* could also be similarly bounded from below by:

$$\sum_{j=0}^{b} \frac{\min\{\beta_{a,b,a+L_T,j}, 0\}}{(a+L_T+1)!} \cdot \left(\zeta^{(b)}(L_T) - \sum_{i=0}^{d-1} i^{-L_T} \cdot \ln^b i\right)$$

By further plugging in the corresponding pseudo-polynomial for the integral $\int_d^n x^a \cdot \ln^b x \mathrm{d}x$, our algorithm could successfully derive a tight approximation of (18) with additive constant error.

F DETAILED DESCRIPTION OF ALGCHECK(P, N_0): CHECKING THE NON-POSITIVITY OF PSEUDO-POLYNOMIAL P

In the verification procedure of our algorithm, we need to check whether the given pseudo-polynomial P is non-positive for every $n \geq n_0$. Below we fix a univariate pseudo-polynomial $P(n) \coloneqq \sum_{a,b\in\mathbb{Z}} c_{a,b} \cdot n^a \cdot \ln^b n$. To briefly introduce our algorithm, it firstly obtains an equivalent form $P_1(n)$ by extracting the term with largest order and dividing the polynomial with the largest terms. Then, our algorithm follows the naive idea that try to enumerate every $n \geq n_0$ and evaluate $P_1(n)$. However, the enumeration involves infinite steps. Thus, we propose a condition that decides the termination of enumeration. The condition considers an over-approximation $P_2(n)$ of $P_1(n)$ by removing all non-constant terms with negative coefficients, and a constant n_e that serves as an underapproximation of monotonically decreasing interval of $P_2(n)$. i.e., $P_2(n)$ will monotonically decreasing for every $n \geq n_e$. Intuitively, the condition states that if $P_2(n_1) \leq 0$ for some $n_1 \geq n_e$, then $P_2(n) \leq 0$ for every $n \geq n_1$, combining with the fact that $P_2(n)$ is an over-approximation of $P_1(n)$, we could conclude that $P_1(n)$ is also non-positive for every $n \geq n_1$, thus we could safely terminate the enumeration. This algorithm also applies to check non-positivity only for odd/even n's, by simply enumerating only odd/even numbers $n \geq n_0$ rather than every $n \geq n_0$. Please also note that our algorithm is complete, in the sense that if $P_1(n)$ is always non-positive, the algorithm will eventually terminate.

Our algorithm relies on the following monotonicity property of $i^a \cdot \ln^b i$.

Monotonicity of $i^a \cdot \ln^b i$. For every $a \leq -1$ and $b \geq 0$, $i^a \cdot \ln^b i$ is monotonically increasing in $(0, \exp(-b/a)]$ and monotonically decreasing in $[\exp(-b/a), +\infty)$. This follows directly through computing the derivative.

In detail, the algorithm is as the follows.

Step 1. (Extracting and Dividing the Major Term). Our algorithm firstly finds the term with largest order. In detail, our algorithm will scan each term in the polynomial and finds the term with

highest degree of n, if there are several terms with the same highest degree of n, our algorithm chooses the term among them with the highest degree of $\ln n$. Formally, our algorithm finds the term $n^{a^*} \cdot \ln^{b^*} n$ with $c_{a^*,b^*} \neq 0$ and highest a^*, if there are more than one terms such that their degree of n are all equal to a^*, we choose the largest b^* among them. Then, our algorithm divides each term in $P(n)$ by $n^{a^*} \cdot \ln^{b^*} n$, and derives the pseudo-polynomial $P_1(n)$ below:

$$P_1(n) = c_{a^*,b^*} + \sum_{a,b \neq a^*,b^*} c_{a,b} \cdot n^{a-a^*} \cdot \ln^{b-b^*} n$$

Note that by our choice of a^*, b^*, the degree of n and $\ln n$ in each non-constant term in $P_1(n)$, i.e., $a - a^*, b - b^*$ either satisfies $a - a^* \leq -1$, or satisfies $a - a^* = 0$ and $b - b^* \leq -1$, also deciding non-positivity of $P(n)$ is equivalent to deciding non-positivity of $P_1(n)$.

Step 2. (Bounded Checking). In general, to check whether $P_1(n)$ is always non-positive for every $n \geq n_0$, our algorithm follows the naive idea that try to enumerate every $n \geq n_0$ and evaluate $P_1(n)$. However, the enumeration involves infinite steps. Thus, we propose a condition (Proposition F.2) that decides the termination of enumeration. The condition considers an over-approximation $P_2(n)$ of $P_1(n)$ by removing all non-constant terms with negative coefficients, and a constant n_e that serves as an underapproximation of monotonically decreasing interval of $P_2(n)$. i.e., $P_2(n)$ will monotonically decreasing for every $n \geq n_e$. Intuitively, the condition states that if $P_2(n_1) \leq 0$ for some $n_1 \geq n_e$, then $P_2(n) \leq 0$ for every $n \geq n_1$, combining with the fact that $P_2(n)$ is an over-approximation of $P_1(n)$, we could conclude that $P_1(n)$ is also non-positive for every $n \geq n_1$, thus we could safely terminate the enumeration. Please also note that our algorithm is complete, in the sense that if $P_1(n)$ is always non-positive, the algorithm will eventually terminate.

We will firstly introduce two propositions deciding the termination of enumeration.

Proposition F.1. *Let $P_2(n)$ to be an over-approximation of $P_1(n)$ by removing all non-constant terms with negative coefficients.i.e.*

$$P_2(n) \coloneqq c_{a^*,b^*} + \sum_{a,b \neq a^*,b^*} \max\{0, c_{a,b}\} \cdot n^{a-a^*} \cdot \ln^{b-b^*} n$$

Let $n_e \coloneqq \lceil \max\{\exp(-(b-b^*)/(a-a^*)) \mid (a,b) \neq (a^*,b^*) \text{ and } c_{a,b} \neq 0\} \rceil$ be a constant. Then we have that $P_2(n)$ monotonically decreases for every $n \geq n_e$.

Proof. Note that every coefficient for non-constant terms in $P_2(n)$ is non-negative. Thus the proposition directly follows from the monotonicity of the function $i^a \cdot \ln^b i$. □

Proposition F.2. *If $P_2(n_1) \leq 0$ for some $n_1 \geq n_e$, then $P_2(n_1) \leq 0$ for every $n_1 \geq n_e$.*

Proof. The proposition directly follows from the monotonicity of $P_2(n)$. □

In detail, our algorithm follows the 3 substeps below.

Step 2(a). (Checking Constants). If $c_{a^*,b^*} > 0$, then we could directly conclude that $P_1(n) > 0$ for some sufficiently large n. This is due to the fact that the degree of n and $\ln n$ in each non-constant term in $P_1(n)$, i.e., $a - a^*, b - b^*$ either satisfies $a - a^* \leq -1$, or satisfies $a - a^* = 0$ and $b - b^* \leq -1$, thus is negligible for when n is sufficiently large.

Step 2(b). (Calculating the lower bound n_e). Following Proposition F.1, we calculate the n_e by scanning every element in $P_2(n)$, and trivially evaluate $P_1(n)$ for every $n_0 \leq n \leq n_e - 1$, if $P_1(n) > 0$ for some $1 \leq n \leq n_e$, then we could conclude that $P_1(n) > 0$ for some $n \geq n_0$.

Step 2(c). (Enumeration from n_e). Our algorithm then tries every $n \geq n_e$, and do the following judgements:

- If $P_1(n) > 0$, then we could conclude that $P_1(n) > 0$ for some $n \geq n_0$ and terminates the enumeration.

- If $P_2(n) \leq 0$, then through Proposition F.2, we could conclude that $P_2(n) \leq 0$ for every $n \geq n_0$, combining with the fact that $P_2(n)$ is an over-approximation of $P_1(n)$, we could conclude that $P_1(n)$ is also non-positive for every $n \geq n_1$, and terminates the enumeration.

The soundness is straightforward. The completeness of our algorithm is stated below:

Theorem F.3 (Soundness). *If $P_1(n) \leq 0$ for every $n \geq n_0$, then our algorithm will terminates.*

Proof. We just need to check whether step 2(c) will eventually terminate. Since we pass the check at step 2(a), we have that $c_{a^*,b^*} < 0$. Then due to the fact that the degree of n and $\ln n$ in each non-constant term in $P_1(n)$, i.e., $a - a^*, b - b^*$ either satisfies $a - a^* \leq -1$, or satisfies $a - a^* = 0$ and $b - b^* \leq -1$, thus is negligible for when n is sufficiently large. Thus $P_2(n)$ will be negative for some sufficiently large n, thus the algorithm will terminate in finite steps. □

G PROOF OF THE PROPOSITION THAT APPROXIMATES LOGARITHM

Proof of Proposition 5.1:

Proof. We firstly prove the case for $b > 0$. The lower bound is trivial since $b > 0$. For the upper bound, we have that:

$$\begin{aligned}\ln(a \cdot x + b) - \ln(a \cdot x) &= \ln\left(1 + \frac{b}{a \cdot x}\right) \\ &\leq \frac{b}{a \cdot x}\end{aligned}$$

For the case $b < 0$, the upper bound is also derive through the same procedure as the case $b > 0$. For the lower bound, first note that $\ln(a \cdot x) = \ln(a \cdot x + b - b)$, and:

$$\begin{aligned}\ln((a \cdot x + b) - b) - \ln(a \cdot x + b) &= \ln\left(1 - \frac{b}{a \cdot x + b}\right) \\ &\leq -\frac{b}{a \cdot x + b}\end{aligned}$$

Thus we have that:

$$\ln(a \cdot x + b) \geq \ln a + \ln x + \frac{b}{a \cdot x + b}$$

Since we assume that $x \geq \frac{-2 \cdot b}{a}$, we have that $\frac{2 \cdot b}{a \cdot x} < \frac{b}{a \cdot x + b}$, thus we have that:

$$\ln(a \cdot x + b) \geq \ln a + \ln x + \frac{2 \cdot b}{2 \cdot x}$$

□

H A METHOD THAT AUTOMATICALLY CALCULATING THE BOUND $B_R(N)$

In this part, we propose an algorithm that automatically calculates $B_r(n)$ (the "Setting up Template" paragraph in Section 5.3). We will firstly propose a simplified version of our algortihm under restricted settings, and then incrementally add features to support the whole setting for our algorithm.

Firstly recall that

$$X_r(n) = Cost_r(n) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,j}}(g_{r,j}(n)) - f_r(n)$$

The goal of this algorithm is to synthesize a pseudo-monomial $B_r(n)$ such that $-B_r(n) \leq X_r(n) \leq B_r(n)$ for every $n \geq n_0$.

The organization of this section is as follows. We firstly briefly introduce the mathematical background for this algorithm, and then gives a simplified algorithm for a restricted setting. Finally we incrementally add features to this algorithm.

Mathematical Backgrounds. Here we present the mathematical backgrounds that our algorithm relies on.

The first is a proposition that approximates $\ln(a \cdot x + b)$ with $a \geq 0$ for positive x, through pseudo-polynomials from above and below. To achieve this, first note that the case where $a = 0$ or $b = 0$ is trivial. We will next show how to solve case for $a > 0$ and $b \neq 0$, the following proposition splits the case where $b > 0$ or $b < 0$.

Proposition H.1. *For every $a > 0, b > 0$, and $x > 0$, we have that:*

$$\ln a + \ln x \leq \ln(a \cdot x + b) \leq \ln a + \ln x + \frac{b}{a \cdot x}$$

Also, for every $a > 0, b < 0$, and $x \geq \frac{-2 \cdot b}{a}$, we have that:

$$\ln a + \ln x + \frac{2 \cdot b}{a \cdot x} \leq \ln(a \cdot x + b) \leq \ln a + \ln x + \frac{b}{a \cdot x}$$

Proof. We firstly prove the case for $b > 0$. The lower bound is trivial since $b > 0$. For the upper bound, we have that:

$$\begin{aligned} \ln(a \cdot x + b) - \ln(a \cdot x) &= \ln\left(1 + \frac{b}{a \cdot x}\right) \\ &\leq \frac{b}{a \cdot x} \end{aligned}$$

For the case $b < 0$, the upper bound is also derive through the same procedure as the case $b > 0$. For the lower bound, first note that $\ln(a \cdot x) = \ln(a \cdot x + b - b)$, and:

$$\begin{aligned} \ln((a \cdot x + b) - b) - \ln(a \cdot x + b) &= \ln\left(1 - \frac{b}{a \cdot x + b}\right) \\ &\leq -\frac{b}{a \cdot x + b} \end{aligned}$$

Thus we have that:

$$\ln(a \cdot x + b) \geq \ln a + \ln x + \frac{b}{a \cdot x + b}$$

Since we assume that $x \geq \frac{-2 \cdot b}{a}$, we have that $\frac{2 \cdot b}{a \cdot x} < \frac{b}{a \cdot x + b}$, thus we have that:

$$\ln(a \cdot x + b) \geq \ln a + \ln x + \frac{2 \cdot b}{2 \cdot x}$$

□

Proposition H.2. *For any pseudo-monomial $f(n) := n^u \cdot \ln^v n$ with $u, v \in \mathbb{N}$ and $u \geq 1$, $f(x) + f(n - 1 - x)$ achieves its maximum at $x = 0$ and minimum at $x = \frac{n-1}{2}$, for $0 < x < n - 1$.*

Proof. Computing the derivative of $f(x)$, we have that

$$\begin{aligned} f'(x) &= u \cdot x^{u-1} \ln^v x + v \cdot x^{u-1} \cdot \ln^{v-1} x \\ &= x^{u-1} \cdot \ln^{v-1} x \cdot (u \cdot \ln x + v) \end{aligned}$$

First we consider corner cases, if $u = 0$ and $v = 0$, the proposition obviously holds.

Then if $u \geq 1$ and $v = 0$, we have that $f'(x) + f'(n - 1 - x) = u \cdot (x^{u-1} - (n - 1 - x)^{u-1})$, thus $f'(x) + f'(n - 1 - x) < 0$ for every $0 \leq x \leq (n - 1)/2$, and $f'(x) + f'(n - 1 - x) > 0$ for every $(n - 1)/2 \leq x \leq n - 1$, thus the proposition holds.

If $u \geq 1$ and $v \geq 1$. we have that $f'(x)+f'(n-1-x) = x^{u-1} \cdot \ln^{v-1} x \cdot (u \cdot \ln x + v) - (n-1-x)^{u-1} \cdot \ln^{v-1}(n-1-x) \cdot (u \cdot \ln(n-1-x)+v)$, thus $f'(x)+f'(n-1-x) < 0$ for every $0 \leq x \leq (n-1)/2$, and $f'(x)+f'(n-1-x) > 0$ for every $(n-1)/2 \leq x \leq n-1$, thus the proposition holds.

□

A Simplifed Version of Our Algorithm. At the beginning, we present our algorithm in a restricted setting, we assume that:

- Every random variable in $X_r(n)$ observes as a discrete distribution.
- The support of all discrete distributions does not involve floors and ceils, thus each element in the support is a linear expression $a \cdot n + b$ where $a \in \mathbb{N}$ and $b \in \mathbb{Z}$.
- The number of self-recursion calls is 1, i.e. $S_r = 1$

Our algorithm is as the follows.

Step 1. (Enumerating the Support). Since every random variable observes as a discrete distribution, our algorithm firstly enumerate every possible combination of these random variables. Formally, we enumerate every ω in the sample space and considers:

$$X_r(n)(\omega) = Cost_r(n)(\omega) + \sum_{i=1}^{S_r} f_r(h_{r,i}(n)(\omega)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n)(\omega)) - f_r(n)$$

Below our algorithm will try to bound $X_r(n)(\omega)$ for some fixed ω, by enumeration and simply taking the maximum for all ω's, we could derive the bound $B_r(n)$.

Step 2. (Approximating $X_r(n)(\omega)$ to Pseudo-Polynomial from Above and Below). Note that since each element in the support is a linear expression $a \cdot n + b$ where $a \in \mathbb{N}$ and $b \in \mathbb{Z}$, and recall that $f_r(n)$ is a pseudo-monomial, each term in $X_r(n)(\omega)$ is in the form $\lambda \cdot (a \cdot n + b)^u \cdot \ln^v(c \cdot n + d)$, for real λ, natural numbers a, c, u, v and integers b, d. First note that it is straightforward to expand $(a \cdot n + b)^u$ as pseudo-polynomials through Newton's binomial Theorem. Then, by combining Proposition H.1, our algorithm could approximate $\ln(c \cdot n + d)$ from above and below, and then through Newton's binomial theorem, we could expand the v-th power of the approximate form of $\ln(c \cdot n + d)$ as a pseudo-polynomial. During this procedure, we also records a value ℓ as the maximum value of $\lceil \frac{-2 \cdot d}{c} \rceil$ among all terms with $d < 0$, which represents the range that our proposition for approximation (Proposition H.1) does not apply.

Below we represent the pseudo-polynomial lower and upper bounds for $X_r(n)(\omega)$ as $L_r^\omega(n)$ and $R_r^\omega(n)$. Please note that this bound only applies to $n \geq \ell$.

Step 3. (Getting the Monomial Bound). In this step, we will further overapproximate $-L_r^\omega(n)$ and $R_r^\omega(n)$ as pseudo-monomials, by taking the maximum of these two monomials, we derive the monomial bound $B_r^\omega(n)$ for $X_r(n)(\omega)$ such that $-B_r^\omega(n) \leq X_r(n)(\omega) \leq B_r^\omega(n)$. Below we illustrate the procedure that overapproximating $R_r^\omega(n)$ as pseudo-monomials, the overapproximation for $-L_r^\omega(n)$ exactly follows the same procedure. We represent the pseudo-bound synthesis result for $R_r^\omega(n)$ as $\theta \cdot n^{u_{r,\omega}} \cdot \ln^{v_{r,\omega}} n$.

Below we represent $R_r^\omega(n)$ as $R_r^\omega(n) := \sum_{a,b \in \mathbb{Z}} c_{a,b} \cdot n^a \cdot \ln^b n$. In detail, this is achieved through 3 substeps.

Step 3(a). (Extracting the Major Term). Our algorithm firstly finds the term with largest order. In detail, our algorithm will scan each term in the polynomial and finds the term with highest degree of n, if there are several terms with the same highest degree of n, our algorithm chooses the term among them with the highest degree of $\ln n$. Formally, our algorithm finds the term $n^{a^*} \cdot \ln^{b^*} n$ with $c_{a^*,b^*} \neq 0$ and highest a^*, if there are more than one terms such that their degree of n are all equal to a^*, we choose the largest b^* among them. After finding the major term $n^{a^*} \cdot \ln^{b^*} n$, we could derive that $u_{r,\omega} = a^*$ and $v_{r,\omega} = b^*$.

Step 3(b). (Considering Corner Cases). We enumerate every $n_0 \leq n \leq \ell$, evaluate the coefficient $\frac{X_r(n)(\omega)}{n^{a^*} \cdot \ln^{b^*} n}$, take the maximum among them as the lower bound θ_0 for the coefficient θ.

Step 3(c). (Binary Search). Finnaly, we binary search θ, the verification proceudre of binary search simply calls the procedure $\textsc{AlgCheck}(R_r^\omega(n) - \theta \cdot n^{a^*} \cdot \ln^{b^*} n, \max\{\ell, n_0\})$ to check the non-positivity, thus concluding whether the currenct choice of θ leads to a bound. By taking the maximum of θ and θ_0 to tackle with corner cases, we derive the final monomial bound for $R_r^\omega(n)$.

Below, we will then try to remove these restrictions one by one, thus giving the complete version of our algorithm.

Supporting Floors and Ceils. To support floors and ceils, same as our method for in ExpSyn and PolySyn, we just need to case analysis whether n is even or odd and takes the maximum.

Supporting Uniform(n) and MUniform(n) Distribution. To support $\textsc{Uniform}(n)$ distribution, since we are only interested in the bound, it suffices to consider the case when the value is 0 or $n-1$, thus reducing to previous cases. To support $\textsc{MUniform}(n)$ distribution, first by case analysis whether n is even or odd, we remove the floors in this distribution, and then we still only need to consider two endpoints in this distribuiton.

Supporting the Case $S_r = 2$. In this case, after the processing of $X_r(n)$ in the case $S_r = 2$ of the algorithm ExpSyn, if $h_{r,1}(n)$ observes as discrete distribution, we could directly reduce to previous cases. Otherwise, if $h_{r,1}(n)$ observes as $\textsc{Uniform}(n)$ or $\textsc{MUniform}(n)$, since $f_r(n)$ is an over-approximation of expected running time, and $Cost_r(n) + \sum_{j=1}^{O_r} T_{q_{r,j}}(g_{r,j}(n)) \geq 1$, we have that $f_r(n) = \Omega(n)$, thus the Proposition H.2 applies, in this case, we only need to consider the case when the value is $f(0) + f(n-1)$ or $2 \cdot f(\frac{n-1}{2})$, thus reducing to previous cases.