Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments

Satoshi Kura[1,2](✉), Natsuki Urabe[1,2], and Ichiro Hasuo[2,3]

[1] Department of Computer Science, University of Tokyo, Tokyo, Japan
kurasatoshi@is.s.u-tokyo.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
[3] The Graduate University for Advanced Studies (SOKENDAI), Kanagawa, Japan

Abstract. Programs with randomization constructs is an active research topic, especially after the recent introduction of martingale-based analysis methods for their termination and runtimes. Unlike most of the existing works that focus on proving almost-sure termination or estimating the expected runtime, in this work we study the *tail probabilities* of runtimes—such as "the execution takes more than 100 steps with probability at most 1%." To this goal, we devise a theory of supermartingales that overapproximate *higher moments* of runtime. These higher moments, combined with a suitable concentration inequality, yield useful upper bounds of tail probabilities. Moreover, our vector-valued formulation enables automated template-based synthesis of those supermartingales. Our experiments suggest the method's practical use.

1 Introduction

The important roles of *randomization* in algorithms and software systems are nowadays well-recognized. In algorithms, randomization can bring remarkable speed gain at the expense of small probabilities of imprecision. In cryptography, many encryption algorithms are randomized in order to conceal the identity of plaintexts. In software systems, randomization is widely utilized for the purpose of fairness, security and privacy.

Embracing randomization in programming languages has therefore been an active research topic for a long time. Doing so does not only offer a solid infrastructure that programmers and system designers can rely on, but also opens up the possibility of *language-based, static* analysis of properties of randomized algorithms and systems.

The current paper's goal is to analyze imperative programs with randomization constructs—the latter come in two forms, namely probabilistic branching

T. Vojnar and L. Zhang (Eds.): TACAS 2019, Part II, LNCS 11428, pp. 135–153, 2019.
https://doi.org/10.1007/978-3-030-17465-1_8

and assignment from a designated, possibly continuous, distribution. We shall refer to such programs as *randomized programs*.[1]

Runtime and Termination Analysis of Randomized Programs. The *runtime* of a randomized program is often a problem of our interest; so is *almost-sure termination*, that is, whether the program terminates with probability 1. In the programming language community, these problems have been taken up by many researchers as a challenge of both practical importance and theoretical interest.

Most of the existing works on runtime and termination analysis follow either of the following two approaches.

- *Martingale-based methods*, initiated with a notion of *ranking supermartingale* in [4] and extended [1,6,7,11,13], have their origin in the theory of stochastic processes. They can also be seen as a probabilistic extension of *ranking functions*, a standard proof method for termination of (non-randomized) programs. Martingale-based methods have seen remarkable success in *automated synthesis* using templates and constraint solving (like LP or SDP).
- The *predicate-transformer* approach, pursued in [2,17,19], uses a more syntax-guided formalism of program logic and emphasizes reasoning by *invariants*.

The essential difference between the two approaches is not big: an invariant notion in the latter is easily seen to be an adaptation of a suitable notion of supermartingale. The work [33] presents a comprehensive account on the order-theoretic foundation behind these techniques.

These existing works are mostly focused on the following problems: deciding almost-sure termination, computing termination probabilities, and computing expected runtime. (Here "computing" includes giving upper/lower bounds.) See [33] for a comparison of some of the existing martingale-based methods.

Our Problem: Tail Probabilities for Runtimes. In this paper we focus on the problem of *tail probabilities* that is not studied much so far.[2] We present a method for *overapproximating* tail probabilities; here is the problem we solve.

Input: a randomized program Γ, and a *deadline* $d \in \mathbb{N}$
Output: an upper bound of the *tail probability* $\Pr(T_{\mathrm{run}} \geq d)$, where T_{run} is the runtime of Γ

Our target language is a imperative language that features randomization (probabilistic branching and random assignment). We also allow nondeterminism; this makes the program's runtime depend on the choice of a *scheduler* (i.e. how nondeterminism is resolved). In this paper we study the longest, worst-case runtime (therefore our scheduler is *demonic*). In the technical sections, we use the presentation of these programs as *probabilistic control graphs (pCFGs)*—this is as usual in the literature. See e.g. [1,33].

[1] With the rise of statistical machine learning, *probabilistic programs* attract a lot of attention. Randomized programs can be thought of as a fragment of probabilistic programs without *conditioning* (or *observation*) constructs. In other words, the Bayesian aspect of probabilistic programs is absent in randomized programs.

[2] An exception is [5]; see Sect. 7 for comparison with the current work.

An example of our target program is in Fig. 1. It is an imperative program with randomization: in Line 3, the value of z is sampled from the uniform distribution over the interval $[-2, 1]$. The symbol $*$ in the line 4 stands for a nondeterministic Boolean value; in our analysis, it is resolved so that the runtime becomes the longest.

```
1  x := 2;  y := 2;
2  while (x > 0 && y > 0) do
3    z := Unif (-2,1);
4    if * then
5      x := x + z
6    else
7      y := y + z
8    fi
9  od
```

Fig. 1. An example program

Given the program in Fig. 1 and a choice of a deadline (say $d = 400$), we can ask the question "what is the probability $\Pr(T_{\text{run}} \geq d)$ for the runtime T_{run} of the program to exceed $d = 400$ steps?" As we show in Sect. 6, our method gives a guaranteed upper bound 0.0684. This means that, if we allow the time budget of $d = 400$ steps, the program terminates with the probability at least 93%.

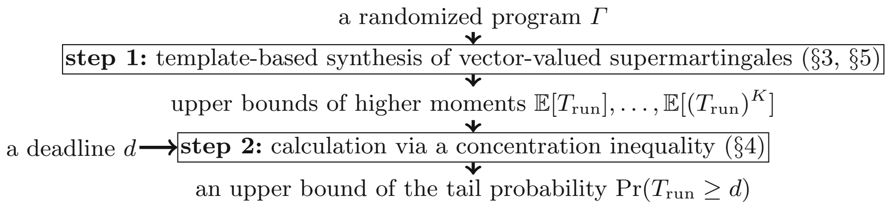

a randomized program Γ
step 1: template-based synthesis of vector-valued supermartingales (§3, §5)
upper bounds of higher moments $\mathbb{E}[T_{\text{run}}], \ldots, \mathbb{E}[(T_{\text{run}})^K]$
a deadline d
step 2: calculation via a concentration inequality (§4)
an upper bound of the tail probability $\Pr(T_{\text{run}} \geq d)$

Fig. 2. Our workflow

Our Method: Concentration Inequalities, Higher Moments, and Vector-Valued Supermartingales. Towards the goal of computing tail probabilities, our approach is to use *concentration inequalities*, a technique from probability theory that is commonly used for overapproximating various tail probabilities. There are various concentration inequalities in the literature, and each of them is applicable in a different setting, such as a nonnegative random variable (Markov's inequality), known mean and variance (Chebyshev's inequality), a difference-bounded martingale (Azuma's inequality), and so on. Some of them were used for analyzing randomized programs [5] (see Sect. 7 for comparison).

In this paper, we use a specific concentration inequality that uses *higher moments* $\mathbb{E}[T_{\text{run}}], \ldots, \mathbb{E}[(T_{\text{run}})^K]$ of runtimes T_{run}, up to a choice of the maximum degree K. The concentration inequality is taken from [3]; it generalizes Markov's and Chebyshev's. We observe that a higher moment yields a tighter bound of the tail probability, as the deadline d grows bigger. Therefore it makes sense to strive for computing higher moments.

For computing higher moments of runtimes, we systematically extend the existing theory of ranking supermartingales, from the expected runtime (i.e. the first moment) to higher moments. The theory features a *vector-valued* supermartingale, which not only generalizes easily to degrees up to arbitrary $K \in \mathbb{N}$, but also allows automated synthesis much like usual supermartingales.

We also claim that the soundness of these vector-valued supermartingales is proved in a mathematically clean manner. Following our previous work [33], our arguments are based on the order-theoretic foundation of fixed points (namely the Knaster-Tarski, Cousot–Cousot and Kleene theorems), and we give upper bounds of higher moments by suitable least fixed points.

Overall, our workflow is as shown in Fig. 2. We note that the step 2 in Fig. 2 is computationally much cheaper than the step 1: in fact, the step 2 yields a symbolic expression for an upper bound in which d is a free variable. This makes it possible to draw graphs like the ones in Fig. 3. It is also easy to find a deadline d for which $\Pr(T_{\mathrm{run}} \geq d)$ is below a given threshold $p \in [0, 1]$.

We implemented a prototype that synthesizes vector-valued supermartingales using linear and polynomial templates. The resulting constraints are solved by LP and SDP solvers, respectively. Experiments show that our method can produce nontrivial upper bounds in reasonable computation time. We also experimentally confirm that higher moments are useful in producing tighter bounds.

Our Contributions. Summarizing, the contribution of this paper is as follows.

- We extend the existing theory of ranking supermartingales from expected runtimes (i.e. the first moment) to *higher moments*. The extension has a solid foundation of order-theoretic fixed points. Moreover, its clean presentation by vector-valued supermartingales makes automated synthesis as easy as before. Our target randomized programs are rich, embracing nondeterminism and continuous distributions.
- We study how these vector-valued supermartingales (and the resulting upper bounds of higher moments) can be used to yield upper bounds of *tail probabilities of runtimes*. We identify a concentration lemma that suits this purpose. We show that higher moments indeed yield tighter bounds.
- Overall, we present a comprehensive language-based framework for overapproximating tail probabilities of runtimes of randomized programs (Fig. 2). It has been implemented, and our experiments suggest its practical use.

Organization. We give preliminaries in Sect. 2. In Sect. 3, we review the order-theoretic characterization of ordinary ranking supermartingales and present an extension to higher moments of runtimes. In Sect. 4, we discuss how to obtain an upper bound of the tail probability of runtimes. In Sect. 5, we explain an automated synthesis algorithm for our ranking supermartingales. In Sect. 6, we give experimental results. In Sect. 7, we discuss related work. We conclude and give future work in Sect. 8. Some proofs and details are deferred to the appendices available in the extended version [22].

2 Preliminaries

We present some preliminary materials, including the definition of pCFGs (we use them as a model of randomized programs) and the definition of runtime.

Given topological spaces X and Y, let $\mathcal{B}(X)$ be the set of Borel sets on X and $\mathcal{B}(X, Y)$ be the set of Borel measurable functions $X \to Y$. We assume that the set $\mathbb{R}$ of reals, a finite set L and the set $[0, \infty]$ are equipped with the usual topology, the discrete topology, and the order topology, respectively. We use the induced Borel structures for these spaces. Given a measurable space X, let $\mathcal{D}(X)$ be the set of probability measures on X. For any $\mu \in \mathcal{D}(X)$, let $\mathrm{supp}(\mu)$ be the support of μ. We write $\mathbb{E}[X]$ for the expectation of a random variable X.

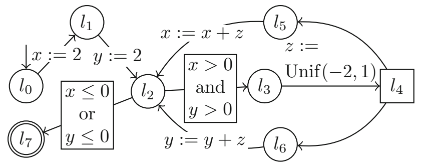
Our use of pCFGs follows recent works including [1].

Definition 2.1 (pCFG). A *probabilistic control flow graph (pCFG)* is a tuple $\Gamma = (L, V, l_{\mathrm{init}}, \boldsymbol{x}_{\mathrm{init}}, \mapsto, \mathrm{Up}, \mathrm{Pr}, G)$ that consists of the following.

– A finite set L of *locations*. It is a disjoint union of sets L_D, L_P, L_n and L_A of *deterministic*, *probabilistic*, *nondeterministic* and *assignment* locations.
– A finite set V of *program variables*.
– An *initial location* $l_{\mathrm{init}} \in L$. – An *initial valuation* $\boldsymbol{x}_{\mathrm{init}} \in \mathbb{R}^V$
– A *transition relation* $\mapsto \subseteq L \times L$ which is total (i.e. $\forall l. \exists l'. l \mapsto l'$).
– An *update function* $\mathrm{Up} : L_A \to V \times \big(\mathcal{B}(\mathbb{R}^V, \mathbb{R}) \cup \mathcal{D}(\mathbb{R}) \cup \mathcal{B}(\mathbb{R})\big)$ for assignment.
– A family $\mathrm{Pr} = (\mathrm{Pr}_l)_{l \in L_P}$ of probability distributions, where $\mathrm{Pr}_l \in \mathcal{D}(L)$, for probabilistic locations. We require that $l' \in \mathrm{supp}(\mathrm{Pr}_l)$ implies $l \mapsto l'$.
– A *guard function* $G : L_D \times L \to \mathcal{B}(\mathbb{R}^V)$ such that for each $l \in L_D$ and $\boldsymbol{x} \in \mathbb{R}^V$, there exists a unique location $l' \in L$ satisfying $l \mapsto l'$ and $\boldsymbol{x} \in G(l, l')$.

The update function can be decomposed into three functions $\mathrm{Up}_D : L_{AD} \to V \times \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, $\mathrm{Up}_P : L_{AP} \to V \times \mathcal{D}(\mathbb{R})$ and $\mathrm{Up}_N : L_{AN} \to V \times \mathcal{B}(\mathbb{R})$, under a suitable decomposition $L_A = L_{AD} \cup L_{AP} \cup L_{AN}$ of assignment locations. The elements of L_{AD}, L_{AP} and L_{AN} represent *deterministic*, *probabilistic* and *nondeterministic* assignments, respectively. See e.g. [33].

An example of a pCFG is shown on the right. It models the program in Fig. 1. The node l_4 is a nondeterministic location. $\mathrm{Unif}(-2, 1)$ is the uniform distribution on the interval $[-2, 1]$.



A *configuration* of a pCFG Γ is a pair $(l, \boldsymbol{x}) \in L \times \mathbb{R}^V$ of a location and a valuation. We regard the set $S = L \times \mathbb{R}^V$ of configurations is equipped with the product topology where L is equipped with the discrete topology. We say a configuration $(l', \boldsymbol{x}')$ is a *successor* of $(l, \boldsymbol{x})$, if $l \mapsto l'$ and the following hold.

– If $l \in L_D$, then $\boldsymbol{x}' = \boldsymbol{x}$ and $\boldsymbol{x} \in G(l, l')$. – If $l \in L_N \cup L_P$, then $\boldsymbol{x}' = \boldsymbol{x}$.
– If $l \in L_A$, then $\boldsymbol{x}' = \boldsymbol{x}(x_j \leftarrow a)$, where $\boldsymbol{x}(x_j \leftarrow a)$ denotes the vector obtained by replacing the x_j-component of $\boldsymbol{x}$ by a. Here x_j is such that $\mathrm{Up}(l) = (x_j, u)$, and a is chosen as follows: (1) $a = u(\boldsymbol{x})$ if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$; (2) $a \in \mathrm{supp}(u)$ if $u \in \mathcal{D}(\mathbb{R})$; and (3) $a \in u$ if $u \in \mathcal{B}(\mathbb{R})$.

An *invariant* of a pCFG Γ is a measurable set $I \in \mathcal{B}(S)$ such that $(l_{\mathrm{init}}, \boldsymbol{x}_{\mathrm{init}}) \in I$ and I is closed under taking successors (i.e. if $c \in I$ and c' is a successor of c then $c' \in I$). Use of invariants is a common technique in automated synthesis of supermartingales [1]: it restricts configuration spaces and thus makes the constraints on supermartingales weaker. It is also common to take an invariant as a measurable set [1]. A *run* of Γ is an infinite sequence of configurations $c_0 c_1 \dots$ such that c_0 is the initial configuration $(l_{\mathrm{init}}, \boldsymbol{x}_{\mathrm{init}})$ and c_{i+1} is a successor of c_i for each i. Let $\mathrm{Run}(\Gamma)$ be the set of runs of Γ.

A *scheduler* resolves nondeterminism: at a location in $L_N \cup L_{AN}$, it chooses a distribution of next configurations depending on the history of configurations visited so far. Given a pCFG Γ and a scheduler σ of Γ, a probability measure ν_σ^Γ on $\mathrm{Run}(\Gamma)$ is defined in the usual manner. See [22, Appendix B] for details.

Definition 2.2 (reaching time $T_C^\Gamma, T_{C,\sigma}^\Gamma$). Let Γ be a pCFG and $C \subseteq S$ be a set of configurations called a *destination*. The *reaching time* to C is a function $T_C^\Gamma : \mathrm{Run}(\Gamma) \to [0, \infty]$ defined by $(T_C^\Gamma)(c_0 c_1 \dots) = \mathrm{argmin}_{i \in \mathbb{N}}(c_i \in C)$. Fixing a scheduler σ makes T_C^Γ a random variable, since σ determines a probability measure ν_σ^Γ on $\mathrm{Run}(\Gamma)$. It is denoted by $T_{C,\sigma}^\Gamma$.

Runtimes of pCFGs are a special case of reaching times, namely to the set of terminating configurations.

The following higher moments are central to our framework. Recall that we are interested in demonic schedulers, i.e. those which make runtimes longer.

Definition 2.3 ($\mathbb{M}_{C,\sigma}^{\Gamma,k}$ and $\overline{\mathbb{M}}_C^{\Gamma,k}$). Assume the setting of Definition 2.2, and let $k \in \mathbb{N}$ and $c \in S$. We write $\mathbb{M}_{C,\sigma}^{\Gamma,k}(c)$ for the k-th moment of the reaching time of Γ from c to C under the scheduler σ, i.e. that is, $\mathbb{M}_{C,\sigma}^{\Gamma,k}(c) = \mathbb{E}[(T_{C,\sigma}^{\Gamma_c})^k] = \int (T_C^{\Gamma_c})^k \, \mathrm{d}\nu_\sigma^{\Gamma_c}$ where Γ_c is a pCFG obtained from Γ by changing the initial configuration to c. Their supremum under varying σ is denoted by $\overline{\mathbb{M}}_C^{\Gamma,k} := \sup_\sigma \mathbb{M}_{C,\sigma}^{\Gamma,k}$.

3 Ranking Supermartingale for Higher Moments

We introduce one of the main contributions in the paper, a notion of ranking supermartingale that overapproximates higher moments. It is motivated by the following observation: martingale-based reasoning about the second moment must concur with one about the first moment. We conduct a systematic theoretical extension that features an order-theoretic foundation and vector-valued supermartingales. The theory accommodates nondeterminism and continuous distributions, too. We omit some details and proofs; they are in [22, Appendix C].

The fully general theory for higher moments will be presented in Sect. 3.2; we present its restriction to the second moments in Sect. 3.1 for readability.

Prior to these, we review the existing theory of ranking supermartingales, through the lens of order-theoretic fixed points. In doing so we follow [33].

Definition 3.1 ("nexttime" operation $\overline{\mathbb{X}}$ (pre-expectation)). Given $\eta : S \to [0, \infty]$, let $\overline{\mathbb{X}}\eta : S \to [0, \infty]$ be the function defined as follows.

- If $l \in L_D$ and $\boldsymbol{x} \vDash G(l, l')$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \eta(l', \boldsymbol{x})$.
- If $l \in L_P$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \sum_{l \mapsto l'} \mathrm{Pr}_l(l')\eta(l', \boldsymbol{x})$.
- If $l \in L_N$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \sup_{l \mapsto l'} \eta(l', \boldsymbol{x})$.
- If $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$ and $l \mapsto l'$, if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \eta(l', \boldsymbol{x}(x_j \leftarrow u(\boldsymbol{x})))$; if $u \in \mathcal{D}(\mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \int_{\mathbb{R}} \eta(l', \boldsymbol{x}(x_j \leftarrow y))\,\mathrm{d}u(y)$; and if $u \in \mathcal{B}(\mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \boldsymbol{x}) = \sup_{y \in u} \eta(l', \boldsymbol{x}(x_j \leftarrow y))$.

Intuitively, $\overline{\mathbb{X}}\eta$ is the expectation of η after one transition. Nondeterminism is resolved by the maximal choice.

We define $F_1 : (S \to [0, \infty]) \to (S \to [0, \infty])$ as follows.

$$(F_1(\eta))(c) = \begin{cases} 1 + (\overline{\mathbb{X}}\eta)(c) & c \in I \setminus C \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Here "1+" accounts for time elapse)}$$

The function F_1 is an adaptation of the *Bellman operator*, a classic notion in the theory of Markov processes. A similar notion is used e.g. in [19]. The function space $(S \to [0, \infty])$ is a complete lattice structure, because $[0, \infty]$ is; moreover F_1 is easily seen to be monotone. It is not hard to see either that the expected reaching time $\overline{\mathbb{M}}_C^{\Gamma,1}$ to C coincides with the least fixed point μF_1.

The following theorem is fundamental in theoretical computer science.

Theorem 3.2 (Knaster–Tarski, [34]). *Let $(L, \leq)$ be a complete lattice and $f : L \to L$ be a monotone function. The least fixed point μf is the least prefixed point, i.e. $\mu f = \min\{l \in L \mid f(l) \leq l\}$.* □

The significance of the Knaster-Tarski theorem in verification lies in the induced proof rule: $f(l) \leq l \Rightarrow \mu f \leq l$. Instantiating to the expected reaching time $\overline{\mathbb{M}}_C^{\Gamma,1} = \mu F_1$, it means $F_1(\eta) \leq \eta \Rightarrow \overline{\mathbb{M}}_C^{\Gamma,1} \leq \eta$, i.e. an arbitrary prefixed point of F_1—which coincides with the notion of ranking supermartingale [4]—overapproximates the expected reaching time. This proves soundness of ranking supermartingales.

3.1 Ranking Supermartingales for the Second Moments

We extend ranking supermartingales to the second moments. It paves the way to a fully general theory (up to the K-th moments) in Sect. 3.2.

The key in the martingale-based reasoning of expected reaching times (i.e. first moments) was that they are characterized as the least fixed point of a function F_1. Here it is crucial that for an arbitrary random variable T, we have $\mathbb{E}[T + 1] = \mathbb{E}[T] + 1$ and therefore we can calculate $\mathbb{E}[T + 1]$ from $\mathbb{E}[T]$. However, this is not the case for second moments. As $\mathbb{E}[(T + 1)^2] = \mathbb{E}[T^2] + 2\mathbb{E}[T] + 1$, calculating the second moment requires not only $\mathbb{E}[T^2]$ but also $\mathbb{E}[T]$. This encourages us to define a vector-valued supermartingale.

Definition 3.3 (time-elapse function El_1). A function $\mathrm{El}_1 : [0, \infty]^2 \to [0, \infty]^2$ is defined by $\mathrm{El}_1(x_1, x_2) = (x_1 + 1, x_2 + 2x_1 + 1)$.

Then, an extension of F_1 for second moments can be defined as a combination of the time-elapse function El_1 and the pre-expectation $\overline{\mathbb{X}}$.

Definition 3.4 (F_2). Let I be an invariant and $C \subseteq I$ be a Borel set. We define $F_2 : (S \to [0,\infty]^2) \to (S \to [0,\infty]^2)$ by

$$(F_2(\eta))(c) = \begin{cases} (\overline{\mathbb{X}}(\mathrm{El}_1 \circ \eta))(c) & c \in I \setminus C \\ (0,0) & \text{otherwise.} \end{cases}$$

Here $\overline{\mathbb{X}}$ is applied componentwise: $(\overline{\mathbb{X}}(\eta_1,\eta_2))(c) = ((\overline{\mathbb{X}}\eta_1)(c), (\overline{\mathbb{X}}\eta_2)(c))$.

We can extend the complete lattice structure of $[0,\infty]$ to the function space $S \to [0,\infty]^2$ in a pointwise manner. It is a routine to prove that F_2 is monotone with respect to this complete lattice structure. Hence F_2 has the least fixed point. In fact, while $\overline{\mathbb{M}}_C^{\Gamma,1}$ was characterized as the least fixed point of F_1, a tuple $(\overline{\mathbb{M}}_C^{\Gamma,1}, \overline{\mathbb{M}}_C^{\Gamma,2})$ is *not* the least fixed point of F_2 (cf. Example 3.8 and Theorem 3.9). However, the least fixed point of F_2 *overapproximates* the tuple of moments.

Theorem 3.5. *For any configuration $c \in I$, $(\mu F_2)(c) \geq (\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c))$.* □

Let $T^{\Gamma}_{C,\sigma,n} = \min\{n, T^{\Gamma}_{C,\sigma}\}$. To prove the above theorem, we inductively prove

$$(F_2)^n(\bot)(c) \geq \left(\int T^{\Gamma_c}_{C,\sigma,n}\, \mathrm{d}\nu^{\Gamma_c}_{\sigma},\ \int (T^{\Gamma_c}_{C,\sigma,n})^2\, \mathrm{d}\nu^{\Gamma_c}_{\sigma}\right)$$

for each σ and n, and take the supremum. See [22, Appendix C] for more details.

Like ranking supermartingale for first moments, ranking supermartingale for second moments is defined as a prefixed point of F_2, i.e. a function η such that $\eta \geq F_2(\eta)$. However, we modify the definition for the sake of implementation.

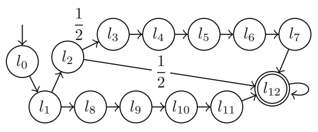
Definition 3.6 (ranking supermartingale for second moments). A ranking supermartingale for second moments is a function $\eta : S \to \mathbb{R}^2$ such that: (i) $\eta(c) \geq (\overline{\mathbb{X}}(\mathrm{El}_1 \circ \eta))(c)$ for each $c \in I \setminus C$; and (ii) $\eta(c) \geq 0$ for each $c \in I$.

Here, the time-elapse function El_1 captures a positive decrease of the ranking supermartingale. Even though we only have inequality in Theorem 3.5, we can prove the following desired property of our supermartingale notion.

Theorem 3.7. *If $\eta : S \to \mathbb{R}^2$ is a supermartingale for second moments, then $\left(\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c)\right) \leq \eta(c)$ for each $c \in I$.* □

The following example and theorem show that we cannot replace $\geq$ with $=$ in Theorem 3.5 in general, but it is possible in the absence of nondeterminism.

Example 3.8. The figure on the right shows a pCFG such that $l_2 \in L_P$ and all the other locations are in L_N, the initial location is l_0 and l_{12} is a terminating location. For the pCFG, the left-hand side of the inequality in Theorem 3.5 is $\mu F_2(l_0) = (6, 37.5)$. In contrast, if a scheduler σ takes a transition from l_1 to l_2 with probability p, $(\mathbb{M}^{\Gamma,1}_{C,\sigma}(l_0), \mathbb{M}^{\Gamma,2}_{C,\sigma}(l_0)) = \left(6 - \frac{1}{2}p, 36 - \frac{5}{2}p\right)$. Hence the right-hand side is $(\overline{\mathbb{M}}_C^{\Gamma,1}(l_0), \overline{\mathbb{M}}_C^{\Gamma,2}(l_0)) = (6, 36)$.



Theorem 3.9. *If $L_N = L_{AN} = \emptyset$, $\forall c \in I.\, (\mu F_2)(c) = (\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c))$.* □

3.2 Ranking Supermartingales for the Higher Moments

We extend the result in Sect. 3.1 to moments higher than second.

Firstly, the time-elapse function El_1 is generalized as follows.

Definition 3.10 (time-elapse function $\mathrm{El}_1^{K,k}$). For $K \in \mathbb{N}$ and $k \in \{1, \dots, K\}$, a function $\mathrm{El}_1^{K,k} : [0,\infty]^K \to [0,\infty]$ is defined by $\mathrm{El}_1^{K,k}(x_1, \dots, x_K) = 1 + \sum_{j=1}^{k} \binom{k}{j} x_j$. Here $\binom{k}{j}$ is the binomial coefficient.

Again, a monotone function F_K is defined as a combination of the time-elapse function $\mathrm{El}_1^{K,k}$ and the pre-expectation $\overline{\mathbb{X}}$.

Definition 3.11 (F_K). Let I be an invariant and $C \subseteq I$ be a Borel set. We define $F_K : (S \to [0,\infty]^K) \to (S \to [0,\infty]^K)$ by $F_K(\eta)(c) = (F_{K,1}(\eta)(c), \dots, F_{K,K}(\eta)(c))$, where $F_{K,k} : (S \to [0,\infty]^K) \to (S \to [0,\infty])$ is given by

$$(F_{K,k}(\eta))(c) = \begin{cases} (\overline{\mathbb{X}}(\mathrm{El}_1^{K,k} \circ \eta))(c) & c \in I \setminus C \\ 0 & \text{otherwise.} \end{cases}$$

As in Definition 3.6, we define a supermartingale as a prefixed point of F_K.

Definition 3.12 (ranking supermartingale for K-th moments). We define $\eta_1, \dots, \eta_K : S \to \mathbb{R}$ by $(\eta_1(c), \dots, \eta_K(c)) = \eta(c)$. A *ranking supermartingale for K-th moments* is a function $\eta : S \to \mathbb{R}^K$ such that for each k, (i) $\eta_k(c) \geq (\overline{\mathbb{X}}(\mathrm{El}_1^{K,k} \circ \eta_k))(c)$ for each $c \in I \setminus C$; and (ii) $\eta_k(c) \geq 0$ for each $c \in I$.

For higher moments, we can prove an analogous result to Theorem 3.7.

Theorem 3.13. *If η is a supermartingale for K-th moments, then for each $c \in I$, $(\overline{\mathbb{M}}_C^{\Gamma,1}(c), \dots, \overline{\mathbb{M}}_C^{\Gamma,K}(c)) \leq \eta(c)$.* □

4 From Moments to Tail Probabilities

We discuss how to obtain upper bounds of tail probabilities of runtimes from upper bounds of higher moments of runtimes. Combined with the result in Sect. 3, it induces a martingale-based method for overapproximating tail probabilities.

We use a concentration inequality. There are many choices of concentration inequalities (see e.g. [3]), and we use a variant of Markov's inequality. We prove that the concentration inequality is not only sound but also complete in a sense.

Formally, our goal is to calculate is an upper bound of $\Pr(T_{C,\sigma}^{\Gamma} \geq d)$ for a given deadline $d > 0$, under the assumption that we know upper bounds $u_1, \dots, u_K$ of moments $\mathbb{E}[T_{C,\sigma}^{\Gamma}], \dots, \mathbb{E}[(T_{C,\sigma}^{\Gamma})^K]$. In other words, we want to overapproximate $\sup_\mu \mu([d,\infty])$ where μ ranges over the set of probability measures on $[0,\infty]$ satisfying $\left(\int x \,\mathrm{d}\mu(x), \dots, \int x^K \,\mathrm{d}\mu(x)\right) \leq (u_1, \dots, u_K)$.

To answer this problem, we use a generalized form of Markov's inequality.

Proposition 4.1 (see e.g. [3, §2.1]). *Let X be a real-valued random variable and ϕ be a nondecreasing and nonnegative function. For any $d \in \mathbb{R}$ with $\phi(d) > 0$,*

$$\Pr(X \geq d) \leq \frac{\mathbb{E}[\phi(X)]}{\phi(d)}.$$

□

By letting $\phi(x) = x^k$ in Proposition 4.1, we obtain the following inequality. It gives an upper bound of the tail probability that is "tight."

Proposition 4.2. *Let X be a nonnegative random variable. Assume $\mathbb{E}[X^k] \leq u_k$ for each $k \in \{0, \dots, K\}$. Then, for any $d > 0$,*

$$\Pr(X \geq d) \leq \min_{0 \leq k \leq K} \frac{u_k}{d^k}. \tag{1}$$

Moreover, this upper bound is tight: for any $d > 0$, there exists a probability measure such that the above equation holds.

Proof. The former part is immediate from Proposition 4.1. For the latter part, consider $\mu = p\delta_d + (1-p)\delta_0$ where δ_x is the Dirac measure at x and p is the value of the right-hand side of (1). □

By combining Theorem 3.13 with Proposition 4.2, we obtain the following corollary. We can use it for overapproximating tail probabilities.

Corollary 4.3. *Let $\eta : S \to \mathbb{R}^K$ be a ranking supermartingale for K-th moments. For each scheduler σ and a deadline $d > 0$,*

$$\Pr(T^{\Gamma}_{C,\sigma} \geq d) \leq \min_{0 \leq k \leq K} \frac{\eta_k(l_{\text{init}}, \boldsymbol{x}_{\text{init}})}{d^k}. \tag{2}$$

Here $\eta_0, \dots, \eta_K$ are defined by $\eta_0(c) = 1$ and $\eta(c) = (\eta_1(c), \dots, \eta_K(c))$. □

Note that if $K = 1$, Corollary 4.3 is essentially the same as [5, Thm 4]. Note also that for each K there exists $d > 0$ such that $\frac{\eta_K(l_{\text{init}}, \boldsymbol{x}_{\text{init}})}{d^K} = \min_{0 \leq k \leq K} \frac{\eta_k(l_{\text{init}}, \boldsymbol{x}_{\text{init}})}{d^k}$. Hence higher moments become useful in overapproximating tail probabilities as d gets large. Later in Sect. 6, we demonstrate this fact experimentally.

5 Template-Based Synthesis Algorithm

We discuss an automated synthesis algorithm that calculates an upper bound for the k-th moment of the runtime of a pCFG using a supermartingale in Definitions 3.6 or 3.12. It takes a pCFG Γ, an invariant I, a set $C \subseteq I$ of configurations, and a natural number K as input and outputs an upper bound of K-th moment.

Our algorithm is adapted from existing template-based algorithms for synthesizing a ranking supermartingale (for first moments) [4,6,7]. It fixes a linear or polynomial template with unknown coefficients for a supermartingale and using numerical methods like linear programming (LP) or semidefinite programming

(SDP), calculate a valuation of the unknown coefficients so that the axioms of ranking supermartingale for K-th moments are satisfied.

We hereby briefly explain the algorithms. See [22, Appendix D] for details.

Linear Template. Our linear template-based algorithm is adapted from [4,7]. We should assume that Γ, I and C are all "linear" in the sense that expressions appearing in Γ are all linear and I and C are represented by linear inequalities. To deal with assignments from a distribution like $x := \mathrm{Norm}(0,1)$, we also assume that expected values of distributions appearing in Γ are known.

The algorithm first fixes a template for a supermartingale: for each location l, it fixes a K-tuple $\left(\sum_{j=1}^{|V|} a^l_{j,1} x_j + b^l_1, \ldots, \sum_{j}^{|V|} a^l_{j,K} x_j + b^l_K\right)$ of linear formulas. Here each $a^l_{j,i}$ and b^l_i are unknown variables called *parameters*. The algorithm next collects conditions on the parameters so that the tuples constitute a ranking supermartingale for K-th moments. It results in a conjunction of formulas of a form $\varphi_1 \geq 0 \wedge \cdots \wedge \varphi_m \geq 0 \Rightarrow \psi \geq 0$. Here $\varphi_1, \ldots, \varphi_m$ are linear formulas without parameters and ψ is a linear formula where parameters linearly appear in the coefficients. By Farkas' lemma (see e.g. [29, Cor 7.1h]) we can turn such formulas into linear inequalities over parameters by adding new variables. Its feasibility is efficiently solvable with an LP solver. We naturally wish to minimize an upper bound of the K-th moment, i.e. the last component of $\eta(l_{\mathrm{init}}, \boldsymbol{x}_{\mathrm{init}})$. We can minimize it by setting it to the objective function of the LP problem.

Polynomial Template. The polynomial template-based algorithm is based on [6]. This time, Γ, I and C can be "polynomial." To deal with assignments of distributions, we assume that the n-th moments of distributions in Γ are easily calculated for each $n \in \mathbb{N}$. It is similar to the linear template-based one.

It first fixes a polynomial template for a supermartingale, i.e. it assigns each location l a K-tuple of polynomial expressions with unknown coefficients. Likewise the linear template-based algorithm, the algorithm reduces the axioms of supermartingale for higher moments to a conjunction of formulas of a form $\varphi_1 \geq 0 \wedge \cdots \wedge \varphi_m \geq 0 \Rightarrow \psi \geq 0$. This time, each φ_i is a polynomial formula without parameters and ψ is a polynomial formula whose coefficients are *linear* formula over the parameters. In the polynomial case, a conjunction of such formula is reduced to an SDP problem using a theorem called Positivstellensatz (we used a variant called Schmüdgen's Positivstellensatz [28]). We solve the resulting problem using an SDP solver setting $\eta(l_{\mathrm{init}}, \boldsymbol{x}_{\mathrm{init}})$ as the objective function.

6 Experiments

We implemented two programs in OCaml to synthesize a supermartingale based on (a) a linear template and (b) a polynomial template. The programs translate a given randomized program to a pCFG and output an LP or SDP problem as described in Sect. 5. An invariant I and a terminal configuration C for the input program are specified manually. See e.g. [20] for automatic synthesis of an invariant. For linear templates, we have used GLPK (v4.65) [12] as an LP solver. For polynomial templates, we have used SOSTOOLS (v3.03) [31] (a sums of squares optimization tool that internally uses an SDP solver) on Matlab (R2018b). We used SDPT3 (v4.0) [30] as an SDP solver. The experiments were carried out on a Surface Pro 4 with an Intel Core i5-6300U (2.40 GHz) and 8 GB RAM. We tested our implementation for the following two programs and their variants, which were also used in the literature [7,19]. Their code is in [22, Appendix E].

Coupon collector's problem. A probabilistic model of collecting coupons enclosed in cereal boxes. There exist n types of coupons, and one repeatedly buy cereal boxes until all the types of coupons are collected. We consider two cases: (1-1) $n = 2$ and (1-2) $n = 4$. We tested the linear template program for them.

Random walk. We used three variants of 1-dimensional random walks: (2-1) integer-valued one, (2-2) real-valued one with assignments from continuous distributions, (2-3) with adversarial nondeterminism; and two variants of 2-dimensional random walks (2-4) and (2-5) with assignments from continuous distributions and adversarial nondeterminism. We tested both the linear and the polynomial template programs for these examples.

Experimental results. We measured execution times needed for Step 1 in Fig. 2. The results are in Table 1. Execution times are less than 0.2 s for linear template programs and several minutes for polynomial template programs. Upper bounds of tail probabilities obtained from Proposition 4.2 are in Fig. 3.

We can see that our method is applicable even with nondeterministic branching ((2-3), (2-4) and (2-5)) or assignments from continuous distributions ((2-2), (2-4) and (2-5)). We can use a linear template for bounding higher moments as long as there exists a supermartingale for higher moments representable by linear expressions ((1-1), (1-2) and (2-3)). In contrast, for (2-1), (2-2) and (2-4), only a polynomial template program found a supermartingale for second moments.

It is expectable that the polynomial template program gives a better bound than the linear one because a polynomial template is more expressive than a linear one. However, it did not hold for some test cases, probably because of numerical errors of the SDP solver. For example, (2-1) has a supermartingale for third moments that can be checked by a hand calculation, but the SDP solver returned "infeasible" in the polynomial template program. It appears that our program fails when large numbers are involved (e.g. the third moments of (2-1), (2-2) and (2-3)). We have also tested a variant of (2-1) where the initial position is multiplied by 10000. Then the SDP solver returned "infeasible" in the polynomial template program while the linear template program returns a nontrivial bound. Hence it seems that numerical errors are likely to occur to the polynomial template program when large numbers are involved.

Figure 3 shows that the bigger the deadline d is, the more useful higher moments become (cf. a remark just after Corollary 4.3). For example, in (1-2), an upper bound of $\Pr(T^{\Gamma}_{C,\sigma} \geq 100)$ calculated from the upper bound of the first moment is 0.680 while that of the fifth moment is 0.105.

To show the merit of our method compared with sampling-based methods, we calculated a tail probability bound for a variant of (2-2) (shown in Fig. 4 on

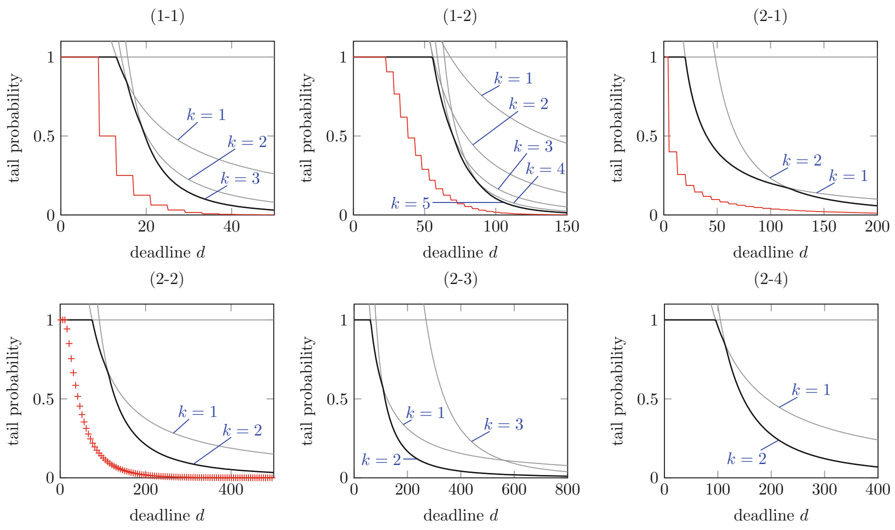


Fig. 3. Upper bounds of the tail probabilities (except (2-5)). Each gray line is the value of $\frac{u_k}{d^k}$ where u_k is the best upper bound in Table 1 of k-th moments and d is a deadline. Each black line is the minimum of gray lines, i.e. the upper bound by Proposition 4.2. The red lines in (1-1), (1-2) and (2-1) show the true tail probabilities calculated analytically. The red points in (2-2) show tail probabilities calculated by Monte Carlo sampling where the number of trials is 100000000. We did not calculate the true tail probabilities nor approximate them for (2-4) and (2-5) because these examples seem difficult to do so due to nondeterminism. (Color figure online)

Table 1. Upper bounds of the moments of runtimes. "-" indicates that the LP or SDP solver returned "infeasible". The "degree" column shows the degree of the polynomial template used in the experiments.

		(a) linear template		(b) polynominal template		
	moment	upper bound	time (s)	upper bound	time (s)	degree
(1-1)	1st	13	0.012			
	2nd	201	0.019			
	3rd	3829	0.023			
(1-2)	1st	68	0.024			
	2nd	3124	0.054			
	3rd	171932	0.089			
	4th	12049876	0.126			
	5th	1048131068	0.191			
(2-1)	1st	20	0.024	20.0	24.980	2
	2nd	-	0.013	2320.0	37.609	2
	3rd	-	0.017	-	30.932	3
(2-2)	1st	75	0.009	75.0	33.372	2
	2nd	-	0.014	8375.0	73.514	2
	3rd	-	0.021	-	170.416	3
(2-3)	1st	62	0.020	62.0	40.746	2
	2nd	28605.4	0.038	6710.0	97.156	2
	3rd	19567043.36	0.057	-	35.427	3
(2-4)	1st	96	0.020	95.95	157.748	2
	2nd	-	0.029	10944.0	361.957	2
(2-5)	1st	90	0.022	-	143.055	2
	2nd	-	0.042	-	327.202	2

```
x := 200000000;
while true do
  if prob(0.7) then
    z := Unif(0,1);
    x := x - z
  else
    z := Unif(0,1);
    x := x + z
  fi;
  refute (x < 0)
od
```

Fig. 4. A variant of (2-2).

p. 12) with a deadline $d = 10^{11}$. Because of its very long expected runtime, a sampling-based method would not work for it. In contrast, the linear template-based program gave an upper bound $\Pr(T^{\Gamma}_{C,\sigma} \geq 10^{11}) \leq 5000000025/10^{11} \approx 0.05$ in almost the same execution time as (2-2) (< 0.02 s).

7 Related Work

Martingale-Based Analysis of Randomized Programs. Martingale-based methods are widely studied for the termination analysis of randomized programs. One of the first is *ranking supermartingales*, introduced in [4] for proving almost sure termination. The theory of ranking supermartingales has since been extended actively: accommodating nondeterminism [1,6,7,11], syntax-oriented composition of supermartingales [11], proving properties beyond termination/reachability [13], and so on. Automated template-based synthesis of supermartingales by constraint solving has been pursued, too [1,4,6,7].

Other martingale-based methods that are fundamentally different from ranking supermartingales have been devised, too. They include: different notions of *repulsing supermartingales* for refuting termination (in [8,33]; also studied in control theory [32]); and *multiply-scaled submartingales* for underapproximating reachability probabilities [33,36]. See [33] for an overview.

In the literature on martingale-based methods, the one closest to this work is [5]. Among its contribution is the analysis of tail probabilities. It is done by either of the following combinations: (1) *difference-bounded* ranking supermartingales and the corresponding Azuma's concentration inequality; and (2) (not necessarily difference-bounded) ranking supermartingales and Markov's concentration inequality. When we compare these two methods with ours, the first method requires repeated martingale synthesis for different parameter values, which can pose a performance challenge. The second method corresponds to the restriction of our method to the first moment; recall that we showed the advantage of using higher moments, theoretically (Sect. 4) and experimentally (Sect. 6). See [22, Appendix F.1] for detailed discussions. Implementation is lacking in [5], too.

We use Markov's inequality to calculate an upper bound of $\Pr(T_{\text{run}} \geq d)$ from a ranking supermartingale. In [7], Hoeffding's and Bernstein's inequalities are used for the same purpose. As the upper bounds obtained by these inequalities are exponentially decreasing with respect to d, they are asymptotically tighter than our bound obtained by Markov's inequality, assuming that we use the same ranking supermartingale. However, Hoeffding's and Bernstein's inequalities are applicable to limited classes of ranking supermartingales (so-called difference-bounded and incremental ones, respectively). There exists a randomized program whose tail probability for runtimes is decreasing only polynomially (not exponentially, see [22, Appendix G]); this witnesses that there are cases where the methods in [7] do not apply but ours can.

The work [1] is also close to ours in that their supermartingales are vector-valued. The difference is in the orders: in [1] they use the *lexicographic* order between vectors, and they aim to prove almost sure termination. In contrast, we use the *pointwise* order between vectors, for overapproximating higher moments.

The Predicate-Transformer Approach to Runtime Analysis. In the runtime/termination analysis of randomized programs, another principal line of work uses *predicate transformers* [2,17,19], following the precedent works on probabilistic predicate transformers such as [21,25]. In fact, from the mathematical point of view, the main construct for witnessing runtime/termination in those predicate transformer calculi (called *invariants*, see e.g. in [19]) is essentially the same thing as ranking supermartingales. Therefore the difference between the martingale-based and predicate-transformer approaches is mostly the matter of presentation—the predicate-transformer approach is more closely tied to program syntax and has a stronger deductive flavor. It also seems that there is less work on automated synthesis in the predicate-transformer approach.

In the predicate-transformer approach, the work [17] is the closest to ours, in that it studies *variance* of runtimes of randomized programs. The main differences are as follows: (1) computing tail probabilities is not pursued [17]; (2) their extension from expected runtimes to variance involves an additional variable τ, which poses a challenge in automated synthesis as well as in generalization to even higher moments; and (3) they do not pursue automated analysis. See Appendix F.2 of the extended version [22] for further details.

Higher Moments of Runtimes. Computing and using higher moments of runtimes of probabilistic systems—generalizing randomized programs—has been pursued before. In [9], computing moments of runtimes of *finite-state* Markov chains is reduced to a certain linear equation. In the study of randomized algorithms, the survey [10] collects a number of methods, among which are some tail probability bounds using higher moments. Unlike ours, none of these methods are language-based static ones. They do not allow automated analysis.

Other Potential Approaches to Tail Probabilities. We discuss potential approaches to estimating tail probabilities, other than the martingale-based one.

Sampling is widely employed for approximating behaviors of probabilistic systems; especially so in the field of probabilistic programming languages, since exact symbolic reasoning is hard in presence of conditioning. See e.g. [35]. We also used sampling to estimate tail probabilities in (2-2), Fig. 3. The main advantages of our current approach over sampling are threefold: (1) our upper bounds come with a mathematical guarantee, while the sampling bounds can always be erroneous; (2) it requires ingenuity to sample programs with nondeterminism; and (3) programs whose execution can take millions of years can still be analyzed by our method in a reasonable time, without executing them. The latter advantage is shared by static, language-based analysis methods in general; see e.g. [2].

Another potential method is probabilistic model checkers such as PRISM [23]. Their algorithms are usually only applicable to finite-state models, and thus not to randomized programs in general. Nevertheless, fixing a deadline d can make the reachable part $S_{\leq d}$ of the configuration space S finite, opening up the possibility of use of model checkers. It is an open question how to do so precisely, and the following challenges are foreseen: (1) if the program contains continuous distributions, the reachable part $S_{\leq d}$ becomes infinite; (2) even if $S_{\leq d}$ is finite, one has to repeat (supposedly expensive) runs of a model checker for each choice of d. In contrast, in our method, an upper bound for the tail probability $\Pr(T_{\mathrm{run}} \geq d)$ is symbolically expressed as a function of d (Proposition 4.2). Therefore, estimating tail probabilities for varying d is computationally cheap.

8 Conclusions and Future Work

We provided a technique to obtain an upper bound of the tail probability of runtimes given a randomized algorithm and a deadline. We first extended the ordinary ranking supermartingale notion using the order-theoretic characterization so that it can calculate upper bounds of higher moments of runtimes for randomized programs. Then by using a suitable concentration inequality, we introduced a method to calculate an upper bound of tail probabilities from upper bounds of higher moments. Our method is not only sound but also complete in a sense. Our method was obtained by combining our supermartingale and the concentration inequality. We also implemented an automated synthesis algorithm and demonstrated the applicability of our framework.

Future Work. Example 3.8 shows that our supermartingale is not complete: it sometimes fails to give a tight bound for higher moments. Studying and improving the incompleteness is one possible direction of future work. For example, the following questions would be interesting: Can bounds given by our supermartingale be arbitrarily bad? Can we remedy the completeness by restricting the type of nondeterminism? Can we define a complete supermartingale?

Making our current method compositional is another direction of future research. Use of continuations, as in [18], can be a technical solution.

We are also interested in improving the implementation. The polynomial template program failed to give an upper bound for higher moments because of numerical errors (see Sect. 6). We wish to remedy this situation. There exist several studies for using numerical solvers for verification without affected by numerical errors [14–16, 26, 27]. We might make use of these works for improvements.

Acknowledgement. We thank the anonymous referees for useful comments. The authors are supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), the JSPS-INRIA Bilateral Joint Research Project “CRECOGI,” and JSPS KAKENHI Grant No. 15KT0012 & 15K11984. Natsuki Urabe is supported by JSPS KAKENHI Grant No. 16J08157.

References

1. Agrawal, S., Chatterjee, K., Novotný, P.: Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. PACMPL **2**(POPL), 34:1–34:32 (2018)
2. Batz, K., Kaminski, B.L., Katoen, J.-P., Matheja, C.: How long, O Bayesian network, will I sample thee? - A program analysis perspective on expected sampling times. In: Ahmed, A. (ed.) ESOP 2018. LNCS, vol. 10801, pp. 186–213. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89884-1_7
3. Boucheron, S., Lugosi, G., Massart, P.: Concentration Inequalities: A Nonasymptotic Theory of Independence. Oxford University Press, Oxford (2013)
4. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 511–526. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_34
5. Chatterjee, K., Fu, H.: Termination of nondeterministic recursive probabilistic programs. CoRR, abs/1701.02944 (2017)
6. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through Positivstellensatz's. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_1
7. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. ACM Trans. Program. Lang. Syst. **40**(2), 7:1–7:45 (2018)
8. Chatterjee, K., Novotný, P., Zikelic, D.: Stochastic invariants for probabilistic termination. In: POPL, pp. 145–160. ACM (2017)
9. Dayar, T., Akar, N.: Computing moments of first passage times to a subset of states in markov chains. SIAM J. Matrix Anal. Appl. **27**(2), 396–412 (2005)
10. Doerr, B.: Probabilistic tools for the analysis of randomized optimization heuristics. CoRR, abs/1801.06733 (2018)
11. Ferrer Fioriti, L.M., Hermanns, H.: Probabilistic termination: soundness, completeness, and compositionality. In: POPL, pp. 489–501. ACM (2015)
12. The GNU linear programming kit. https://www.gnu.org/software/glpk/
13. Jagtap, P., Soudjani, S., Zamani, M.: Temporal logic verification of stochastic systems using barrier certificates. In: Lahiri and Wang [24], pp. 177–193
14. Jansson, C.: Termination and verification for ill-posed semidefinite programming problems. Optimization Online (2005)
15. Jansson, C.: VSDP: a MATLAB software package for verified semidefinite programming. In: NOLTA, pp. 327–330 (2006)
16. Jansson, C., Chaykin, D., Keil, C.: Rigorous error bounds for the optimal value in semidefinite programming. SIAM J. Numer. Anal. **46**(1), 180–200 (2007)
17. Kaminski, B.L., Katoen, J.-P., Matheja, C.: Inferring covariances for probabilistic programs. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 191–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_14
18. Kaminski, B.L., Katoen, J.-P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run–times of probabilistic programs. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 364–389. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49498-1_15
19. Kaminski, B.L., Katoen, J.-P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. J. ACM **65**(5), 30:1–30:68 (2018)

20. Katoen, J.-P., McIver, A., Meinicke, L., Morgan, C.C.: Linear-invariant generation for probabilistic programs: automated support for proof-based methods. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 390–406. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15769-1_24
21. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
22. Kura, S., Urabe, N., Hasuo, I.: Tail probabilities for randomized program runtimes via martingales for higher moments. CoRR, abs/1811.06779 (2018)
23. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
24. Lahiri, S.K., Wang, C. (eds.): ATVA 2018. LNCS, vol. 11138. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4
25. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Trans. Program. Lang. Syst. **18**(3), 325–353 (1996)
26. Roux, P., Iguernlala, M., Conchon, S.: A non-linear arithmetic procedure for control-command software verification. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 132–151. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_8
27. Roux, P., Voronin, Y.-L., Sankaranarayanan, S.: Validating numerical semidefinite programming solvers for polynomial invariants. Form. Methods Syst. Des. **53**(2), 286–312 (2018)
28. Schmüdgen, K.: The k-moment problem for compact semi-algebraic sets. Math. Ann. **289**(1), 203–206 (1991)
29. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1986)
30. SDPT3. http://www.math.nus.edu.sg/~mattohkc/SDPT3.html
31. SOSTOOLS. http://sysos.eng.ox.ac.uk/sostools/
32. Steinhardt, J., Tedrake, R.: Finite-time regional verification of stochastic non-linear systems. Int. J. Robot. Res. **31**(7), 901–923 (2012)
33. Takisaka, T., Oyabu, Y., Urabe, N., Hasuo, I.: Ranking and repulsing supermartingales for reachability in probabilistic programs. In: Lahiri and Wang [24], pp. 476–493
34. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**, 285–309 (1955)
35. Tolpin, D., van de Meent, J.-W., Yang, H., Wood, F.D.: Design and implementation of probabilistic programming language anglican. In: IFL, pp. 6:1–6:12. ACM (2016)
36. Urabe, N., Hara, M., Hasuo, I.: Categorical liveness checking by corecursive algebras. In: Proceedings of LICS 2017, pp. 1–12. IEEE Computer Society (2017)