Automated Tail Bound Analysis for Probabilistic Recurrence Relations

Yican Sun[1], Hongfei Fu[2(✉)], Krishnendu Chatterjee[3], and Amir Kafshdar Goharshady[4]

[1] School of Computer Science, Peking University, Beijing, China
sycpku@pku.edu.cn
[2] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
fuhf@cs.sjtu.edu.cn
[3] Institute of Science and Technology, Klosterneuburg, Austria
krishnendu.chatterjee@ist.ac.at
[4] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, Hong Kong SAR, China
goharshady@cse.ust.hk

Abstract. Probabilistic recurrence relations (PRRs) are a standard formalism for describing the runtime of a randomized algorithm. Given a PRR and a time limit κ, we consider the tail probability $\Pr[T \geq \kappa]$, i.e., the probability that the randomized runtime T of the PRR exceeds κ. Our focus is the formal analysis of tail bounds that aims at finding a tight asymptotic upper bound $u \geq \Pr[T \geq \kappa]$. To address this problem, the classical and most well-known approach is the cookbook method by Karp (JACM 1994), while other approaches are mostly limited to deriving tail bounds of specific PRRs via involved custom analysis.

In this work, we propose a novel approach for deriving the common exponentially-decreasing tail bounds for PRRs whose preprocessing time and random passed sizes observe discrete or (piecewise) uniform distribution and whose recursive call is either a single procedure call or a divide-and-conquer. We first establish a theoretical approach via Markov's inequality, and then instantiate the theoretical approach with a template-based algorithmic approach via a refined treatment of exponentiation. Experimental evaluation shows that our algorithmic approach is capable of deriving tail bounds that are (i) asymptotically tighter than Karp's method, (ii) match the best-known manually-derived asymptotic tail bound for QuickSelect, and (iii) is only slightly worse (with a $\log \log n$ factor) than the manually-proven optimal asymptotic tail bound for QuickSort. Moreover, our algorithmic approach handles all examples (including realistic PRRs such as QuickSort, QuickSelect, DiameterComputation, etc.) in less than 0.1 s, showing that our approach is efficient in practice.

Due to different academic norms, authors in Mainland China are ordered by contribution, whereas authors in Austria and Hong Kong SAR are ordered alphabetically. The code and benchmarks are available at https://github.com/boyvolcano/PRR.

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13966, pp. 16–39, 2023.
https://doi.org/10.1007/978-3-031-37709-9_2

1 Introduction

Probabilistic program verification is a fundamental area in formal verification [3]. It extends the classical (non-probabilistic) program verification by considering randomized computation in a program and hence can be applied to the formal analysis of probabilistic computations such as probabilistic models [14], randomized algorithms [2,9,28,30], etc. In this line of research, verifying the time complexity of probabilistic recurrence relations (PRRs) is an important subject [9,30]. PRRs are a simplified form of recursive probabilistic programs and extend recurrence relations by incorporating randomization such as randomized preprocessing and divide-and-conquer. They are widely used in analyzing the time complexity of randomized algorithms (e.g., QuickSort [16], QuickSelect [17], and DiameterComputation [26, Chapter 9]). Compared with probabilistic programs, PRRs abstract away detailed computational aspects, such as problem-specific divide-and-conquer and data-structure manipulations, and include only key information on the runtime of the underlying randomized algorithm. Hence, PRRs provide a clean model for time-complexity analysis of randomized algorithms and randomized computations in a general sense.

In this work, we focus on the formal analysis of PRRs and consider the fundamental problem of tail bound analysis that aims at bounding the probability that a given PRR does not terminate within a prescribed time limit. In the literature, prominent works on tail bound analysis include the following. First, Karp proposed a classic "cookbook" formula [21] similar to Master Theorem. This method is further improved, extended, and mechanized by follow-up works [5,13,30]. While Karp's method has a clean form and is easy to use and automate, the bounds from the method are known to be not tight (see e.g. [15,25]). Second, the works [25] and resp. [15] performed ad-hoc custom analysis to derive asymptotically tight tail bounds for the PRRs of QuickSort and resp. QuickSelect, respectively. These methods require manual effort and do not have the generality to handle a wide class of PRRs.

From the literature, an algorithmic approach capable of deriving tight tail bounds over a wide class of PRRs is a major unresolved problem. Motivated by this challenge, we have the following contributions to this work:

- Based on Markov's inequality, we propose a novel theoretical approach to derive exponentially-decreasing tail bounds, a common type for many randomized algorithms. We further show that our theoretical approach can always derive an exponentially-decreasing tail bound at least as tight as Karp's method under mild assumptions.
- From our theoretical approach, we propose a template-based algorithmic approach for a wide class of PRRs that have (i) common probability distributions such as (piecewise) uniform distribution and discrete probability distributions and (ii) either a single call or a divide-and-conquer for the form of the recursive call. The technical novelties in our algorithm lie in a refined treatment of the estimation of the exponential term arising from our theoretical approach via integrals, suitable over-approximation, and the monotonicity of the template function.

– Experiments show that our algorithmic approach derives asymptotically tighter tail bounds when compared with Karp's method. Furthermore, the tail bounds derived from our approach match the best-known bound for QuickSelect [15], and are only slightly worse by a $\log \log n$ factor against the optimal manually-derived bound for QuickSort [25]. Moreover, our algorithm synthesizes each of these tail bounds in less than 0.1 s and is efficient in practice.

A limitation of our approach is that we do not consider the transformation from a realistic implementation of a randomized algorithm into its PRR representation. However, such a transformation would require examining a diversified number of randomization patterns (e.g., randomized divide-and-conquer) in randomized algorithms and thus is an orthogonal direction. In this work, we focus on the tail bound analysis and present a novel approach to address this problem. Due to space limitations, we relegate some details in the extended version [29].

2 Preliminaries

Below we present necessary background in probability theory and the tail bound analysis problem we consider.

A *probability space* is a triple $(\Omega, \mathcal{F}, \Pr)$ such that Ω is a non-empty set termed as 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 complement and countable union), and $\Pr(\cdot)$ is a *probability measure* on $\mathcal{F}$ (i.e., a function $\mathcal{F} \to [0,1]$ such that $\Pr(\Omega) = 1$ and for every pairwise disjoint set-sequence $A_1, A_2, \ldots$ in $\mathcal{F}$, we have that $\sum_{i \geq 1} \Pr(A_i) = \Pr\left(\bigcup_{i \geq 1} A_i\right)$.

A *random variable* X from a probability space $(\Omega, \mathcal{F}, \Pr)$ is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R}$, i.e., for every $d \in \mathbb{R}$, we have that $\{\omega \in \Omega \mid X(\omega) < d\} \in \mathcal{F}$. We denote $\mathbb{E}[X]$ as its expected value; formally, we have $\mathbb{E}[X] := \int X \, \mathrm{dPr}$. A *discrete probability distribution* (DPD) 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 the DPD is defined as $\mathrm{supp}(\eta) := \{u \in U \mid \eta(u) > 0\}$. We abbreviate finite-support DPD as FSDPD.

A *filtration* of probability space $(\Omega, \mathcal{F}, \Pr)$ is an infinite sequence of $\{\mathcal{F}_n\}_{n \geq 0}$ of σ-algebra over Ω such that $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$ for every $n \geq 0$. Intuitively, it models the information at the n-th step. A *discrete-time stochastic process* is an infinite sequence $\Gamma = \{X_n\}_{n \geq 0}$ of random variables from the probability space $(\Omega, \mathcal{F}, \Pr)$. The process Γ is *adapted* to a filtration $\{\mathcal{F}_n\}_{n \geq 0}$ if for all $n \geq 0$, X_n is $\mathcal{F}_n$-measurable. Given a filtration $\{\mathcal{F}_n\}_{n \geq 0}$, a *stopping time* is a random variable $\tau : \Omega \to \mathbb{N}$, such that for every $n \geq 0$, $\{\omega \in \Omega \mid \tau(\omega) \leq n\} \in \mathcal{F}_n$.

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.

Example 1. 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. □

We refer to standard textbooks (such as [6,34]) for a detailed treatment of all the concepts illustrated above.

2.1 Probabilistic Recurrence Relations

In this work, we focus on probabilistic recurrence relations (PRRs) that describe the runtime behaviour of a single recursive procedure. Instead of having a direct syntax for a PRR, we propose a mini programming language *LRec* that captures a wide class of PRRs that have common probability distributions such as (piecewise) uniform distributions and discrete probability distributions, and whose recursive call consists of either a procedure call or two procedure calls in a divide-and-conquer style. We present the grammar of *LRec* in Fig. 1.

$$
\begin{array}{rrl}
\textbf{(PRR)} & \mathsf{proc} ::= & \texttt{def}\ p(n; c_p) = \{\mathsf{comm}\} \\
\textbf{(Command)} & \mathsf{comm} ::= & \texttt{sample}\ v \leftarrow \mathsf{dist}\ \texttt{in}\ \{\mathsf{body}\} \mid \bigoplus_{i=1}^{k} c_i : \mathsf{comm}_i \\
\textbf{(Recursive Body)} & \mathsf{body} ::= & \texttt{pre}(\mathsf{expr}); \texttt{invoke}\ \mathsf{call} \\
\textbf{(Recurive Call)} & \mathsf{call} ::= & p(v); p(\mathsf{size} - v) \mid p(v) \mid p(\mathsf{size} - v) \\
 & & (\text{where size is either } \lfloor \tfrac{n}{b} \rfloor + c \text{ or } \lceil \tfrac{n}{b} \rceil + c) \\
\textbf{(Distribution)} & \mathsf{dist} ::= & \texttt{uniform}(n) \mid \texttt{muniform}(n) \mid \texttt{discrete} \mid \ldots \\
\textbf{(Expression)} & \mathsf{expr} ::= & v \mid v^{-1} \mid \ln v \mid n \mid \ln n \mid n^{-1} \mid c \\
 & & \mid \mathsf{expr} + \mathsf{expr} \mid \mathsf{expr} - \mathsf{expr} \mid \mathsf{expr} \times \mathsf{expr}
\end{array}
$$

Fig. 1. The Grammar of *LRec*

In the grammar, we have two positive-integer valued variables n, v which stand for the input size and the sampled value in the randomization of the passed size to the recursive calls of a procedure, respectively. We use $b > 0, c, c_p$ to denote integer constants, and use p to denote the name of the single procedure in the PRR. We consider arithmetic expressions expr as polynomials over $v, v^{-1}, \ln v$ and $n, n^{-1}, \ln n$ (which we call *pseudo-polynomials* in this work) and common probability distributions, including (i) the uniform distribution $\texttt{uniform}(n)$ over $\{0, 1, \ldots, n-1\}$, (ii) the piecewise uniform distribution $\texttt{muniform}(n)$ that returns $\max\{i, n-i-1\}$ where i observes the uniform distribution $\texttt{uniform}(n)$, and (iii) any FSDPD (indicated by discrete) whose probabilities and values are constants and pseudo-polynomials, respectively. We also support other piecewise uniform

distribution, e.g., the distribution that each $v \in \{0, \ldots, n/2\}$ has probability $\frac{2}{3n}$ and each $v \in \{n/2+1, \ldots, n-1\}$ has probability $\frac{4}{3n}$.

The nonterminal proc generates the PRR in the form $\texttt{def}\ p(n; c_p) = \{\mathsf{comm}\}$, for which c_p is an integer constant as the threshold of recursion, meaning that the procedure halts immediately when $n < c_p$, and comm is the function body of the procedure. The nonterminal comm generates all statements with one of the two forms as follows.

- A sampling statement (indicated by sample) followed by first a special expression $\texttt{pre}(\mathsf{expr})$ that stands for the preprocessing time of expr amount, then the recursive calls generated by the nonterminal call.
- A probabilistic choice in the form $\bigoplus_{i=1}^{k} c_i{:}\mathsf{comm}_i$ where each statement comm_i is executed with probability c_i.

We restrict the recursive calls to be either a single recursive call $p(v)$ or $p(\mathsf{size} - v)$, or a divide-and-conquer composed of two consecutive recursive calls $p(v)$ and $p(\mathsf{size} - v)$, for which we consider a general setting that the relevant overall size size is in the form of the input size n divided by some positive integer b with possibly an offset c. Choosing $b = 1, c = -1$ means the normal situation that the overall size is $n - 1$, i.e., removing one element from the original input.

Given a PRR p, we use $\mathsf{func}(p)$ to represent its function body.

We always assume that the given PRR is *well-formed*, i.e., every c_i in a probabilistic choice is within $[0, 1]$ and every random passed size (e.g. $v, \mathsf{size} - v$) falls in $[0, n]$. Below, we present two examples for PRRs.

Example 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 [17] with $O(n)$ expected running time. We model the algorithm as the following PRR:

$$\texttt{def}\ p(n; 2) = \{\texttt{sample}\ v \leftarrow \texttt{muniform}(n)\ \texttt{in}\ \{\texttt{pre}(n);\ \texttt{invoke}\ p(v);\}\}$$

Here, we use $p(n; 2)$ to represent the number of comparisons performed by QuickSelect over an input of size n, and v is the variable that captures the size of the remaining array that has to be searched recursively. It observes as the value $\max\{i, n-1-i\}$ where the value of i is sampled uniformly from $\{0, \ldots, n-1\}$, we use $\texttt{muniform}(n)$ to represent this distribution. □

Example 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 [16]. We model the algorithm as the following PRR.

$$\texttt{def}\ p(n; 2) = \{\texttt{sample}\ v \leftarrow \texttt{uniform}(n)\ \texttt{in}\ \{\texttt{pre}(n);\ \texttt{invoke}\ p(v); p(n-1-v);\}\}$$

Here, v and $n - 1 - v$ capture the sizes of the two sub-arrays. □

Below we present the semantics of a PRR in a nutshell. Consider a PRR generated by *LRec* with the procedure name p, a *configuration* σ is a pair $\sigma =$

$(comm, \widehat{n})$ where $comm$ represents the current statement to be executed and $\widehat{n} \geq c_p$ is the current value for the variable n. A *PRR state* μ is a triple $\langle \sigma, C, \mathbf{K} \rangle$ for which:

- σ is either a configuration, or halt for the termination of the whole PRR.
- $C \geq 0$ records the cumulative preprocessing time so far.
- $\mathbf{K}$ is a stack of configurations that remain to be executed.

We use emp to denote an empty stack, and say that a PRR state $\langle \sigma, C, \mathbf{K} \rangle$ is *final* if $\mathbf{K} = \mathsf{emp}$ and $\sigma = \mathsf{halt}$. Note that in a final PRR state $\langle \mathsf{halt}, C, \mathsf{emp} \rangle$, the value C represents the total execution runtime of the PRR. The semantics of the PRR is defined as a discrete-time Markov chain whose state space is the set of all PRR states and whose transition function $\mathbf{P}$, where $\mathbf{P}(\mu, \mu')$ is the probability that the next PRR state is μ' given the current PRR state is $\mu = ((comm, \widehat{n}), C, \mathbf{K})$. The probability is determined by the following cases.

- For final PRR states μ, $\mathbf{P}(\mu, \mu) := 1$ and $\mathbf{P}(\mu, \mu') := 0$ for other $\mu' \neq \mu$. This means that the PRR stays at termination once it terminates.
- In the divide-and-conquer case $comm = \texttt{sample}\ v \leftarrow dist\ \texttt{in}\ \{\texttt{pre}(e); \texttt{invoke}\ p(v); p(s-v)\}$, we first sample v from the distribution $dist$. Then, with probability $dist(v)$, we accumulate the preprocessing time e into the cumulative processing time C. We recursively invoke $p(v)$ and push the remaining task $p(s-v)$ into the stack. The probability for the single recursion case is defined analogously. The only difference is that there is no need to push some recursive call into the stack in the single recursion case.
- In the case $comm = \bigoplus_{i=1}^{k} c_i : comm_i$, we have that $\mathbf{P}(\mu, \mu_i) = c_i$ for each $1 \leq i \leq k$ for which we have $\mu_i := ((comm_i, \widehat{n}), C, \mathbf{K})$.

With an initial PRR state $((\mathsf{func}(p), n^*), 0, \mathsf{emp})$ where $n^* \geq c_p$ is the input size, the Markov chain induces a probability space where the sample space is the set of all infinite sequences of PRR states, the σ-algebra is generated by all *cylinder sets* over infinite sequences of PRR states, and the probability measure is uniquely determined by the transition function $\mathbf{P}$. We refer to [3] for details. We use $\Pr_{n^*}$ for the probability measure where $n^* \geq c_p$ is the input size.

We further define the random variable τ such that for any infinite sequence of PRR states $\rho = \mu_0, \mu_1, \ldots, \mu_t, \ldots$ with each $\mu_t = ((comm_t, \widehat{n}_t), C_t, \mathbf{K}_t)$, $\tau(\rho)$ equals the first moment that the sequence reaches a final PRR state, i.e., $\tau(\rho) = \inf\{t \mid \text{the PRR state } \mu_t \text{ is final}\}$, for which $\inf \emptyset = \infty$. We will always ensure that τ is almost-surely finite, i.e., $\Pr_{n^*}(\tau < \infty) = 1$). Note that the random cumulative processing time C_τ in the PRR state $\mu_\tau \in \rho$ is the total execution time of the given PRR.

We formulate the tail bound analysis over PRRs as follows. Given a time limit $\alpha \cdot \kappa(n^*)$ symbolic in the initial input n^* and the coefficient α, the goal of tail bound analysis is to infer an upper bound $u(\alpha, n^*)$ symbolic in n^* and α such that for every input size n^* and plausible value for α, we have that

$$\Pr_{n^*}[C_\tau \geq \alpha \cdot \kappa(n^*)] \leq u(\alpha, n^*). \tag{1}$$

As tails bounds are often evaluated asymptotically, we focus on deriving tight $u(\alpha, n^*)$ when α, n^* are sufficiently large. To compare the magnitude of two tail bounds, we follow the straightforward way that first treats α as a fixed constant and compares the bounds over n^*, and then if the magnitude over n^* is identical, we take a further comparison over the magnitude on the coefficient α.

Example 4 (Our result on QuickSelect). Continue with Example 2, suppose the user is interested in the tail bound $\Pr[C_\tau \geq \alpha \cdot n^*]$, where C_τ is the running time of the QuickSelect algorithm over an array with length n^*. Then, Karp's method produces the symbolic tail bound as follows.

$$\Pr[C_\tau \geq \alpha \cdot n^*] \leq \exp(1.15 - 0.28 \cdot \alpha)$$

However, our method can produce the following tail bound.

$$\Pr[C_\tau \geq \alpha \cdot n^*] \leq \exp(2 \cdot \alpha - \alpha \cdot \ln \alpha)$$

Note that our method produces tail bounds with a better magnitude on α. □

Example 5 (Our result on QuickSort). Continue with Example 3, consider the tail bound $\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*]$, where C_τ is the running time of QuickSort over a length-n^* array. Then, Karp's method produces the symbolic tail bound as:

$$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*] \leq \exp(0.5 - 0.5 \cdot \alpha),$$

while our method can produce the bound as:

$$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*] \leq \exp((4 - \alpha) \cdot \ln n^*)$$

Note that our method produces tail bounds with a better magnitude on n^*. □

3 Exponential Tail Bounds via Markov's Inequality

In this section, we demonstrate our theoretical approach for deriving exponentially decreasing tail bounds based on Markov's inequality.

Before illustrating our approach, we first translate a PRR in the language *LRec* with the single procedure p into the canonical form as follows.

$$p(n; c_p) = \texttt{pre}(S(n)); \texttt{invoke}\ p(\mathsf{size}_1(n)); \ldots; p(\mathsf{size}_r(n)) \tag{2}$$

where (i) $S(n)$ is a random variable related to the input size n that represents the randomized pre-processing time and observes a probability distribution resulting from a discrete probability choice of piecewise uniform distributions, and (ii) $\texttt{invoke}\ p(\mathsf{size}_1(n)); \ldots; p(\mathsf{size}_r(n))$ is a statement that is either a single recursive call $p(\mathsf{size}_1(n))$ or a divide-and-conquer $p(\mathsf{size}_1(n)); p(\mathsf{size}_2(n))$ upon the resolution of the randomization. For the latter, we use a random variable r (which is either 1 or 2) to represent the number of recursive calls.

The translation can be implemented by a straightforward recursive procedure $\mathsf{Tf}(n, Prog)$ that takes on input a positive integer n (as the input size) and a statement $Prog$ (generated by the nonterminal comm) to be processed, Note that the procedure $\mathsf{Tf}(n, Prog)$ outputs the *joint* distribution of the random value $S(n)$ and the recursive call $p(\mathsf{size}_1(n)); \ldots; p(\mathsf{size}_r(n))$ with randomized input size. These random variables may be dependent.

Our theoretical approach then works directly on the canonical form (2). It consists of two major steps to derive an exponentially-decreasing tail bound. In the first step, we apply Markov's inequality and reduce the tail bound analysis problem to the over-approximation of the moment generating function $\mathbb{E}[\exp(t \cdot C_\tau)]$ where C_τ is the cumulative pre-processing time defined previously and $t > 0$ is a scaling factor that aids the derivation of the tail bound. In the second step, we apply Optional Stopping Theorem (a classical theorem in martingale theory) to over-approximate the expected value $\mathbb{E}[\exp(t \cdot C_\tau)]$. Below we fix an PRR with procedure p in the canonical form (2), and a time limit $\alpha \cdot \kappa(n^*)$.

Our first step applies Markov's inequality. Our approach relies on the well-known exponential form of Markov's inequality below.

Theorem 1. *For every random variable X and any scaling factor $t > 0$, we have that $\Pr[X \geq d] \leq \mathbb{E}[\exp(t \cdot X)]/\exp(t \cdot d)$.*

The detailed application of Markov's inequality to tail bound analysis requires to choose a scaling factor $t := t(\alpha, n)$ symbolic in α and n. After choosing the scaling factor, Markov's inequality gives the following tail bound:

$$\Pr[C_\tau \geq \alpha \cdot \kappa(n^*)] \leq \mathbb{E}[\exp(t(\alpha, n^*) \cdot C_\tau)]/\exp(t(\alpha, n^*) \cdot \alpha \cdot \kappa(n^*)). \quad (3)$$

The role of the scaling factor $t(\alpha, n^*)$ is to scale the exponent in the term $\exp(\kappa(\alpha, n^*))$, and this is in many cases necessary as a tail bound may not be exponentially decreasing directly in the time limit $\alpha \cdot \kappa(n^*)$.

An unsolved part in the tail bound above is the estimation of the expected value $\mathbb{E}[\exp(t(\alpha, n^*) \cdot C_\tau)]$. Our second step over-approximates the expected value $\mathbb{E}[\exp(t(\alpha, n^*) \cdot C_\tau)]$. To achieve this goal, we impose a constraint on the scaling factor $t(\alpha, n)$ and an extra function $f(\alpha, n)$ and show that once the constraint is fulfilled, then one can derive an upper bound for $\mathbb{E}[\exp(t(\alpha, n^*) \cdot C_\tau)]$ from $t(\alpha, n)$ and $f(\alpha, n)$. The theorem is proved via Optional Stopping Theorem. The theorem requires the almost-sure termination of the given PRR, a natural prerequisite of exponential tail bound. In this work, we consider PRRs with finite termination time that implies the almost-sure termination.

Theorem 2. *Suppose we have functions $t, f : [0, \infty) \times \mathbb{N} \to [0, \infty)$ such that*

$$\mathbb{E}[\exp(t(\alpha, n) \cdot \mathsf{Ex}(n \mid f))] \leq \exp(t(\alpha, n) \cdot f(\alpha, n)) \quad (4)$$

for all sufficiently large $\alpha, n^* > 0$ and all $c_p \leq n \leq n^*$, where

$$\mathsf{Ex}(n \mid f) := S(n) + \textstyle\sum_{i=1}^{r} f(\alpha, \mathsf{size}_i(n)).$$

Then for $t_*(\alpha, n^*) := \min_{c_p \le n \le n^*} t(\alpha, n)$*, we have that*

$$\mathbb{E}[\exp(t_*(\alpha, n^*) \cdot C_\tau)] \le \mathbb{E}[\exp(t_*(\alpha, n^*) \cdot f(\alpha, n^*))].$$

Thus, we obtain the upper bound $u(\alpha, n^*) := \exp(t_*(\alpha, n^*) \cdot (f(\alpha, n^*) - \alpha \cdot \kappa(n^*)))$ *for the tail bound in (1).*

Proof Sketch. We fix a procedure p, and some sufficiently large α and n^*. In general, we apply the martingale theory to prove this theorem. To construct a martingale, we need to make two preparations.

First, by the convexity of $\exp(\cdot)$, substituting $t(\alpha, n)$ with $t_*(\alpha, n^*)$ in (4) does not affect the validity of (4).

Second, given an infinite sequence of the PRR states $\rho = \mu_0, \mu_1, \ldots$ in the sample space, we consider the subsequence $\rho' = \mu'_0, \mu'_1, \ldots$ as follows, where we represent μ'_i as $((\mathsf{func}(p), \hat{n}'_i), C'_i, \mathbf{K}'_i)$. It only contains states that are either final or at the entry of p, i.e., $comm = \mathsf{func}(p)$. We define $\tau' := \inf\{t : \mu'_t \text{ is final}\}$, then it is straightforward that $C'_{\tau'} = C_\tau$. We observe that μ'_{i+1} represents the recursive calls of μ'_i. Thus, we can characterize the conditional distribution $\mu'_{i+1} \mid \mu_i$ by the transformation function $\mathsf{Tf}(\hat{n}, \mathsf{func}(p))$ as follows.

- We first draw $(S, \mathsf{size}_1, \mathsf{size}_2, r)$ from $\mathsf{Tf}(\hat{n}'_i, \mathsf{func}(p))$.
- We accumulate S into the global cost. If there is a single recursion ($r = 1$), we invoke this sub-procedure. If there are two recursive calls, we push the second call $p(\mathsf{size}_2)$ into the stack and invoke the first one $p(\mathsf{size}_1)$.

Now we construct the super-martingale as follows. For each $i \ge 0$, we denote the stack as $\mathbf{K}'_i$ for μ'_i as $(\mathsf{func}(p), \mathsf{s}_{i,1}) \cdots (\mathsf{func}(p), \mathsf{s}_{i,q_i})$, where q_i is the stack size. We prove that another process $y_0, y_1, \ldots$ that forms a super-martingale, where $y_i := \exp\left(t_*(\alpha, n^*) \cdot \left(C'_i + f(\alpha, \hat{n}'_i) + \sum_{j=1}^{q_i} f(\alpha, \mathsf{s}_{i,j})\right)\right)$. Note that $y_0 = \exp(t_*(\alpha, n^*) \cdot f(\alpha, n^*))$, and $y_{\tau'} = \exp(t_*(\alpha, n^*) \cdot C'_{\tau'}) = \exp(t_*(\alpha, n^*) \cdot C_\tau)$. Thus we informally have that $\mathbb{E}[\exp(t_*(\alpha, n^*) \cdot C_\tau)] = \mathbb{E}[y_{\tau'}] \le \mathbb{E}[y_0] = \exp(t_*(\alpha, n^*) \cdot f(\alpha, n^*))$ and the theorem follows. □

It is natural to ask whether our theoretical approach can always find an exponential-decreasing tail bound over PRRs. We answer this question by showing that under a difference boundedness and a monotone condition, the answer is yes. We first present the difference boundedness condition (A1) and the monotone condition (A2) for a PRR Δ in the canonical form (2) as follows.

(A1) Δ is *difference-bounded* if there exist two real constants $M' \le M$, such that for every $n \ge c_p$, and every possible value $(V, s_1, \ldots, s_k)$ in the support of the probability distribution $\mathsf{Tf}(n, \mathsf{func}(p))$, we have that

$$M' \cdot \mathbb{E}[S(n)] \le V + (\sum_{i=1}^{k} \mathbb{E}[p(s_i)]) - \mathbb{E}[p(n)] \le M \cdot \mathbb{E}[S(n)].$$

(A2) Δ is *expected non-decreasing* if $\mathbb{E}[S(n)]$ does not decrease as n increases.

In other words, (A1) says that for any possible concrete pre-processing time V and passed sizes $s_1, \ldots, s_k$, the difference between the expected runtime before and after the recursive call is bounded by the magnitude of the expected pre-processing time. (A2) simply specifies that the expected pre-processing time be monotonically non-decreasing.

With the conditions (A1) and (A2), our theoretical approach guarantees a tail bound that is exponentially decreasing in the coefficient α and the ratio $\mathbb{E}[p(n^*)]/\mathbb{E}[S(n^*)]$. The theorem statement is as follows.

Theorem 3. *Let Δ be a PRR in the canonical form (2). If Δ satisfies (A1) and (A2), then for any function $w : [1, \infty) \to (1, \infty)$, the functions f, t given by*

$$f(\alpha, n) := w(\alpha) \cdot \mathbb{E}[p(n)] \text{ and } \qquad t(\alpha, n) := \frac{\lambda(\alpha)}{\mathbb{E}[S(n)]}$$
$$\text{with} \qquad \lambda(\alpha) := \frac{8(w(\alpha) - 1)}{w(\alpha)^2 (M_2 - M_1)^2}$$

fulfill the constraint (4) in Theorem 2. Furthermore, by choosing $w(\alpha) := \frac{2\alpha}{1+\alpha}$ in the functions f, t above and $\kappa(\alpha, n^*) := \alpha \cdot \mathbb{E}[p(n^*)]$, one obtains the tail bound

$$\Pr[C_\tau \geq \alpha \mathbb{E}[p(n^*)]] \leq \exp\left(-\frac{2(\alpha-1)^2}{\alpha(M_2 - M_1)^2} \cdot \frac{\mathbb{E}[p(n^*)]}{\mathbb{E}[S(n^*)]}\right).$$

Proof Sketch. We first rephrase the constraint (4) as

$$\mathbb{E}\left[\exp\left(t(\alpha, n) \cdot (S(n) + \sum\nolimits_{i=1}^{r} f(\alpha, \mathsf{size}_i(n)) - f(\alpha, n))\right)\right] \leq 1$$

Then we focus on the exponent in the $\exp(\cdot)$, by (A1), the exponent is a bounded random variable. By further calculating its expectation and applying Hoeffding's Lemma [18], we obtain the theorem above. □

Note that since $\mathbb{E}[p(n)] \geq \mathbb{E}[S(n)]$ when $n \geq c_p$, the tail bound is at least exponentially-decreasing with respect to the coefficient α. This implies that our theoretical approach derives tail bounds that are at least as tight as Karp's method when (A1) and (A2) holds. When $\mathbb{E}[p(n)]$ is of a strictly greater magnitude than $\mathbb{E}[S(n)]$, our approach derives asymptotically tighter bounds.

Below, we apply the theorem above to prove tail bounds for Quickselect (Example 2) and Quicksort (Example 3).

Example 6. For QuickSelect, its canonical form is $p(n; 2) = n + p(\mathsf{size}_1(n))$, where $\mathsf{size}_1(n)$ observes as $\texttt{muniform}(n)$. Solving the recurrence relation, we obtain that $\mathbb{E}[p(n)] = 4 \cdot n$. We further find that this PRR satisfies (A1) with two constants $M' = -1, M = 1$. Note that the PRR satisfies (A2) obviously. Hence, we apply Theorem 3 and derive the tail bound for every sufficiently large α:

$$\Pr[C_\tau \geq 4 \cdot \alpha \cdot n^*] \leq \exp\left(-\frac{2(\alpha-1)^2}{\alpha}\right).$$

On the other hand, Karp's cookbook has the tail bound

$$\Pr[C_\tau \geq 4 \cdot \alpha \cdot n^*] \leq \exp\left(1.15 - 1.12 \cdot \alpha\right).$$

Our bound is asymptotically the same as Karp's but has a better coefficient. □

Example 7. For QuickSort, its canonical form is $p(n;2) = n + p(\mathsf{size}_1(n)) + p(\mathsf{size}_2(n))$, where $\mathsf{size}_1(n)$ observes as $\texttt{muniform}(n)$ and $\mathsf{size}_2(n) = n - 1 - \mathsf{size}_1(n)$. Similar to the example above, we first calculate $\mathbb{E}[p(n)] = 2 \cdot n \cdot \ln n$. Note that this PRR also satisfies two assumptions above with two constants $M' = -2\log 2, M = 1$. Hence, for every sufficiently large α, we can derive the tail bound as follows:

$$Pr[C_\tau \geq 2 \cdot \alpha \cdot n^* \cdot \ln n^*] \leq \exp\left(-\frac{0.7(\alpha-1)^2}{\alpha} \cdot \ln n^*\right).$$

On the other hand, Karp's cookbook has the tail bound

$$Pr[C_\tau \geq 2 \cdot \alpha \cdot n^* \cdot \ln n^*] \leq \exp\left(-\alpha + 0.5\right).$$

Note that our tail bound is tighter than Karp's with a $\ln n$ factor. □

From the generality of Markov's inequality, our theoretical approach can handle to general PRRs with three or more sub-procedure calls. However, the tail bounds derived from Theorem 3 is still not tight since the theorem only uses the expectation and bound of the given distribution. For example, for QuickSelect, the tightest known bound $\exp(-\Theta(\alpha \cdot \ln \alpha))$ [15], is tighter than that derived from Theorem 3. Below, we present an algorithmic approach that fully utilizes the distribution information and derives tight tail bounds that can match [15].

4 An Algorithmic Approach

In this section, we demonstrate an algorithmic implementation for our theoretical approach (Theorem 2). Our algorithm synthesizes the functions t, f through template and a refined estimation on the exponential terms from the inequality (4). The estimation is via integration and the monotonicity of the template. Below we fix a PRR $p(n; c_p)$ in the canonical form (2) and a time limit $\alpha \cdot \kappa(n^*)$.

Recall that to apply Theorem 2, one needs to find functions t, f that satisfy the constraint (4). Thus, the first step of our algorithm is to have pseudo-monomial template for $f(\alpha, n)$ and $t(\alpha, n)$ in the following form:

$$f(\alpha, n) := c_f \cdot \alpha^{p_f} \cdot \ln^{q_f} \alpha \cdot n^{u_f} \cdot \ln^{v_f} n \tag{5}$$

$$t(\alpha, n) := c_t \cdot \alpha^{p_t} \cdot \ln^{q_t} \alpha \cdot n^{u_t} \cdot \ln^{v_t} n \tag{6}$$

In the template, we have $p_f, q_f, u_f, v_f, p_t, q_t, u_t, v_t$ are given integers, and $c_f, c_t > 0$ are unknown positive coefficients to be solved. For several compatibility reasons (see Proposition 1 and 2 in the following), we require that $u_f, v_f \geq 0$ and

$u_t, v_t \leq 0$. We say that the concrete values $\overline{c_f}, \overline{c_t}$ for the unknown coefficients $c_f, c_t > 0$ are *valid* if the concrete functions $\overline{f}, \overline{t}$ obtained by substituting $\overline{c_f}, \overline{c_t}$ for c_f, c_t in the template (5) and (6) satisfy the constraint (4) for every sufficiently large $\alpha, n^* \geq 0$ and all $c_p \leq n \leq n^*$.

We consider the pseudo-polynomial template since the runtime behavior of randomized algorithms can be mostly captured by pseudo-polynomials. We choose monomial templates since our interest is the asymptotic magnitude of the tail bound. Thus, only the monomial with the highest degrees matter.

Our algorithm searches the values for $p_f, q_f, u_f, v_f, p_t, q_t, u_t, v_t$ by an enumeration within a bounded range $\{-B, \ldots, B\}$, where B is a manually specified positive integer. To avoid exhaustive enumeration, we use the following proposition to prune the search space.

Proposition 1. *Suppose that we have functions $t, f : [0, \infty) \times \mathbb{N} \to [0, \infty)$ that fulfill the constraint (4). Then it holds that (i)*

$(p_f, q_f) \leq (1, 0)$ and $(p_t, q_t) \geq (-1, 0)$, and (ii)

$f(\alpha, n) = \Omega(\mathbb{E}[p(n)])$, $f(\alpha, n) = O(\kappa(n))$ and $t(\alpha, n) = \Omega(\kappa(n)^{-1})$ for any fixed $\alpha > 0$, where we write $(a, b) \leq (c, d)$ for the lexicographic order, i.e., $(a \leq c) \wedge (a = c \to b \leq d)$.

Proof. Except for the constraint that $f(\alpha, n) = \Omega(\mathbb{E}[p(n)])$, the other constraints simply ensure that the tail bound is exponentially-decreasing. To see why $f(\alpha, n) = \Omega(\mathbb{E}[p(n)])$, we apply Jensen's inequality [27] to (4) and obtain $f(n) \geq \mathbb{E}[\mathsf{Ex}(n|f)] = \mathbb{E}[S(n) + \sum_{i=1}^{r} f(\mathsf{size}_i(n))]$. Then we imitate the proof of Theorem 2 and derive that $f(n) \geq \mathbb{E}[p(n)]$. □

Proposition 1 shows that it suffices to consider (i) the choice of u_f, v_f that makes the magnitude of f to be within $\mathbb{E}[p(n)]$ and $\kappa(n)$, (ii) the choice of u_t, v_t that makes the magnitude of t^{-1} within $\kappa(n)$, and (iii) the choice of p_f, q_f, p_t, q_t that fulfills $(p_f, q_f) \leq (1, 0), (p_t, q_t) \geq (-1, 0)$. Note that an over-approximation of $\mathbb{E}[p(n)]$ can be either obtained manually or derived from automated approaches [9].

Example 8. Consider the quickselect example (Example 2), suppose we are interested in the tail bound $\Pr[C_\tau \geq \alpha \cdot n]$, and we enumerate the eight integers in the template from -1 to 1. Since $\mathbb{E}[p(n)] = 4 \cdot n$, by the proposition above, we must have that $(u_f, v_f) = (1, 0)$, $(u_t, v_t) \geq (-1, 0)$, $(p_t, q_t) \geq (-1, 0)$, $(p_f, q_f) \leq (1, 0)$. This reduces the number of choices for the template from 1296 to 128, where these numbers are automatically generated by our implementation. A choice is $f(\alpha, n) := c_f \cdot \alpha \cdot (\ln \alpha)^{-1} \cdot n$ and $t(\alpha, n) := c_t \cdot \ln \alpha \cdot n^{-1}$. □

In the second step, our algorithm solves the unknown coefficients c_t, c_f in the template. Once they are solved, our algorithm applies Theorem 2 to obtain the tail bound. In detail, our algorithm computes $t_*(\alpha, n^*)$ as the minimum of $t(\alpha, n)$ over $c_p \leq n \leq n^*$, and by $u_t, v_t \leq 0$, $t_*(\alpha, n^*)$ is simply $t(\alpha, n^*)$, so that we obtain the tail bound $u(\alpha, n^*) = \exp(t(\alpha, n^*) \cdot (f(\alpha, n^*) - \alpha \cdot \kappa(n^*)))$.

Example 9. Continue with Example 8. Suppose we have successfully found that $\overline{c_f} = 2, \overline{c_t} = 1$ is a valid concrete choice for the unknown coefficients in the

template. Then $t_*(\alpha, n^*)$ is $t(\alpha, n^*) = \ln \alpha \cdot (n^*)^{-1}$, and we have the tail bound $u(\alpha, n^*) = \exp(2 \cdot \alpha - \alpha \cdot \ln \alpha)$, which has better magnitude than the tail bound by Karp's method and our Theorem 3 (See Example 6). □

Our algorithm follows the guess-and-check paradigm. The guess procedure explores possible values $\overline{c_f}, \overline{c_t}$ for c_f, c_t and invokes the check procedure to verify whether the current choice is valid. Below we present the guess procedure in Sect. 4.1, and the check procedure in Sect. 4.2.

4.1 The Guess Procedure Guess(f, t)

The pseudocode for our guess procedure Guess(f, t) is given in Algorithm 1. In detail, it first receives a positive integer M as the doubling and halving number (Line 1), then iteratively enumerates possible values for the unknown coefficients c_f and c_t by doubling and halving for M times (Line 3 – Line 4), and finally calls the check procedure (Line 5). It is justified by the following theorem.

Theorem 4. *Given the template for $f(\alpha, n)$ and $t(\alpha, n)$ as in (5) and (6), if $\overline{c_f}, \overline{c_t}$ are valid choices, then (i) for every $k > 1$, $k \cdot \overline{c_f}, \overline{c_t}$ remains to be valid, and (ii) for every $0 < k < 1$, $\overline{c_f}, k \cdot \overline{c_t}$ remains to be valid.*

Algorithm 1: Guess Procedure

```
   Input  : Template for f(α, n) and
            t(α, n) as in (5) and (6)
   Output: c_f, c_t > 0 for (5) and (6)
1  Parameter: M for the maximum steps
     of doubling and halving.
2  Procedure Guess(f, t):
3    for c_t := 1, 2^-1, ..., 2^-M do
4      for c_f := 1/2, 1, 2, ..., 2^(M-1) do
5        if CheckCond(c_f, c_t) then
6          Return (c_f, c_t)
```

By Theorem 4, if the check procedure is sound and complete (i.e., CheckCond always terminates and $\overline{c_f}, \overline{c_t}$ fulfills the constraint (4) iff CheckCond$(\overline{c_f}, \overline{c_t})$ returns true), then the guess procedure guarantees to find a solution $\overline{c_f}, \overline{c_t}$ (if it exists) when the parameter M is large enough.

Example 10. Continued with Example 8, suppose $M = 2$, we enumerate $\overline{c_f}$ from $\{\frac{1}{2}, 1, 2\}$, and $\overline{c_t}$ from $\{1, \frac{1}{2}, \frac{1}{4}\}$. We try every possible combination, and we find that CheckCond$(2, 1)$ returns true. Thus, we return $(2, 1)$ as the result. In Sect. 4.2, we will show how to conclude that CheckCond$(2, 1)$ is true. □

4.2 The Check Procedure CheckCond$(\overline{c_f}, \overline{c_t})$

The check procedure takes as input the concrete values $\overline{c_f}, \overline{c_t}$ for the unknown coefficients in the template, and outputs whether they are valid. It is the most involved part in our algorithm due to the difficulty to tackle the validity of the constraint (4) that involves the composition of polynomials, exponentiation and logarithms. The existence of a sound and complete decision procedure for such validity is extremely difficult and is a long-standing open problem [1,33].

To circumvent this difficulty, the check procedure first strengthens the original constraint (4) into a canonical constraint with a specific form, so that a decision algorithm that is sound and complete up to any additive error applies. Below we fix a PRR with procedure p in the canonical form (2). We also discuss possible extensions for the check procedure in Remark 1.

The Canonical Constraint. We first present the canonical constraint $Q(\alpha, n)$ and how to decide the canonical constraint. The constraint is given by (where $\forall^\infty$ means "for all sufficiently large α" or formally $\exists \alpha_0. \forall \alpha \geq \alpha_0$)

$$Q(\alpha, n) := \forall^\infty \alpha. \forall n \geq c_p. \left[\sum_{i=1}^{k} \gamma_i \cdot \exp(f_i(\alpha) + g_i(n)) \leq 1 \right] \tag{7}$$

subject to:

(C1) For each $1 \leq i \leq k$, we have $\gamma_i > 0$ is a positive constant, $f_i(\alpha)$ is a pseudo-polynomial in α, and $g_i(n)$ is a pseudo-polynomial in n.
(C2) For each $1 \leq i \leq k$, the exponents for n and $\ln n$ in $g_i(n)$ are non-negative.

We use $Q_L(\alpha, n)$ to represent the summation term $\sum_{i=1}^{k} \gamma_i \cdot \exp(f_i(\alpha) + g_i(n))$ in (7). Below we show that this can be checked by the algorithm *Decide* up to any additive error. We present an overview of this algorithm. We also present its pseudo-code in Algorithm 2.

The algorithm *Decide* requires an external function $\mathsf{NegativeLB}(P(n))$ that takes on input a pseudo-polynomial $P(n)$ and outputs an integer T_n^* such that $P(n) \leq 0$ for every $n \geq T_n^*$, or output $+\infty$ for the absence of T_n^*. The idea of this function is to apply the monotonicity of pseudo-polynomials. With the function $\mathsf{NegativeLB}(P(n))$, the algorithm *Decide* consists of two steps as follows.

First, we can change the bound of n from $[c_p, \infty)$ into $[c_p, T_n]$, where T_n is a constant, without affecting the soundness and completeness. This is achieved by the observation that either: (i) we can conclude $Q(\alpha, n)$ does not hold, or (ii) there is an integer T_n such that $Q_L(\alpha, n)$ is non-increasing when $n \geq T_n$. Hence, it suffices only to consider $c_p \leq n \leq T_n$. Below we show how to compute T_n by case analysis of the limit M_i of $g_i(n)$ as $n \to \infty$, for each $1 \leq i \leq k$.

- If $M_i = +\infty$, then $\exp(g_i(n) + f_i(\alpha))$ could be arbitrarily large when $n \to \infty$. As a result, we can conclude that $Q(\alpha, n)$ does not hold.
- Otherwise, by (C2), either $g_i(n)$ is a constant function, or $M_i = -\infty$. In both cases, $g_i(n)$ is non-increasing for every sufficiently large n. More precisely, there exists L_i such that $g_i'(n) \leq 0$ for every $n \geq L_i$, where $g_i'(n)$ is the derivative of $g_i(n)$. Moreover, we can invoke $\mathsf{NegativeLB}(g_i'(n))$ to get L_i.

Finally, we set T_n as the maximum of L_i's and c_p.

Second, for every integer $c_p \leq \overline{n} \leq T_n$, we substitute n with $\overline{n}$ to eliminate n in $Q(\alpha, n)$. Then, each exponent $f_i(\alpha) + g_i(\overline{n})$ becomes a pseudo-polynomial solely over α. Since we only concern sufficiently large α, we can compute the limit $R_{\overline{n}}$ for $Q_L(\alpha, \overline{n})$ as $\alpha \to \infty$. We decide based on the limit $R_{\overline{n}}$ as follows.

– If $R_{\overline{n}} < 1$ for every $c_p \leq \overline{n} \leq L$, we conclude that $Q(\alpha, n)$ holds.
– If $R_{\overline{n}} \geq 1$ for some $c_p \leq \overline{n} \leq L$, we conclude that $Q(\alpha, n)$ does not hold to ensure soundness.

Algorithm 2: The Decision procedure for canonical constraints

Input : A canonical constraint $Q(\alpha, n)$ in the form of (7)
Output: Decide whether $Q(\alpha, n)$ holds.

Procedure *Decide*$(Q(\alpha, n))$**:**
  $T_n := c_p$; // ◁ The first step
  for $i := 1, 2, \ldots, k$ **do**
    $M_i :=$ The limit of $g_i(n)$ as $n \to \infty$.
    if $M_i = +\infty$ **then**
      Return False
    else
      $g_i'(n) :=$ the derivative of $g_i(n)$
      $T_n := \max\{T_n, \mathsf{NegativeLB}(g_i'(n))\}$
  for $\overline{n} := c_p, \ldots, T_n$ **do** // ◁ The second step
    $R := 0$
    for $i := 1, 2, \ldots, k$ **do**
      $\Delta :=$ the limit of $f_i(\alpha) + g_i(\overline{n})$ as $\alpha \to \infty$.
      if $\Delta = +\infty$ **then**
        Return False
      else
        $R := R + \gamma_i \cdot \exp(\Delta)$
      if $R \geq 1$ **then Return** False
  Return True

Algorithm *Decide* is sound, and complete up to any additive error, as is illustrated by the following theorem.

Theorem 5. *Algorithm Decide has the following properties:*

– *(Completeness) If $Q(\alpha, n)$ does not hold for infinitely many α and some $n \geq c_p$, then the algorithm returns false.*
– *(Soundness) For every $\varepsilon > 0$, we have that if $Q_L(\alpha, n) \leq 1 - \varepsilon$ for all sufficiently large α and all $n \geq c_p$, then the algorithm returns true.*

The Strengthening Procedure. Then we show how to strengthen the constraint (4) into the canonical constraint (7), so that Algorithm *Decide* applies. We rephrase (4) as

$$\mathbb{E}\left[\exp(t(\alpha, n) \cdot \left(S(n) + \sum\nolimits_{i=1}^{r} f(\alpha, \mathsf{size}_i(n)) - f(\alpha, n)\right)\right] \leq 1 \tag{8}$$

and consider two functions $\overline{f}, \overline{t}$ obtained by substituting the concrete values $\overline{c_f}, \overline{c_t}$ for unknown coefficients into the template (5) and (6). We observe that the joint-distribution of the random quantities $S(n), r \in \{1, 2\}$ and $\mathsf{size}_1(n), \ldots, \mathsf{size}_r(n)$ in the canonical form (2) over PRRs can be described by several probabilistic branches $\{c_1 : B_1, \ldots, c_k : B_k\}$, which corresponds to the probabilistic choice commands in the PRR. Each probabilistic branch B_i has a constant probability c_i, a deterministic pre-processing time $S_i(n)$, a fixed number of subprocedure

calls r_i, and a probability distribution for the variable v. The strengthening first handles each probabilistic branch, and then combines the strengthening results of every branch into a single canonical constraint.

The strengthening of each branch is an application of a set of rewriting rules. Intuitively, each rewriting step over-approximates and simplifies the expectation term in the LHS of (8). Through multiple steps of rewriting, we eventually obtain the final canonical constraint. Below we present the details of the strengthening for a single probabilistic branch with the single recursion case. The divide-and-conquer case follows a similar treatment, see the extended version for details.

Consider the single recursion case $r = 1$ where a probabilistic branch has deterministic pre-processing time $S(n)$, distribution dist for the variable v and passed size $H(v, n)$ for the recursive call. We have a case analysis on the distribution dist as follows.

— *Case I*: dist is a FSDPD $\texttt{discrete}\{c'_1 : \mathsf{expr}_1, \ldots, c'_k : \mathsf{expr}_k\}$, where v observes as expr_i with probability c'_i. Then the expectation in (8) is exactly:

$$\sum_{i=1}^{k} c'_i \cdot \exp\left(t(\alpha, n) \cdot S(n) + t(\alpha, n) \cdot f(\alpha, H(\mathsf{expr}_i, n)) - t(\alpha, n) \cdot f(\alpha, n)\right)$$

Thus it suffices to over-approximate the exponent $X_i(\alpha, n) := t(\alpha, n) \cdot S(n) + t(\alpha, n) \cdot f(\alpha, H(\mathsf{expr}_i, n)) - t(\alpha, n) \cdot f(n)$ into the form subject to (C1)–(C2). For this purpose, our strengthening repeatedly applies the following rewriting rules (R1)–(R4) for which $0 < a < 1$ and $b > 0$:

(R1) $f(\alpha, H(\mathsf{expr}_i, n)) \leq f(\alpha, n)$

(R2) $\ln(an - b) \leq \ln n + \ln a \quad \ln(an + b) \leq \ln n + \ln(\min\{1, a + \frac{b}{c_p}\})$

(R3) $0 \leq n^{-1} \leq c_p^{-1} \quad 0 \leq \ln^{-1} n \leq \ln^{-1} c_p$ (R4) $\lfloor \frac{n}{b} \rfloor \leq \frac{n}{b} \quad \lceil \frac{n}{b} \rceil \leq \frac{n}{b} + \frac{b-1}{b}$

(R1) follows from the well-formedness $0 \leq H(\mathsf{size}_i, n) \leq n$ and the monotonicity of $f(\alpha, n)$ with respect to n. (R2)–(R4) are straightforward. Intuitively, (R1) can be used to cancel the term $f(\alpha, H(\mathsf{size}_i, n)) - f(\alpha, n)$, (R2) simplifies the sub-expression in ln, (R3) is used to remove floors and ceils, and (R4) to remove n^{-c} and $\ln^{-c} n$ to satisfy the restriction (C2) of the canonical constraint. To apply these rules, we consider two strategies below.

(S1-D) Apply (R1) and over-approximate $X_i(\alpha, n)$ as $t(\alpha, n) \cdot S(n)$. Then, we repeatedly apply (R3) to remove terms n^{-c} and $\ln^{-c} n$.

(S2-D) Substitute f and t with the concrete functions $\overline{f}, \overline{t}$ and expand $H(\mathsf{expr}_i, n)$. Then we first apply (R4) to remove all floors and ceils, and repeatedly apply (R2) to replace all occurrences of $\ln(an+b)$ with $\ln n + \ln C$ for some constant C. By the previous replacement, the whole term $X_i(\alpha, n)$ will be over-approximated as a pseudo-polynomial over α and n. Finally, we eagerly apply (R3) to remove all terms n^{-c} and $\ln^{-c} n$.

Our algorithm first tries to apply (S2-D), if it fails to derive a canonical constraint, then we apply the alternative (S1-D) to the original constraint. If both the strategies fails, we report failure and exit the check procedure.

Example 11. Suppose v observes as $\{0.5 : n-1, 0.5 : n-2\}, S(n) := \ln n, t(\alpha,n) := \frac{\ln\alpha}{\ln n}, f(\alpha,n) := 4 \cdot \frac{\alpha}{\ln\alpha} \cdot n \cdot \ln n, H(v,n) := v$. We consider applying both strategies to the first term $\mathsf{expr}_1 := n-1$ and $X_1(\alpha,n) := t(\alpha,n)\cdot(S(n)+f(\alpha,n-1)-f(\alpha,n))$. If we apply (S1-D) to X_1, it will be approximated as $\exp(\ln\alpha)$. If we apply (S2-D) to X_1, it will be first over-approximated as $\frac{\ln\alpha}{\ln n}\cdot(\ln n + 4\cdot\frac{\alpha}{\ln\alpha}\cdot v\cdot\ln n - 4\cdot\frac{\alpha}{\ln\alpha}\cdot n\cdot\ln n)$, then we substitute $v = n-1$ and derive the final result $\exp(\ln\alpha - 4\cdot\alpha)$. Hence, both the strategies succeed. □

— *Case II*: dist is $\texttt{uniform}(n)$ or $\texttt{muniform}(n)$. Note that $H(v,n)$ is linear with respect to v, thus $H(v,n)$ is a bijection over v for every fixed n. Hence, if v observes as $\texttt{uniform}(n)$, then

$$\mathbb{E}[\exp(t(\alpha,n)\cdot f(\alpha,H(v,n)))] \le \frac{1}{n}\sum_{v=0}^{n-1}\exp(t(\alpha,n)\cdot f(\alpha,v)) \tag{9}$$

If v observes as $\texttt{muniform}(n)$, a similar inequality holds by replacing $\frac{1}{n}$ with $\frac{2}{n}$. Since $f(\alpha,v)$ is a non-decreasing function with respect to v, we further over-approximate the summation in (9) by the integral $\int_0^n \exp(t(\alpha,n)\cdot f(\alpha,v))\mathrm{d}v$.

Example 12. Continue with Example 10, we need to check $\overline{t}(\alpha,n) = \frac{\ln\alpha}{n}$ and $\overline{f}(\alpha,n) = \frac{2\cdot\alpha}{\ln\alpha}\cdot n$. By the inequality (9), we expand the constraint (8) into $\frac{2}{n}\cdot\exp(\ln\alpha - 2\cdot\alpha)\cdot\sum_{v=0}^{n-1}\exp(\frac{2\cdot\alpha\cdot i}{n})$. By integration, it is further over-approximated as $\frac{2}{n}\cdot\exp(\ln\alpha - 2\cdot\alpha)\cdot\int_0^n\exp(\frac{2\cdot\alpha\cdot v}{n})\mathrm{d}v$. □

Note that we still need to resolve the integration of an exponential function whose exponent is a pseudo-monomial over α, n, v. Below we denote by d_v the degree on the variable v and by ℓ_v the degree of $\ln v$. We first list the situations where the integral can be computed exactly.

- If $(d_v,\ell_v) = (1,0)$, then the exponent could be expressed as $W(\alpha,n)\cdot v$,where $W(\alpha,n)$ is a pseudo-monomial over α and n. We can compute the integral as $\frac{\exp(n\cdot W(\alpha,n))-1}{W(\alpha,n)}$ and over-approximate it as $\frac{\exp(n\cdot W(\alpha,n))}{W(\alpha,n)}$ by removing -1 in the numerator.
- If $(d_v,\ell_v) = (0,1)$, then the exponent is of the form $W(\alpha,n)\cdot\ln v$. We follow a similar procedure with the case above and obtain the over-approximation $\frac{n\cdot\exp(\ln n\cdot W(\alpha,n))}{W(\alpha,n)}$.
- If $(d_v,\ell_v) = (0,0)$, then the result is trivially $n\cdot\exp(W(\alpha,n))$.

Then we handle the situation where the exact computation of the integral is infeasible. In this situation, the strengthening further over-approximates the integral into simpler forms by first replacing $\ln v$ with $\ln n$, and then replacing v with n to reduce the degrees ℓ_v and d_v. Eventually, the exponent in the

integral bows down to one of the three situations (where the integral can be computed exactly) above, and the strengthening returns the exact value of the integral.

Example 13. Continue with Example 12. We express the exponent as $\frac{2\cdot\alpha}{n}\cdot v$. Thus, we can plug $\frac{2\cdot\alpha}{n}$ into $W(\alpha,n)$ and obtain the integration result $\frac{\exp(2\cdot\alpha)}{2\cdot\alpha/n}$. Furthermore, we can simplify the formula in Example 12 as $\frac{\exp(\ln\alpha)}{\alpha}$. □

In the end, we move the term $\frac{1}{n}$ (or $\frac{2}{n}$) that comes from the `uniform` (or `muniform`) distribution and the coefficient term $W(\alpha,n)$ into the exponent. If we move these terms directly, it may produce $\ln\ln n$ and $\ln\ln\alpha$ that comes from taking the logarithm of $\ln n$ and $\ln\alpha$. Hence, we first apply $\ln c_p \le \ln n \le n$ and $1 \le \ln\alpha \le \alpha$ to remove all terms $\ln n$ and $\ln\alpha$ outside the exponent (e.g., $\frac{\ln\alpha}{\ln n}$ is over-approximated as $\frac{\alpha}{\ln c_p}$). After the over-approximation, the terms outside the exponentiation form a polynomial over α and n, we can trivially move these terms into the exponent by taking the logarithm. Finally, we apply (R4) in Case I to remove n^{-c} and $\ln^{-c} n$. If we fail to obtain the canonical constraint, the strengthening reports failure.

Example 14. Continue with Example 13, we move the term α into the exponentiation and simplify the over-approximation result as $\exp(\ln\alpha - \ln\alpha) = 1$. As a result, we over-approximate the LHS of (8) as 1 and we conclude that $\mathsf{CheckCond}(2,1)$ holds. □

The details of the divide-and-conquer case are similar and omitted. Furthermore, we present how to combine the strengthening results for different branches into a single canonical constraint. Suppose for every probabilistic branch B_i, we have successfully obtained the canonical constraint $Q_{L,i}(\alpha,n) \le 1$ as the strengthening of the original constraint (8). Then, the canonical constraint for the whole distribution is $\sum_{i=1}^{k} c_i \cdot Q_{L,i}(\alpha,n) \le 1$. Intuitively, there is probability c_i for the branch B_i, thus the combination follows by simply expanding the expectation term.

A natural question is to ask whether our algorithm can always succeed to obtain the canonical constraint. We have the proposition as follows.

Proposition 2. *If the template for t has a lower magnitude than $S(n)^{-1}$ for every branch, then the rewriting always succeeds.*

Proof. We first consider the single recursion case. When dist is FSDPD, we can apply (S1-D) to over-approximate the exponent as $t(\alpha,n)\cdot S(n)$. Since $t(\alpha,n)$ has a lower magnitude than $S(n)^{-1}$, by further applying (R3) to eliminate n^{-c} and $\ln^{-c} n$, we obtain the canonical constraint. If dist is $\texttt{uniform}(n)$ or $\texttt{muniform}(n)$, we observe that the over-approximation result for the integral is

either $\frac{\exp(f(\alpha,n))}{f(\alpha,n)\cdot t(\alpha,n)}$ (when $d_v > 0$) or $\frac{\ln n\cdot\exp(f(\alpha,n))}{f(\alpha,n)\cdot t(\alpha,n)}$ (when $d_v = 0$). Thus, we can cancel the term $f(\alpha, n)$ in the exponent and obtain the canonical constraint by the subsequent steps. The proof is the same for the divide-and-conquer case. □

By Proposition 2, we restrict $u_t, v_t \leq 0$ in the template to ensure our algorithm never fails.

Remark 1. Our algorithm can be extended to support piecewise uniform distributions (e.g. each of $0, \ldots, n/2$ with probability $\frac{2}{3n}$ and each of $n/2+1, \ldots, n-1$ with probability $\frac{4}{3n}$) by handling each piece separately.

5 Experimental Results

In this section, we evaluated our algorithm over classical randomized algorithms such as QuickSort (Example 3), QuickSelect (Example 2), DiameterComputation [26, Chapter 9], RandomizedSearch [24, Chapter 9], ChannelConflictResolution [22, Chapter 13], examples such as Rdwalk and Rdadder in the literature [7], and four manually-crafted examples (MC1 – MC4). For each example, we manually compute its expected running time for the prunning.

We implemented our algorithm in `C++`. We choose $B = 2$ (as the bounded range for the template), $M = 4$ (in the guess procedure), $Q = 8$ (for the number of parts in the integral), and prune the search space by Theorem 1. All results were obtained on an Ubuntu 18.04 machine with an 8-Core Intel i7-7900x Processor (4.30 GHz) and 40 GB of RAM.

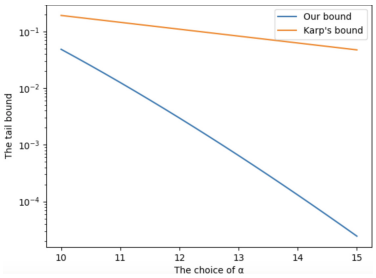

Fig. 2. Plot for QuickSelect

We report the tail bound derived by our algorithm in Table 1, where "Benchmark" lists the benchmarks, "$\alpha\cdot\kappa(n^*)$" lists the time limit of interest, "Our bound" lists the tail bound by our approach, "Time(s)" lists the runtime (in seconds) of our approach, and "Karp's bound" lists the bounds by Karp's method. From the table, our algorithm constantly derives asymtotically tighter tail bounds than Karp's method. Moreover, all these bounds are obtained in a few seconds, demonstrating the efficiency of our algorithm. Furthermore, our algorithm obtains bounds with tighter magnitude than our completeness theorem (Theorem 3) in 9 benchmarks, and bounds with the same magnitude as the others.

For an intuitive comparison, we also report the concrete bounds and their plots of our method and Karp's method. We choose three concrete choices of α and n^* and plot the concrete bounds over $10 \leq \alpha \leq 15, n^* = 17$. For concrete bounds, we also report the ratio $\frac{\text{Karp's Bound}}{\text{Our Bound}}$ to show the strength of our method. Due to space limitations, we only report the results for QuickSelect (Example 2) in Table 2 and Fig. 2.

Table 1. Experimental Result

Benchmark	$\alpha \cdot \kappa(n^*)$ in (1)	Our bound	Time(s)	Karp's bound
QuickSelect	$\alpha \cdot n^*$	$\exp(2 \cdot \alpha - \alpha \cdot \ln \alpha)$	0.03	$\exp(1.15 - 0.28 \cdot \alpha)$
QuickSort	$\alpha \cdot n^* \cdot \ln n^*$	$\exp((4 - \alpha) \cdot \ln n^*)$	0.02	$\exp(0.5 - 0.5 \cdot \alpha)$
L1Diameter	$\alpha \cdot n^*$	$\exp(\alpha - \alpha \cdot \ln \alpha)$	0.03	$\exp(1.39 - 0.69 \cdot \alpha)$
L2Diameter	$\alpha \cdot n^* \cdot \ln n^*$	$\exp(\alpha - \alpha \cdot \ln \alpha)$	0.03	$\exp(1.39 - 0.69 \cdot \alpha)$
RandSearch	$\alpha \cdot \ln n^*$	$\exp((2 \cdot \alpha - \alpha \cdot \ln \alpha) \cdot \ln n^*)$	0.03	$\exp(-0.29 \cdot \alpha \cdot \ln n^*)$
Channel	$\alpha \cdot n^*$	$\exp((8 - \alpha) \cdot n^*)$	0.05	$\exp(1 - 0.37 \cdot \alpha)$
Rdwalk	$\alpha \cdot n^*$	$\exp((0.5 - \alpha) \cdot n^*)$	0.05	$\exp(0.60 - 0.41 \cdot \alpha)$
Rdadder	$\alpha \cdot n^*$	$\exp((4 - 0.5 \cdot \alpha) \cdot n^*)$	0.04	Not applicable
MC1	$\alpha \cdot \ln n^*$	$\exp((\alpha - \alpha \cdot \ln \alpha) \cdot \ln n^*)$	0.03	$\exp(-0.69 \cdot \alpha \cdot \ln n^*)$
MC2	$\alpha \cdot \ln^2 n^*$	$\exp((\alpha - \alpha \cdot \ln \alpha) \cdot \ln n^*)$	0.03	$\exp(-0.69 \cdot \alpha \cdot \ln n^*)$
MC3	$\alpha \cdot n^* \cdot \ln^2 n^*$	$\exp(\alpha - \alpha \cdot \ln \alpha)$	0.03	$\exp(1.15 - 0.28 \cdot \alpha)$
MC4	$\alpha \cdot n^*$	$\exp(2 \cdot \alpha - \alpha \cdot \ln \alpha)$	0.04	Not applicable

Table 2. Concrete Bounds for QuickSelect

Concrete choice	Our bound	Karp's Bound	Ratio
$\alpha = 10; n^* = 13$	0.0485	0.192	3.96
$\alpha = 11; n^* = 15$	0.0126	0.145	11.6
$\alpha = 12; n^* = 17$	0.00297	0.110	36.9

6 Related Work

Karp's Cookbook. Our approach is orthogonal to Karp's cookbook method [21] since we base our approach on Markov's inequality, and the core of Karp's method is a dedicated proof for establishing that an intricate tail bound function is a prefixed point of the higher order operator derived from the given PRR. Furthermore, our automated approach can derive asymptotically tighter tail bounds than Karp's method over all 12 PRRs in our benchmark. Our approach could also handle randomized preprocessing times, which is beyond the reach of Karp's method. Since Karp's proof of prefixed point is ad-hoc, it is non-trivial to extend his method to handle the randomized cost. Nevertheless, there are PRRs (e.g., Coupon-Collector) that can be handled by Karp's method but not by ours. Thus, our approach provides a novel way to obtain asymptotically tighter tail bounds than Karp's method.

The recent work [30] extends Karp's method for deriving tail bounds for parallel randomized algorithms. This method derives the same tail bounds as Karp's method over PRRs with a single recursive call (such as QuickSelect) and cannot handle randomized pre-processing time. Compared with this approach, our approach derives tail bounds with tighter magnitude on 11/12 benchmarks.

Custom Analysis. Custom analysis of PRRs [15,25] has successfully derived tight tail bounds for QuickSelect and QuickSort. Compared with the custom analysis that requires ad-hoc proofs, our approach is automated, has the generality from Markov's inequality, and is capable of deriving bounds identical or very close to the tail bounds from the custom analysis.

Probabilistic Programs. There are also relevant approaches in probabilistic program verification. These approaches are either based on martingale concentration inequalities (for exponentially-decreasing tail bounds) [7,10–12,19], Markov's inequality (for polynomially-decreasing tail bounds) [8,23,31], fixed-point synthesis [32], or weakest precondition reasoning [4,20]. Compared with these approaches, our approach is dedicated to PRRs (a light-weight representation of recursive probabilistic programs) and involves specific treatment of common recursive patterns (such as randomized pivoting and divide-and-conquer) in randomized algorithms, while these approaches usually do not consider common recursion patterns in randomized algorithms. Below we have detailed technical comparisons with these approaches.

- Compared with the approaches based on martingale concentration inequalities [7,10–12,19], our approach has the same root as them, since martingale concentration inequalities are often proved via Markov's inequality. However, those approaches have more accuracy loss since these martingale concentration inequalities usually make further relaxations after applying Markov's inequality. In contrast, our automated approach directly handles the constraint after applying Markov's inequality by having a refined treatment of exponentiation and hence has better accuracy in deriving tail bounds.
- Compared with the approaches [8,23,31] that derive polynomially-decreasing tail bounds, our approach targets the sharper exponentially-decreasing tail bounds and hence is orthogonal.
- Compared with the fixed-point synthesis approach [32], our approach is orthogonal as it is based on Markov's inequality. Note that the approach [32] can only handle 3/12 benchmarks.
- Compared with weakest precondition reasoning [4,20] that requires first specifying the bound functions and then verifying the bound functions by proof rules related to fixed-point conditions, mainly with manual efforts, our approach can be automated and is based on Markov's inequality rather than fixed point theorems. Although Karp's method is also based on a particular tail bound function as a prefixed point and can thus be embedded into the weakest precondition framework, Karp's proof of prefixed point requires deep insight, which is beyond existing proof rules. Moreover, even a slight relaxation of the tail bound function into a simpler form in Karp's method no longer keeps the bound function to be a prefixed point. Hence, the approach of the weakest precondition may not be suitable for deriving tail bounds.

Acknowledgement. We thank Prof. Bican Xia for valuable information on the exponential theory of reals. The work is partially supported by the National Natural Science Foundation of China (NSFC) with Grant No. 62172271, ERC CoG 863818 (ForM-SMArt), the Hong Kong Research Grants Council ECS Project Number 26208122, the HKUST-Kaisa Joint Research Institute Project Grant HKJRI3A-055 and the HKUST Startup Grant R9272.

References

1. Achatz, M., McCallum, S., Weispfenning, V.: Deciding polynomial-exponential problems. In: Sendra, J.R., González-Vega, L. (eds.) Symbolic and Algebraic Computation, International Symposium, ISSAC 2008, Linz/Hagenberg, Austria, July 20–23, 2008, Proceedings, pp. 215–222. ACM (2008). https://doi.org/10.1145/1390768.1390799
2. Aguirre, A., Barthe, G., Hsu, J., Kaminski, B.L., Katoen, J.P., Matheja, C.: A pre-expectation calculus for probabilistic sensitivity. Proc. ACM Program. Lang. **5**(POPL) (2021). https://doi.org/10.1145/3434333
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Batz, K., Kaminski, B.L., Katoen, J.P., Matheja, C., Verscht, L.: A calculus for amortized expected runtimes. Proc. ACM Program. Lang. **7**(POPL), 1957–1986 (2023). https://doi.org/10.1145/3571260
5. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series, Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07964-5
6. Billingsley, P.: Probability and Measure, 3rd edn. Wiley, New York (1995)
7. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: CAV, pp. 511–526 (2013)
8. Chatterjee, K., Fu, H.: Termination of nondeterministic recursive probabilistic programs. CoRR abs/1701.02944 (2017)
9. Chatterjee, K., Fu, H., Murhekar, A.: Automated recurrence analysis for almost-linear expected-runtime bounds. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 118–139. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_6
10. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. TOPLAS **40**(2), 7:1-7:45 (2018)
11. Chatterjee, K., Goharshady, A.K., Meggendorfer, T., Zikelic, D.: Sound and complete certificates for quantitative termination analysis of probabilistic programs. In: Shoham, S., Vizel, Y. (eds.) CAV 2022. LNCS, vol. 13371, pp. 55–78. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13185-1_4
12. Chatterjee, K., Novotný, P., Žikelić, Đ.: Stochastic invariants for probabilistic termination. In: POPL 2017, pp. 145–160 (2017)
13. Chaudhuri, S., Dubhashi, D.P.: Probabilistic recurrence relations revisited. Theoret. Comput. Sci. **181**(1), 45–56 (1997)
14. Goodman, N.D., Mansinghka, V.K., Roy, D., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: UAI 2008, pp. 220–229. AUAI Press (2008)
15. Grübel, R.: Hoare's selection algorithm: a Markov chain approach. Journal of Applied Probability **35**(1), 36–45 (1998). http://www.jstor.org/stable/3215544
16. Hoare, C.A.R.: Algorithm 64: quicksort. Commun. ACM **4**(7), 321 (1961)
17. Hoare, C.A.R.: Algorithm 65: find. Commun. ACM **4**(7), 321–322 (1961)
18. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
19. Huang, M., Fu, H., Chatterjee, K.: New approaches for almost-sure termination of probabilistic programs. In: APLAS, pp. 181–201 (2018)

20. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. J. ACM **65**(5), 30:1-30:68 (2018). https://doi.org/10.1145/3208102
21. Karp, R.M.: Probabilistic recurrence relations. J. ACM **41**(6), 1136–1150 (1994)
22. Kleinberg, J.M., Tardos, É.: Algorithm Design. Addison-Wesley (2006)
23. Kura, S., Urabe, N., Hasuo, I.: Tail probabilities for randomized program runtimes via martingales for higher moments. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 135–153. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_8
24. McConnell, J.J. (ed.): The Analysis of Algorithms: An Active Learning Approach. Jones & Bartlett Learning (2001)
25. McDiarmid, C., Hayward, R.: Large deviations for quicksort. J. Algorithms **21**(3), 476–507 (1996)
26. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
27. Rudin, W.: Real and Complex Analysis, 3rd edn. McGraw-Hill Inc, USA (1987)
28. Smith, C., Hsu, J., Albarghouthi, A.: Trace abstraction modulo probability. Proc. ACM Program. Lang. **3**(POPL) (2019). https://doi.org/10.1145/3290352
29. Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Automated tail bound analysis for probabilistic recurrence relations. CoRR (2023). http://arxiv.org/abs/2305.15104
30. Tassarotti, J., Harper, R.: Verified tail bounds for randomized programs. In: ITP, pp. 560–578 (2018)
31. Wang, D., Hoffmann, J., Reps, T.W.: Central moment analysis for cost accumulators in probabilistic programs. In: Freund, S.N., Yahav, E. (eds.) PLDI 2021: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20–25, 2021, pp. 559–573. ACM (2021). https://doi.org/10.1145/3453483.3454062
32. Wang, J., Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Quantitative analysis of assertion violations in probabilistic programs. In: Freund, S.N., Yahav, E. (eds.) PLDI, pp. 1171–1186. ACM (2021)
33. Wilkie, A.J.: Schanuel's conjecture and the decidability of the real exponential field. In: Hart, B.T., Lachlan, A.H., Valeriote, M.A. (eds.) Algebraic Model Theory. NATO ASI Series, vol. 496 pp. 223–230. Springer, Dordrecht (1997). https://doi.org/10.1007/978-94-015-8923-9_11
34. Williams, D.: Probability with Martingales. Cambridge University Press, Cambridge (1991)