Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs

SHESHANSH AGRAWAL, IIT Bombay, India

KRISHNENDU CHATTERJEE, IST Austria, Austria

PETR NOVOTNÝ, IST Austria, Austria

Probabilistic programs extend classical imperative programs with real-valued random variables and random branching. The most basic liveness property for such programs is the termination property. The qualitative (aka almost-sure) termination problem asks whether a given program program terminates with probability 1. While ranking functions provide a sound and complete method for non-probabilistic programs, the extension of them to probabilistic programs is achieved via ranking supermartingales (RSMs). Although deep theoretical results have been established about RSMs, their application to probabilistic programs with nondeterminism has been limited only to programs of restricted control-flow structure. For non-probabilistic programs, lexicographic ranking functions provide a compositional and practical approach for termination analysis of real-world programs. In this work we introduce lexicographic RSMs and show that they present a sound method for almost-sure termination of probabilistic programs with nondeterminism. We show that lexicographic RSMs provide a tool for compositional reasoning about almost-sure termination, and for probabilistic programs with linear arithmetic they can be synthesized efficiently (in polynomial time). We also show that with additional restrictions even asymptotic bounds on expected termination time can be obtained through lexicographic RSMs. Finally, we present experimental results on benchmarks adapted from previous work to demonstrate the effectiveness of our approach.

CCS Concepts: • **Theory of computation** → **Logic and verification**; **Program verification**; **Program analysis**; Probabilistic computation; Random walks and Markov chains; • **Mathematics of computing** → *Stochastic processes*;

Additional Key Words and Phrases: probabilistic programs, termination, martingales, termination time

Authors' addresses: Sheshansh Agrawal, IIT Bombay, Mumbai, India, sheshansh@cse.iitb.ac.in; Krishnendu Chatterjee, IST Austria, Klosterneuburg, Austria, Krishnendu.Chatterjee@ist.ac.at; Petr Novotný, IST Austria, Klosterneuburg, Austria, petr.novotny@ist.ac.at.

1 INTRODUCTION

Probabilistic programs with nondeterminism. Randomness plays a fundamental role in many areas across science, and in computer science in particular. In applications such as stochastic network protocols [Baier and Katoen 2008; Kwiatkowska et al. 2011], randomized algorithms [Dubhashi and Panconesi 2009; Motwani and Raghavan 1995], security [Barthe et al. 2016b,c] machine learning [Ghahramani 2015; Kaelbling et al. 1996], the probabilistic behavior must be considered to faithfully model the underlying dynamic system. The extension of classical imperative programs with *random value generators*, that produce random values according to some desired probability distribution, naturally gives rise to probabilistic programs. Along with probability, nondeterminism also plays a crucial role. In particular in program analysis, for effective analysis of large programs, all variables cannot be considered, and abstraction ignores some variables, and the worst-case analysis is represented by adversarial nondeterminism. Hence, probabilistic programs with nondeterminism have become an active and important research focus in program analysis.

Termination problem. In static analysis of programs the most basic, as well most important, liveness property is the *termination* problem. While for non-probabilistic programs the termination question asks whether an input program *always* terminates, for probabilistic programs the termination questions must account for the probabilistic behaviors. The most basic and fundamental extensions of the termination problem for probabilistic programs are:

(1) *Almost-sure termination.* The *almost-sure termination* problem asks whether the program terminates with probability 1.
(2) *Positive termination.* The *positive termination* problem asks whether the expected termination time is finite. A related quantitative generalization of the positive termination question is to obtain asymptotic bounds on the expected termination time.

While the positive termination implies almost-sure termination, the converse is not true (e.g., see Example 4.9).

Ranking functions and ranking supermartingales (RSMs). The key technique that applies for liveness analysis of non-probabilistic programs is the notion of *ranking functions*, which provides a sound and complete method for termination of non-probabilistic programs [Floyd 1967]. There exists a wide variety of approaches for construction of ranking functions for non-probabilistic programs [Bradley et al. 2005a; Colón and Sipma 2001; Podelski and Rybalchenko 2004a; Sohn and Gelder 1991]. The generalization of ranking functions to probabilistic programs is achieved through *ranking supermartingales (RSMs)* [Chakarov and Sankaranarayanan 2013; Chatterjee and Fu 2017; Fioriti and Hermanns 2015]. The ranking supermartingales provide a powerful and automated approach for termination analysis of probabilistic programs, and algorithmic approaches for special cases such as linear and polynomial RSMs have also been considered [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016a,b, 2017].

Limitations of existing approaches. An impressive set of theoretical results related to RSMs has been established [Chakarov and Sankaranarayanan 2013; Chatterjee and Fu 2017; Chatterjee et al. 2016a,b, 2017; Fioriti and Hermanns 2015]. However, previous martingale-based methods for probabilistic programs with nondeterminism [Chatterjee and Fu 2017; Chatterjee et al. 2016a,b, 2017] are only applicable to programs of rather restricted control-flow structure. The key reason can be understood as follows: already in the world of non-probabilistic programs we know that while ranking functions are sound and complete, they do not necessarily provide an efficient approach. This is because to prove termination, a witness in the form of a ranking function has to be computed: to do this

automatically, ranking functions of a restricted form (such as linear ranking functions) have to be considered. But 1-dimensional ranking functions of a restricted type might be too weak to prove termination of programs of more complex control-flow structure, such as programs with multiple nested loops or programs where there are several possible paths through some loop, each path exhibiting a profoundly different behaviour. To remedy this problem in the world of non-probabilistic programs, the notion of lexicographic ranking functions has been widely studied [Alias et al. 2010; Bradley et al. 2005a; Brockschmidt et al. 2016; Cook et al. 2013; Gonnord et al. 2015].

Algorithmic approaches for linear lexicographic ranking functions allow the automated termination analysis to be applied to real-world non-probabilistic programs (after abstraction). However, the theoretical foundations for lexicographic termination arguments are completely missing for probabilistic programs, an issue which we address in this work.

Our contributions. In this paper we present several contributions to termination proving for probabilistic programs, chief of which is defining a notion of a lexicographic ranking supermartingale and proving that lexicographic RSMs are sound for proving almost-sure termination. We describe our main contributions below:

(1) *Theoretical foundations.* First, we introduce the notion of lexicographic RSMs, and show that such supermartingales ensure almost-sure termination (Theorem 3.4 in Section 3). The key contribution is a purely mathematical result that introduces a new concept and proves that it is sound for proving almost-sure termination. The result is independent of any probabilistic program. While the concept of lexicographic RSMs might look as a natural extension of classical lexicographic ranking functions, the proof of soundness is rather delicate and deals with several intricacies that are not present in non-probabilistic world or when dealing with 1-dimensional RSMs. Based on the mathematical result, we show that for probabilistic programs with nondeterminism the existence of a lexicographic RSM with respect to an invariant ensures almost-sure termination (Theorem 4.8 in Section 4). We also show that lexicographic RSMs are capable of proving almost-sure termination of programs with infinite expected termination time (Example 4.9).

(2) *Compositionality.* Second, we study the compositional properties of lexicographic RSMs. A key limitation of the previous approaches related to compositional RSMs [Fioriti and Hermanns 2015] is that it imposes a technical *uniform integrability* condition, which is hard to reason about automatically. We show (in Section 6) how lexicographic RSMs present an easy-to-automatize compositional approach for almost-sure termination of probabilistic programs.

(3) *Algorithm.* We then consider algorithms for synthesis of lexicographic RSMs, and for efficient algorithms we consider nondeterministic probabilistic programs that are *affine* (i.e., the arithmetic operations are linear). We present a polynomial-time algorithm for synthesis of lexicographic RSMs for affine programs (Theorem 5.1), which generalizes an existing termination-proving algorithm for non-probabilistic programs [Alias et al. 2010].

(4) *Asymptotic bounds.* In general, the existence of lexicographic RSMs does not imply positive termination. In other words, we present an example (Example 4.9) where a lexicographic RSM exists ensuring almost-sure termination, yet the expected termination time is infinite. We then present a natural restriction under which the lexicographic RSMs not only imply positive termination, but even asymptotic bounds on the expected termination time can be derived from them (Theorem 7.2 and Corollary 7.3). Moreover, the asymptotic bounds we obtain establish polynomial bounds on the expected termination time (see Remark 5).

(5) *Experimental results.* We present experimental results to demonstrate the applicability of our approach. We consider the benchmarks of non-probabilistic programs from [Alias et al. 2010],

which include abstractions of classical algorithms (e.g. sorting ones), and where lexicographic ranking functions were applied for termination analysis. We extend these benchmarks with probabilistic statements and apply lexicographic RSMs to these programs. While the benchmarks used are still rather small (10-100 LOC), proving their almost-sure termination is beyond the capabilities of one-dimensional RSMs, and hence they are suitable for testing the termination-proving power of lexicographic RSMs. Our results show that our approach works well on top of existing non-probabilistic termination-proving infrastructure pioneered in aforementioned works. We also experiment on larger synthetic programs to obtain data on scalability of our algorithm.

2 PRELIMINARIES

We use a notation standard in the field of probabilistic program analysis [Chatterjee et al. 2017].

2.1 Basic Notions

For a set A we denote by $|A|$ the cardinality of A. We denote by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of all positive integers, non-negative integers, integers, and real numbers, respectively. We assume basic knowledge of matrix calculus. We use boldface notation for vectors, e.g. $\mathbf{x}$, $\mathbf{y}$, etc., and we denote an i-th component of a vector $\mathbf{x}$ by $\mathbf{x}[i]$. We identify 1-dimensional vectors with numbers. For an n-dimensional vector $\mathbf{x}$, index $1 \leq i \leq n$, and number a we denote by $\mathbf{x}(i \leftarrow a)$ a vector $\mathbf{y}$ such that $\mathbf{y}[i] = a$ and $\mathbf{y}[j] = \mathbf{x}[j]$ for all $1 \leq j \leq n$, $j \neq i$. For comparison of vectors (e.g. as in $\mathbf{x} \leq \mathbf{y}$), we consider componentwise comparison. For comparing functions f, g with the same domains, we write $f \leq g$ if $f(x) \leq g(x)$ for all x in the domain. Throughout the paper we fix a countable set of variables $\mathcal{V}$. We consider some arbitrary but fixed linear order on the set of all variables, hence we write $\mathcal{V} = \{x_1, x_2, x_3, \dots\}$.

2.2 Syntax of Probabilistic Programs

In this subsection we define the form of probabilistic programs that we consider in our analysis. We consider two classes of probabilistic programs: general probabilistic programs (PPs) with arbitrary (measurable) expressions and their subclass, affine probabilistic programs (Apps) where all expressions are restricted to be affine (see below for a precise definition). The reason for this dual view is that our work also has two main points of focus: a theoretical one, where we introduce new concepts that can be used to prove properties of general probabilistic programs; and an algorithmic one, where we aim to prove properties of probabilistic programs automatically, using the aforementioned concepts. As already testified in the non-probabilistic world, programs that contain only affine expressions allow for more efficient automation of the analysis and at the same time, due to the presence of non-determinism, they can be used to form sound abstractions of programs with non-linear arithmetic. Hence, we consider general programs when providing our theoretical results and Apps when presenting the automation of our techniques.

Expressions. An *expression* over the set of variables $\{x_1, \dots, x_n\}$ is an expression in the standard programming-language sense, i.e. a formula built in a finite number of steps from constants, variables $x_1, \dots, x_n$, and numerical operators from some fixed finite set. Each expression E over $\{x_1, \dots, x_n\}$ determines a function which for each m-dimensional vector $\mathbf{x}$, where $m \geq n$, returns a number resulting from substituting each x_i in E by $\mathbf{x}[i]$. Slightly abusing our notation, we denote this function also by E and the value of this function on argument $\mathbf{x}$ by $E(\mathbf{x})$. We do not a priori fix

a concrete set of operators that can be used to form expressions. However, in order to ensure that semantics of probabilistic programs with real-valued variables is defined correctly, we impose the following two conditions on the set of expressions used in each program: (1) For each expression E over variables $\{x_1, \ldots, x_n\}$ and each n-dimensional vector $\mathbf{x}$ the value $E(\mathbf{x})$ is well defined.[1] (2) The function defined by each expression E is Borel-measurable (for definition of Borel-measurability, see, e.g. [Billingsley 1995]). From measure theory it is known that these conditions hold in particular for programs where expressions are build using the standard arithmetic operators of addition, subtraction, multiplication, and division (provided that expressions evaluating to zero are not allowed as divisors).

Affine Expressions. An *affine expression* over the set of variables $\{x_1, \ldots, x_n\}$ is an expression of the form $d + \sum_{i=1}^{n} a_i x_i$, where $d, a_1, \ldots, a_n$ are real-valued constants. A function of the form $E(\mathbf{x})$ for some affine expression E is called affine. As noted above, each affine function is Borel-measurable.

Predicates. A *predicate* is a logical formula obtained by a finite number of applications of conjunction, disjunction and negation operations on *atomic predicates* of the form $E \leq E'$, where E, E' are expressions. We denote by $\mathbf{x} \models E$ the fact that E is satisfied by substituting values from of $\mathbf{x}$ for the corresponding variables in E.

Linear constraints, assertions, predicates. In the case of predicates involving only linear expression we use the following standard nomenclature:

- *Linear Constraint.* A *linear constraint* is a formula of the form ψ or $\neg\psi$, where ψ is a non-strict inequality between affine expressions.
- *Linear Assertion.* A *linear assertion* is a finite conjunction of linear constraints.
- *Propositionally Linear Predicate.* A *propositionally linear predicate* (PLP) is a finite disjunction of linear assertions.

The Syntax of Probabilistic Programs (PPs). We consider the standard syntax for probabilistic programs, which encompasses basic programming mechanisms such as assignment statement (indicated by ':='), while-loop, if-branch. Expressions appear on right-hand sides of assignments, and predicates act as loop guards and conditions in if-then-else statements. We also consider basic probabilistic mechanisms such as probabilistic branch (indicated by 'prob') and random sampling (e.g. $x := \mathbf{sample}(\text{Uniform}[-2, 1])$ assigns to x a random number uniformly sampled from interval $[-2, 1]$). We also allow constructs for (demonic) non-determinism, in particular non-deterministic branching indicated by '**if $\star$ then...**' construct and non-deterministic assignment. We assume that all distributions that we sample from have a well-defined and finite expectation; distributions not conforming to this can be abstracted using non-deterministic assignment. Variables (or identifiers) of a probabilistic program are of *real* type, i.e., values of the variables are real numbers. We also assume that each PP $\mathcal{P}$ is preceded by a preamble specifying possible initial values of program variables: the preamble consists of a single predicate characterizing possible initial valuations.

Affine Probabilistic Programs (APPs). A probabilistic program is *affine* if all the expressions that occur in the program (i.e. in loop guards, conditionals, right-hand sides of assignments) are affine and if the set of possible initial valuations is a polyhedron. We refer to the class of affine probabilistic programs as APPs.

---

[1]Our results can be easily extended to programs where encountering an expression of undefined value, such as division by zero, triggers an exception which terminates the program, but we abstract away from such details for the sake of clarity.

Due to space restrictions, details of syntax (such as grammar) are relegated to the full version [Agrawal et al. 2017]. For an example see Figure 1.

2.3 Semantics of Probabilistic Programs

Basics of Probability Theory. A probability space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a non-empty set (so called *sample space*), $\mathcal{F}$ is a *sigma-algebra* of measurable sets over Ω, i.e. a collection of subsets of Ω that contains the empty set $\emptyset$, and that is closed under complementation and countable unions, and $\mathbb{P}$ is a *probability measure* on $\mathcal{F}$, i.e., a function $\mathbb{P} \colon \mathcal{F} \to [0, 1]$ such that: (1) $\mathbb{P}(\emptyset) = 0$, (2) for all $A \in \mathcal{F}$ it holds $\mathbb{P}(\Omega \smallsetminus A) = 1 - \mathbb{P}(A)$, and (3) for all pairwise disjoint countable set sequences $A_1, A_2, \dots \in \mathcal{F}$ (i.e., $A_i \cap A_j = \emptyset$ for all $i \neq j$) we have $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}(\bigcup_{i=1}^{\infty} A_i)$.

Following the usual probabilistic terminology, we say that *almost all* ω belonging to some set $O \subseteq \Omega$ satisfy some property Ψ if it holds that $\mathbb{P}(\{\omega \in O \mid \omega$ does not satisfy $\Psi\}) = 0$.

Random variables and filtrations. A *random variable* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $R \colon \Omega \to \mathbb{R} \cup \{\infty\}$, i.e., a function such that for every $a \in \mathbb{R} \cup \{\infty\}$ the set $\{\omega \in \Omega \mid R(\omega) \le a\}$ belongs to $\mathcal{F}$. If $R(\omega) \in \mathbb{R}$ for all $\omega \in \Omega$, we say that R is *real-valued.* We denote by $\mathbb{E}[X]$ the *expected value* of a random variable X (see [Billingsley 1995, Chapter 5] for a formal definition). We also denote by $\{X \sim x\}$, where $\sim$ is a comparison operator and $x \in \mathbb{R} \cup \{\pm\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) \sim x\}$. A *random vector* in $(\Omega, \mathcal{F}, \mathbb{P})$ is a vector whose every component is a random variable in this probability space. A *stochastic process* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence of random vectors in this space. We will also use random variables of the form $R \colon \Omega \to S$ for some finite set S, which is easily translated to the variables above. A *filtration* of a sigma-algebra $\mathcal{F}$ is a sequence $\{\mathcal{F}_i\}_{i=0}^{\infty}$ of σ-algebras such that $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n \subseteq \cdots \subseteq \mathcal{F}$. A stochastic process $\{\mathbf{X}_i\}_{i=0}^{\infty}$ is *adapted* to filtration $\{\mathcal{F}_i\}_{i=0}^{\infty}$ if for all i it holds that each component of $\mathbf{X}_i$ is $\mathcal{F}_i$-measurable.

Distributions. We assume the standard definition of a probability distribution specified by a cumulative distribution function [Billingsley 1995]. We denote by $\mathcal{D}$ the set of probability distributions on real numbers, both discrete and continuous.

Probabilistic Control Flow Graphs. We consider standard operational semantics of PPs defined via an uncountable state-space Markov decision process (MDP) (uncountable due to real-valued variables). That is, we associate to each program a certain stochastic process. To define this process, we first define so called *probabilistic control flow graphs* [Chatterjee et al. 2016a].

Definition 2.1. A *probabilistic control flow graph (pCFG)* is a tuple $C = (L, V, \ell_{init}, \Xi_{init}, \mapsto, Up, Pr, G)$, where

- L is a finite set of *locations* partitioned into four pairwise disjoint subsets L_{NB}, L_{PB}, L_D, and L_A of non-deterministic branching, probabilistic branching, deterministic, and assignment locations;
- $V = \{x_1, \dots, x_{|V|}\}$ is a finite set of *program variables* (note that $V \subseteq \mathcal{V}$) ;
- ℓ_{init} is an initial location and Ξ_{init} is a set of initial *assignment vectors*;
- $\mapsto \subseteq L \times L$ is a transition relation;
- Up is a function assigning to each transition outgoing from an assignment location a tuple (i, u), where $1 \le i \le |V|$ is a *target variable index* and u is an *update element*, which can be one of the following mathematical objects: (a) a Borel-measurable function $u \colon \mathbb{R}^{|V|} \to \mathbb{R}$; (b) a distribution $d \in \mathcal{D}$ with a well-defined expectation; or (c) a set $R \subseteq \mathbb{R}$ (representing a non-deterministic update).

- $Pr = \{Pr_\ell\}_{\ell \in L_{PB}}$ is a collection of probability distributions, where each Pr_ℓ is a discrete probability distribution on the set of all transitions outgoing from ℓ;
- G is a function assigning a propositionally linear predicate (a *guard*) over V to each transition outgoing from a deterministic location.

We assume that each location has at least one outgoing transition. Also, for every deterministic location ℓ we assume the following: if $\tau_1, \ldots, \tau_k$ are all transitions outgoing from ℓ, then $G(\tau_1) \vee \cdots \vee G(\tau_k) \equiv true$ and $G(\tau_i) \wedge G(\tau_j) \equiv false$ for each $1 \leq i < j \leq k$. For each distribution d appearing in the pCFG we assume the following features are known: expected value $\mathbb{E}[d]$ of d and a set SP_d containing the *support* of d. The support is the smallest closed set of real numbers whose complement has probability zero under d.[2] Finally, we assume that each assignment location has at most (and thus exactly) one outgoing transition. The translation from probabilistic programs to the corresponding pCFG is standard [Chatterjee et al. 2016b], and the details are presented in the full version [Agrawal et al. 2017].

Configurations. A *configuration* of a pCFG C is a tuple $(\ell, \mathbf{x})$, where ℓ is a location of C and $\mathbf{x}$ is an $|V|$-dimensional vector. We say that a transition τ is *enabled* in a configuration $(\ell, \mathbf{x})$ if ℓ is the source location of τ and in addition, $\mathbf{x} \models G(\tau)$ provided that ℓ is deterministic.

Executions and reachable configurations. We say that a configuration $(\ell', \mathbf{x}')$ is a *successor* of a configuration $(\ell, \mathbf{x})$ if there is a transition $\tau = (\ell, \ell')$ enabled in $(\ell, \mathbf{x})$ and $\mathbf{x}'$ satisfies the following:

- if ℓ is not an assignment location, then $\mathbf{x}' = \mathbf{x}$;
- if ℓ is an assignment location with $Up(\tau) = (j, u)$, then $\mathbf{x}_{i+1} = \mathbf{x}_i(j \leftarrow a)$ where a satisfies one of the following depending on the type of u: (a) if u is a Borel-measurable function, then $a = u(\mathbf{x})$; (b) if u is a distribution d, then $a \in supp(d)$; and (c) if u is a set, then a is some element of u.

A *finite path* (or *execution fragment*) of length k in C is a finite sequence of configurations $(\ell_0, \mathbf{x}_0) \cdots (\ell_k, \mathbf{x}_k)$ such that $\ell_0 = \ell_{init}$, $\mathbf{x}_0 \in \Xi_{init}$, and for each $0 \leq i < k$ the configuration $(\ell_{i+1}, \mathbf{x}_{i+1})$ is a successor of $(\ell_i, \mathbf{x}_i)$. A *run* (or *execution*) in C is an infinite sequence of configurations whose every finite prefix is a finite path. A configuration $(\ell, \mathbf{x})$ is *reachable* from the initial configuration $(\ell_{init}, \mathbf{x}_{init})$ (where, $\mathbf{x}_{init} \in \Xi_{init}$) if there is a finite path starting in $(\ell_{init}, \mathbf{x}_{init})$ that ends in $(\ell, \mathbf{x})$. We denote by $Conf_C$, $Fpath_C$ and Run_C the sets of all configurations, finite paths and runs in C, respectively, dropping the index C when known from the context.

Non-determinism and Schedulers. The probabilistic behaviour of C can be captured by constructing a suitable probability measure over the set of all its runs. Before this can be done, non-determinism in C needs to be resolved. This is achieved using the standard notion of a *scheduler*. Note that there are two sources of non-determinism in our programs: one in branching and one in assignments. We call a location ℓ non-deterministic if ℓ is a non-deterministic branching location or if ℓ is an assignment location with the only transition τ outgoing from ℓ having a non-deterministic assignment. A configuration $(\ell, \mathbf{x})$ is non-deterministic if ℓ is non-deterministic.

Definition 2.2 (Schedulers). A scheduler in a pCFG C is a function σ assigning to every finite path that ends in a non-deterministic configuration $(\ell, \mathbf{x})$ a probability distribution on successor configurations of $(\ell, \mathbf{x})$.

---

[2]In particular, a support of a *discrete* probability distribution d is simply the at most countable set of all points on a real line that have positive probability under d.

Measurable schedulers. Note that schedulers can be viewed as partial functions from the set *Fpath* to the set of probability distributions over the set *Conf*. Since we deal with programs operating over real-valued variables, both *Fpath* and *Conf* can be uncountable sets. Hence, we impose an additional *measurability* condition on schedulers, so as to ensure that the semantics of probabilistic non-deterministic programs is defined in a mathematically sound way. The definition of a measurable scheduler that we use is the standard one used when dealing with systems that exhibit both probabilistic and non-deterministic behaviour over a continuous state space [Neuhäußer et al. 2009; Neuhäußer and Katoen 2007]. The definition is somewhat technical, and not substantial to our results, so we defer it to [Agrawal et al. 2017]. In the rest of this work, we refer to measurable schedulers simply as "schedulers." In particular, if the set of all reachable variable valuations is discrete (such as the integers), then each scheduler in the associated pCFG is measurable.

Stochastic process. A pCFG C together with a scheduler σ and initial valuation $\mathbf{x}_{init} \in \Xi_{init}$ define a stochastic process which produces a random run $(\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)(\ell_2, \mathbf{x}_2)\cdots$. The evolution of this process can be informally described as follows: we start in the initial configuration, i.e. $(\ell_0, \mathbf{x}_0) = (\ell_{init}, \mathbf{x}_{init})$. Now assume that i steps have elapsed, i.e. a finite path $\pi_i = (\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\cdots(\ell_i, \mathbf{x}_i)$ has already been produced. Then a successor configuration $(\ell_{i+1}, \mathbf{x}_{i+1})$ is chosen as follows:

- If ℓ_i is a non-deterministic location, then $(\ell_{i+1}, \mathbf{x}_{i+1})$ is sampled according to scheduler σ, i.e. from the distribution $\sigma(\pi_i)$.
- If ℓ_i is an assignment location (but not a non-deterministic one), there is exactly one transition $\tau = (\ell_i, \ell')$ outgoing from it and we put $\ell_{i+1} = \ell'$. Denoting $Up(\tau) = (j, u)$, the vector $\mathbf{x}_{i+1}$ is then defined as $\mathbf{x}_{i+1} = \mathbf{x}_i(j \leftarrow a)$ where a is chosen depending on u:
  - If u is a function $u \colon \mathbb{R}^{|V|} \to \mathbb{R}$, then $a = f(\mathbf{x}_i)$.
  - If u is a distribution d, then a is sampled from d.
- In all other cases we have $\mathbf{x}_{i+1} = \mathbf{x}_i$, and ℓ_{i+1} is determined as follows:
  - If ℓ_i is a probabilistic branching location, then a transition (ℓ_i, ℓ') is sampled from Pr_{ℓ_i} and we put $\ell_{i+1} = \ell'$;
  - If ℓ_i is deterministic, then there is exactly one transition (ℓ_i, ℓ') enabled in $(\ell_i, \mathbf{x}_i)$, in which case we put $\ell_{i+1} = \ell'$.

The above intuitive explanation can be formalized by showing that each pCFG C together with a scheduler σ and initial valuation $\mathbf{x}_{init} \in \Xi_{init}$ uniquely determine a certain probability space $(\Omega_{Run}, \mathcal{R}, \mathbb{P}^\sigma_{\mathbf{x}_{init}})$ in which Ω_{Run} is the set of all runs in C, and a stochastic process $C = \{\mathbf{C}_i\}_{i=0}^\infty$ in this space such that for each run $\varrho \in \Omega_{Run}$ we have that $\mathbf{C}_i(\varrho)$ is the i-th configuration on ϱ (i.e., $\mathbf{C}_i$ is a random vector $(\ell_i, \mathbf{x}_i)$ with ℓ_i taking values in L and $\mathbf{x}_i$ being a random vector of dimension $|V|$ consisting of real-valued random variables). The sigma-algebra $\mathcal{R}$ is the smallest (w.r.t. inclusion) sigma-algebra under which all the functions $\mathbf{C}_i$, for all $i \geq 0$, are $\mathcal{R}$-measurable. The probability measure $\mathbb{P}^\sigma_{\mathbf{x}_{init}}$ is such that for each i, the distribution of $\mathbf{C}_i$ reflects the aforementioned way in which runs are randomly generated. The formal construction of $\mathcal{R}$ and $\mathbb{P}^\sigma_{\mathbf{x}_{init}}$ proceeds via the standard *cylinder construction* [Ash and Doléans-Dade 2000, Theorem 2.7.2] and is somewhat technical, hence we omit it. We denote by $\mathbb{E}^\sigma_{\mathbf{x}_{init}}$ the expectation operator in probability space $(\Omega_{Run}, \mathcal{R}, \mathbb{P}^\sigma_{\mathbf{x}_{init}})$.

2.4 Almost-Sure and Positive Termination

Termination and termination time. In the following, consider a PP P and its associated pCFG C_P. This pCFG has a special location ℓ_{term} corresponding to the value of the program counter after executing P. We say that a run *terminates* if it reaches a configuration whose first component is ℓ_{term}. We define a random variable *Term* such that for each run ϱ the value $Term(\varrho)$ represents the first point

```
while  x ≥ 0  and  y ≥ 0  do
    if  *  then
        if  prob(0.5)  then
            x := x − 2
        else  skip
        fi
    else
        x := 2x ;
        y := y − 1
    fi
od
```
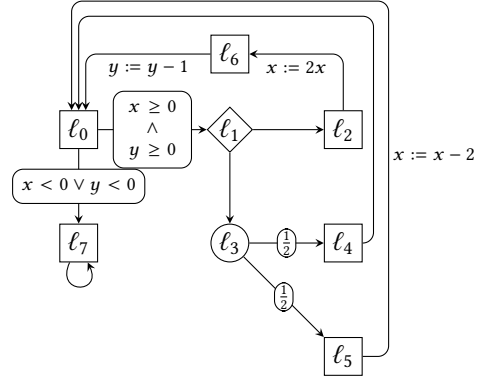

Fig. 1. An example App and its associated pCFG. Probabilistic branching locations are depicted by circles, with probabilities given on outgoing transitions, nondeterministic branching locations are denoted by diamonds. Transitions are labelled by their effects. Location ℓ_0 is initial and ℓ_7 is terminal.

in time when the current location is ℓ_{term}. If a run ϱ does *not* terminate, then $Term(\varrho) = \infty$. We call $Term$ the *termination time* of $\mathcal{P}$. We consider the following fundamental computational problems regarding termination:

(1) *Almost-sure termination:* A probabilistic program P is almost-surely (a.s.) terminating if under each scheduler σ and for each initial valuation $\mathbf{x}_{init} \in \Xi_{init}$ it holds that $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\varrho \mid \varrho \text{ terminates}\}) = 1$, or equivalently, if it holds $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(Term < \infty) = 1$. In almost-sure termination question for P we aim to prove that P is almost-surely terminating.

(2) *Positive termination:* A probabilistic program P is positively terminating if under each scheduler σ and for each initial valuation $\mathbf{x}_{init} \in \Xi_{init}$ it holds that $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] < \infty$. In positive termination question for P we aim to prove that P is positively terminating. Note that each positively terminating program is also a.s. terminating, but the converse does not hold.

3 LEXICOGRAPHIC SUPERMARTINGALES

In this section we introduce the notion of a *lexicographic ranking supermartingale*, which generalizes the standard notion of a ranking supermartingale. However, to define any form of a supermartingale, we need the crucial notion of conditional expectation.

Conditional Expectation. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $X \colon \Omega \to \mathbb{R}$ an $\mathcal{F}$-measurable function, and $\mathcal{F}' \subseteq \mathcal{F}$ sub-sigma-algebra of $\mathcal{F}$. A *conditional expectation* of X given $\mathcal{F}'$ is an $\mathcal{F}'$-measurable random variable denoted by $\mathbb{E}[X|\mathcal{F}']$ which satisfies, for each set $A \in \mathcal{F}'$, the following:

$$\mathbb{E}[X \cdot 1_A] = \mathbb{E}[\mathbb{E}[X|\mathcal{F}] \cdot 1_A], \tag{1}$$

where $1_A \colon \Omega \to \{0, 1\}$ is an *indicator function* of A, i.e. function returning 1 for each $\omega \in A$ and 0 for each $\omega \in \Omega \setminus A$. Note that the left hand-side of (1) intuitively represents the expected value of $X(\omega)$ with domain restricted to A.

Note that any $\mathcal{F}'$-measurable random variable satisfying (1) can be called a conditional expectation. The definition does not guarantee that the conditional expectation is uniquely defined or that it exists at all (though if X itself happens to be $\mathcal{F}$-measurable, which is not always the case, then it satisfies the definition of conditional expectation). However, we have the following:

PROPOSITION 3.1. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $X \colon \Omega \to \mathbb{R}$ an $\mathcal{F}$-measurable function, and $\mathcal{F}' \subseteq \mathcal{F}$ a sub-sigma-algebra of $\mathcal{F}$. Assume that one of the following conditions hold: (a) $\mathbb{E}[|X|] < \infty$, or (b) X is real-valued and non-negative. Then there exists a conditional expectation of X given $\mathcal{F}'$ and it is almost-surely unique, i.e. for each two $\mathcal{F}'$-measurable functions f, g that satisfy (1) it holds $\mathbb{P}(\{\omega \mid f(\omega) \neq g(\omega)\}) = 0$.*

PROOF (KEY IDEAS). The proof for the case when $\mathbb{E}[|X|] < \infty$ is standard and appears in many textbooks on probability theory (e.g. [Ash and Doléans-Dade 2000; Billingsley 1995; Rosenthal 2006]). The proof for the second case is essentially the same: the condition that X is non-negative and not admitting infinite value suffices for satisfying the assumptions of Radon-Nikodym Theorem, the main theoretical tool used in the proof. For the sake of completeness we present the proof in [Agrawal et al. 2017]. □

Since the constraint (1) defining conditional expectation is phrased in terms of expected values, the almost-sure uniqueness cannot be strengthened to uniqueness, as re-defining a random variable on a set of probability zero does not change its expectation. In the following, when we say that a conditional expectation of a random variable X satisfies some inequality (e.g. $\mathbb{E}[X \mid \mathcal{F}] \geq 0$) on a set $L \subseteq \Omega$, we mean that for each $\mathcal{F}$-measurable function $\mathbb{E}[X \mid \mathcal{F}]$ satisfying (1) the inequality holds on some subset $L' \subseteq L$ such that $\mathbb{P}(L') = \mathbb{P}(L)$.

REMARK 1. *In context of probabilistic programs we work with probability spaces of the form $(\Omega, \mathcal{R}, \mathbb{P}^\sigma)$, where Ω is a set of runs in some pCFG C and $\mathcal{R}$ is (the smallest) sigma-algebra such that all the functions $\mathbf{C}_i$, where $i \in \mathbb{N}_0$ and σ is a scheduler, are $\mathcal{R}$-measurable. In such a setting we can also consider sub-sigma-algebras $\mathcal{R}_i$, $i \in \mathbb{N}_0$, of $\mathcal{R}$, where $\mathcal{R}_i$ is the smallest sub-sigma-algebra of $\mathcal{R}$ such that all the functions $\mathbf{C}_j$, $0 \leq j \leq i$, are $\mathcal{R}_i$-measurable. Intuitively, each set $A \in \mathcal{R}_i$ consists of runs whose first i steps satisfy some property. Then, for each $A \in \mathcal{R}_i$, the value $\mathbb{E}[\mathbb{E}[X | \mathcal{R}_i] \cdot 1_A]$ represents the expected value of $X(\varrho)$ for the randomly generated run ϱ provided that we restrict to runs whose prefix of length i satisfies the property given by A. The sequence $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$ forms a filtration of $\mathcal{R}$, which we call a* canonical filtration.

Definition 3.2. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and $\{\mathcal{F}_i\}_{i=0}^{\infty}$ a filtration of $\mathcal{F}$. A random variable $T \colon \Omega \to \mathbb{N}_0 \cup \{+\infty\}$ is a *stopping time* w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$ if for each $i \in \mathbb{N}_0$ the set $\{T > i\}$ belongs to $\mathcal{F}_i$.

Note that termination time *Term* of a PP is a stopping time w.r.t. canonical filtration.

Definition 3.3 (Lexicographic Ranking Supermartingale). Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $\{\mathcal{F}_i\}_{i=0}^{\infty}$ a filtration of $\mathcal{F}$, T a stopping time w.r.t. that filtration, and $\epsilon \geq 0$. An n-dimensional real-valued stochastic process $\{\mathbf{X}_i\}_{i=0}^{\infty}$ is a *lexicographic ϵ-ranking supermartingale for T (ϵ-LexRSM)* if the following conditions hold:

(1) For each $1 \leq j \leq n$ the 1-dimensional stochastic process $\{\mathbf{X}_i[j]\}_{i=0}^{\infty}$ is adapted to $\{\mathcal{F}_i\}_{i=0}^{\infty}$.
(2) For each $\omega \in \Omega$, $i \in \mathbb{N}_0$ and $1 \leq j \leq n$ it holds $\mathbf{X}_i(\omega)[j] \geq 0$.
(3) For each $i \in \mathbb{N}_0$ there exists a partition of the set $\{T > i\}$ into $n + 1$ subsets $L_1^i, \ldots, L_n^i, L_{n+1}^i$, all of them $\mathcal{F}_i$-measurable, such that for each $1 \leq j \leq n$:
   • $\mathbb{E}[\mathbf{X}_{i+1}[j] \mid \mathcal{F}_i] \leq \mathbf{X}_i[j] - \epsilon$ on L_j^i;
   • for all $1 \leq j' < j$ we have $\mathbb{E}[\mathbf{X}_{i+1}[j'] \mid \mathcal{F}_i] \leq \mathbf{X}_i[j']$ on L_j^i; and
   • $\mathbb{E}[\mathbf{X}_{i+1}[j] \mid \mathcal{F}_i] \leq \mathbf{X}_i[j]$ on L_{n+1}^i.

The *n-dimensional LexRSM is* strict *if $L_{n+1}^i = \emptyset$ for each i.*

An *instance* of an n-dimensional LexRSM $\{\mathbf{X}_i\}_{i=0}^{\infty}$ is a tuple $(\{\mathbf{X}_i\}_{i=0}^{\infty}, \{L_1^i, \ldots, L_{n+1}^i\}_{i=0}^{\infty})$ where the second component is a sequence of partitions of Ω satisfying the condition in Definition 3.3. Intuitively, the sets L_j^i for $1 \le j \le n$ represent the lexicographic ranking condition, i.e. for strict LexRSMs, we are in each step able to partition Ω into subsets such that on j-th subset the j-th component of $\mathbf{X}$ is expected to decrease while the previous components are not expected to increase. On additional sets L_{n+1}^i, none of the components is expected to increase, but decrease is not required: this will become handy later when we deal with compositional LexRSM-based proofs (Section 6) – we will not require decrease in every step as long as decrease happens at least once on each cycle in the pCFG. We say that $\omega \in \Omega$ has level j in step i of instance $(\{\mathbf{X}_i\}_{i=0}^{\infty}, \{L_1^i, \ldots, L_{n+1}^i\}_{i=0}^{\infty})$ if $\omega \in L_j^i$. We also say that ω has level 0 in step i if $T(\omega) \le i$.

The strict 1-dimensional lexicographic ϵ-ranking supermartingale is, to a large extent, equivalent to the notion of a ranking supermartingale as studied in [Chatterjee et al. 2016b; Fioriti and Hermanns 2015]. There is one significant difference: in these works there is an additional *integrability* condition imposed on the one-dimensional process $\{X_i\}_{i=0}^{\infty}$, which requires that for each $i \ge 0$ it holds $\mathbb{E}[|X_i|] < \infty$ (or equivalently $\mathbb{E}[X_i] < \infty$, as the process is required to be non-negative). We do not impose this condition, which simplifies possible application of LexRSMs to programs with non-linear arithmetic, where, as already shown in [Fioriti and Hermanns 2015], integrability of program variables is not guaranteed. The reason why integrability condition can be dropped is that it is only needed in the previous works to ensure that the conditional expectations exist and are well-defined. However, the existence of conditional expectations is also guaranteed for random variables that are real-valued and non-negative, see Proposition 3.1. This is exactly the case in both the original 1-dimensional ranking supermartingales and our generalization to LexRSMs.

The following theorem states our main mathematical result on LexRSMs.

THEOREM 3.4. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $\{\mathcal{F}_i\}_{i=0}^{\infty}$ a filtration of $\mathcal{F}$, T a stopping time w.r.t. that filtration, and $\epsilon > 0$. Assume there exists an n-dimensional ϵ-LexRSM for T and its instance $(\{\mathbf{X}_i\}_{i=0}^{\infty}, \{L_1^i, \ldots, L_{n+1}^i\}_{i=0}^{\infty})$ such that $\mathbb{P}(\{\omega \in \Omega \mid \text{level of } \omega \text{ is } < n + 1 \text{ in infinitely many steps}\}) = 1$. Then $\mathbb{P}(T < \infty) = 1$. In particular, if there exists a strict ϵ-LexRSM for T, then $\mathbb{P}(T < \infty) = 1$.*

PROOF. The proof proceeds by contradiction, i.e. we assume that an ϵ-LexRSM for T satisfying the above conditions exists and $\mathbb{P}(T = \infty) > 0$. For succinctness we denote the set $\{\omega \mid T(\omega) = \infty\}$ by A_∞.

For $\omega \in \Omega$ we denote the level of ω at step i by $lev_i(\omega)$. The value $lev_i(\omega)$ is well-defined for all ω and moreover, the random variable lev_i is $\mathcal{F}_i$-measurable. We denote by $min\text{-}lev(\omega)$ the smallest $0 \le j \le n$ such that j is a level of ω at infinitely many steps. Note that $\omega \in A_\infty$ if and only if $min\text{-}lev(\omega) \ne 0$, so $\mathbb{P}(A_\infty) = \mathbb{P}(\{min\text{-}lev \ne 0\})$. We denote by M_i the set of all ω's with $min\text{-}lev(\omega) = i$.

Throughout the proof we use several times the following fundamental fact: if $\mathbb{P}(A) > 0$ for some set A and $A = A_1 \cup A_2 \cup A_3 \cdots$ for some sequence of sets $A_1, A_2, A_3, \ldots$, then there exists i such that $\mathbb{P}(A_i) > 0$.

Now $A_\infty = M_1 \cup \cdots \cup M_n \cup M_{n+1}$ and $\mathbb{P}(M_{n+1}) = 0$ (as the measure of ω's that have level $< n + 1$ in only finitely many steps is zero, per Theorem's assumption). Hence, there must be $1 \le j^* \le n$ s.t. $\mathbb{P}(M_{j^*}) > 0$, i.e. with positive probability the smallest level appearing infinitely often is j^*. For each $\omega \in M_{j^*}$ there is the smallest number $i_{\omega,j^*} \in \mathbb{N}_0$ such that for all $i \ge i_{\omega,j^*}$ it holds $lev_i(\omega) \ge j^*$, i.e. after step i_{ω,j^*} the level of ω in all steps up to infinity is at least j^*. Denote by $S_{j^*,i}$ the set of all ω's

in M_{j^*} s.t. $i_{\omega, j^*} = i$. Since $M_{j^*} = M_{j^*,1} \cup M_{j^*,2} \cup M_{j^*,3} \cup \cdots$, there is $i^* \in \mathbb{N}_0$ s.t. $\mathbb{P}(M_{j^*, i^*}) > 0$. That is, there is a point in time such that with positive probability, after this point the level of ω is at least j^*, and it is equal to j^* infinitely many times. Continuing on the same note, for each $B \in \mathbb{N}$ we denote by M_{j^*, i^*}^B the set off all ω's in M_{j^*, i^*} s.t. $\mathbf{X}_{i^*}[j^*](\omega) \leq B$. Since $M_{j^*, i^*} = M_{j^*, i^*}^1 \cup M_{j^*, i^*}^2 \cup M_{j^*, i^*}^3 \cup \cdots$, there is $B^* \in \mathbb{N}$ s.t. $\mathbb{P}(M_{j^*, i^*}^{B^*}) > 0$.

So there is a set of positive probability $M = M_{j^*, i^*}^{B^*}$ such that for all ω's in the set: after step i^* the level of ω is at least j^* (which, intuitively, means that $\mathbf{X}[j^*]$ does not have a tendency to increase after this time step on ω's in M), the level of ω is infinitely often equal to j^* (intuitively, $\mathbf{X}[j^*]$ has infinitely often the tendency to decrease by $\geq \epsilon$ for ω's in M), and at time i^* the value of $\mathbf{X}[j^*]$ is bounded (by B) on M. This should, again intuitively, lead to a conclusion that when "restricted to M", $\mathbf{X}[j^*]$ has tendency to decrease unboundedly over time, a contradiction with non-negativeness of $\mathbf{X}[j^*]$. However, proving this intuitive result is much more intricate: most importantly, it is not clear what "restricted to M" stands for. The stochastic process $\{\mathbf{X}\}_{i=0}^\infty$ as well as the LexRSM conditions are tied to the filtration $\{\mathcal{F}_i\}_{i=0}^\infty$, but the set M is not necessarily $\mathcal{F}_i$ measurable for any concrete i, since whether ω belongs to M depends on values of $lev_i(\omega)$ for *infinitely many* i. Hence, we use a work-around.

Let D be the set of all $\omega \in \Omega$ such that $\mathbf{X}_{i^*}[j^*](\omega) \leq B^*$. Note that $M \subseteq D$ and $D \in \mathcal{F}_{i^*}$ (and thus also $D \in \mathcal{F}_{i'}$ for all $i' \geq i^*$). Define a stopping time F w.r.t. filtration $\{\mathcal{F}_i\}_{i=0}^\infty$ as follows: for all $\omega \in \Omega$ we put $F(\omega) = \inf\{k \in \mathbb{N}_0 \mid k \geq i^* \text{ and } lev_k(\omega) < j^*\}$.

Define a (one-dimensional) stochastic process $\{Y_k\}_{k=0}^\infty$ as follows:

$$Y_k(\omega) = \begin{cases} 0 & \text{if } \omega \notin D \\ B^* & \text{if } \omega \in D \text{ and } k < i^* \\ \mathbf{X}_k[j^*](\omega) & \text{if } \omega \in D, k \geq i^* \text{ and } F(\omega) > k \\ \mathbf{X}_{F(\omega)}[j^*](\omega) & \text{if } \omega \in D, k \geq i^* \text{ and } F(\omega) \leq k \ . \end{cases}$$

Intuitively, the process $\{Y_k\}_{k=0}^\infty$ is an over-approximation of what we would like to call "$\mathbf{X}[j^*]$ restricted to M." We prove several properties of the process. First, clearly for all $k \geq 0$, $Y_k(\omega) \geq 0$. Second, for each $k \geq i^*$, the variable Y_k is $\mathcal{F}_k$-measurable, as $D \in \mathcal{F}_{i^*}$, $\{\mathbf{X}_i[j^*](\omega)\}_{i=0}^\infty$ is adapted to the filtration $\{\mathcal{F}_i\}_{i=0}^\infty$, and F is a stopping time w.r.t. this filtration. Finally, for any $k \in \mathbb{N}_0$ denote by $\sharp_k$ the random variable such that $\sharp_k(\omega) = |\{i' \in \mathbb{N} \mid i^* \leq i' < k \text{ and } lev_{i'}(\omega) = j^*\}|$, i.e. $\sharp_k(\omega)$ counts the number of steps between i^* and k in which level is j^*. We prove that for each $k \geq i^*$ it holds

$$\mathbb{E}[Y_k] \leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=0}^{k-i^*} \ell \cdot \mathbb{P}(D \cap \{F \geq k\} \cap \{\sharp_k = \ell\}). \tag{2}$$

The proof of (2) goes by induction on k. The computations being somewhat technical, we defer them to the full version [Agrawal et al. 2017].

Now according to (2) it holds $\mathbb{E}[Y_k] \leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=0}^{k-i^*} \ell \cdot \mathbb{P}(D \cap \{F \geq k\} \cap \{\sharp_k = \ell\})$ for all $k \geq i^*$. Let $m = 3B^* \cdot \mathbb{P}(D)/(\epsilon \cdot \mathbb{P}(M))$. For each $\omega \in M$ we see level j^* infinitely often, so there exists step $k(\omega) \geq i^*$ such that $\sharp_{k(\omega)} \geq m$, i.e. ω has level j^* at least in m steps between steps i^* and $k(\omega)$. Clearly, $M = \bigcup_{\ell=i^*}^\infty (M \cap \{k(\omega) \leq \ell\})$ and hence $\mathbb{P}(M) = \lim_{\ell \to \infty} \mathbb{P}(M \cap \{k(\cdot) \leq \ell\})$. Thus, there exists $\ell_0 \geq m + i^*$ such that $\mathbb{P}(M \cap \{\sharp_{\ell_0} \geq m\}) = \mathbb{P}(M \cap \{k(\cdot) \leq \ell\}) \geq \mathbb{P}(M)/2$.

Clearly $M \cap \{\sharp_{\ell_0} \geq m\} \subseteq D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} \geq m\}$. From (2) it follows that

$$\mathbb{E}[Y_{\ell_0}] \leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=0}^{\ell_0-i^*} \ell \cdot \mathbb{P}(D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} = \ell\})$$

$$= B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=1}^{\ell_0-i^*} \mathbb{P}(D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} \geq \ell\})$$

$$\leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=1}^{m} \mathbb{P}(D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} \geq \ell\})$$

$$\leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot \sum_{\ell=1}^{m} \mathbb{P}(D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} \geq m\})$$

$$\leq B^* \cdot \mathbb{P}(D) - \epsilon \cdot m \cdot \mathbb{P}(M)/2 < 0,$$

where the second line follows by standard re-arranging of terms, the third line follows from the fact that $m \leq \ell_0 - i^*$, the fourth line follows from $\{\sharp_{\ell_0} \geq m\} \subseteq \{\sharp_{\ell_0} \geq \ell\}$ for each $\ell \leq m$, the first inequality on the last line follows by using $\mathbb{P}(M)/2 \leq \mathbb{P}(M \cap \{\sharp_{\ell_0} \geq m\}) \leq \mathbb{P}(D \cap \{F \geq \ell_0\} \cap \{\sharp_{\ell_0} \geq m\})$, and the last inequality follows by expanding the definition of m. But for each k the random variable Y_k is non-negative, so it must also have a non-negative expectation, a contradiction. Finally, note that for strict n-dimensional LexRSMs the condition of level $< n + 1$ appearing infinitely many times is trivially satisfied. □

REMARK 2. *This section presents a purely mathematical result about general LexRSMs, and does not present specific examples. In the following section we apply the result to PPs and illustrate on examples.*

4 LEXICOGRAPHIC SUPERMARTINGALES FOR PROBABILISTIC PROGRAMS

We now discuss how to leverage the mathematical results of the previous section to provide a sound method for almost-sure termination of probabilistic programs. Hence, for the rest of this section we fix a PP $\mathcal{P}$ and the associated pCFG $C_{\mathcal{P}} = (L, V, \ell_{init}, \Xi_{init}, \mapsto, Up, Pr, G)$.

We aim to define a function assigning a non-negative vector to each configuration (so called measurable map) such that in each point of computation, the expected value of the function after performing one more computational step is smaller (in lexicographic ordering) than the current one. We formalize this property below.

Definition 4.1 (Measurable Maps and Linear Expression Maps). A 1-dimensional *measurable map* for a PP $\mathcal{P}$ is a real-valued function η assigning to each program location ℓ of C_P a Borel-measurable function $\eta(\ell)$ of program variables, i.e. each $\eta(\ell)$ is a function of type $\mathbb{R}^{|V|} \to \mathbb{R}$. As a special case, if all the functions $\eta(\ell)$ are affine, then we call η a 1-dimensional *linear expression map (LEM)*. An n-dimensional measurable/linear expression map is a vector $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$ of 1-dimensional measurable/linear expression maps.

Each 1-dimensional measurable map η and location ℓ determines a function $\eta(\ell)$ which takes as an argument a $|V|$-dimensional vector. We use $\eta(\ell, \mathbf{x})$ as a shorthand notation for $\eta(\ell)(\mathbf{x})$.

We now formalize the notion of a transition in a pCFG being ranked by a measurable map. We first define this notion for transitions that do not go out of a probabilistic branching location, as the latter require a special treatment.

Definition 4.2. Let η be a measurable map, $(\ell, \mathbf{x})$ be a configuration such that $\ell \notin L_{PB}$ and let $\tau = (\ell, \ell')$ be a transition outgoing from ℓ. For an $\epsilon \in \mathbb{R}$ we say that τ is ϵ-*ranked by η from* $(\ell, \mathbf{x})$ if the following conditions are satisfied, depending on the type of ℓ:

- if ℓ is a deterministic or non-deterministic branching location, then

$$\eta(\ell', \mathbf{x}) \le \eta(\ell, \mathbf{x}) - \epsilon;$$

- If ℓ is an assignment location, then we distinguish three cases, depending on $Up(\tau) = (j, u)$ (recall that u is an update element):
    - If $u \colon \mathbb{R}^{|V|} \to \mathbb{R}$ is a Borel-measurable function, then we require

$$\eta(\ell', \mathbf{x}(j \leftarrow u(\mathbf{x}))) \le \eta(\ell, \mathbf{x}) - \epsilon$$

    - If u is a distribution d, then we require

$$\eta(\ell', \mathbf{x}(j \leftarrow \mathbb{E}[d])) \le \eta(\ell, \mathbf{x}) - \epsilon,$$

    where $\mathbb{E}[d]$ is the expected value of the distribution d.
    - If u is a set, then we require

$$\sup_{a \in u} \eta(\ell', \mathbf{x}(j \leftarrow a)) \le \eta(\ell, \mathbf{x}) - \epsilon.$$

Since ranking supermartingales are required to decrease on average, for individual transitions outgoing from L_{PB} it does not make sense to say that they are ranked or not. Instead, for each $\ell \in L_{PB}$ we consider all outgoing transitions together.

Definition 4.3. Let η be a measurable map, and let $(\ell, \mathbf{x})$ be a configuration with $\ell \in L_{PB}$. For an $\epsilon \in \mathbb{R}$ we say that ℓ is ϵ-*ranked by η from* $(\ell, \mathbf{x})$ if

$$\sum_{(\ell, \ell') \in \mapsto} Pr_\ell\,(\ell, \ell') \cdot \eta(\ell', \mathbf{x}) \le \eta(\ell, \mathbf{x}) - \epsilon. \tag{3}$$

To capture the specific of L_{PB}, we introduce the notion of *generalized transition*.

Definition 4.4. A generalized transition of a pCFG C is either a transition of C outgoing from a location not in L_{PB} or a location $\ell \in L_{PB}$.

Intuitively, we represent the set of transitions outgoing from $\ell \in L_{PB}$ by the source location ℓ. For generalized transitions $\tilde{\tau} = \ell \in L_{PB}$ we say that $\tilde{\tau}$ is outgoing from ℓ.

Definitions 4.2 and 4.3 define when is a generalized transition ϵ-ranked by η from configuration $(\ell, \mathbf{x})$. We say that a generalized transition is *unaffected* by η from $(\ell, \mathbf{x})$ if it is 0-ranked by η from $(\ell, \mathbf{x})$.

As in termination analysis of non-probabilistic programs, our LexRSMs are typically supported by *invariants*, i.e. overapproximations of the set of reachable configurations.

Definition 4.5 (Invariant Map and Linear Invariant Map). An *invariant map* for a PP $\mathcal{P}$ is a function I assigning to each location ℓ of $C_\mathcal{P}$ a Borel-measurable set $I(\ell) \subseteq \mathbb{R}^{|V|}$ of variable valuations, so called invariant of ℓ, such that for each configuration $(\ell, \mathbf{x})$ reachable from some initial configuration it holds $\mathbf{x} \in I(\ell)$. Additionally, if each set $I(\ell)$ is of the form $\{\mathbf{x} \mid \mathbf{x} \models \Psi^\ell\}$ for some propositionally linear predicate Ψ^ℓ, then we call I a *linear invariant map* (LIM).

Slightly abusing the notation, we view each LIM equivalently as a function assigning linear predicates (whose satisfaction sets overapproximate the set of reachable valuations) to program locations.

We now have all the ingredients needed to define the notion of LexRSM maps for probabilistic programs. For notational convenience, we extend the function G (which assigns guards to deterministic transitions) to the set of all generalized transition: for a generalized transition τ' which is not a standard transition outgoing from deterministic location, we put $G(\tau') = 0 \leq 0 \equiv true$.

Definition 4.6 (Lexicographic Ranking Supermartingale Map). Let $\epsilon > 0$. An n-dimensional *lexicographic ϵ-ranking supermartingale map* (ϵ-LexRSM map) for a program $\mathcal{P}$ supported by an invariant map I is an n-dimensional measurable map $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$ for $\mathcal{P}$ such that for each configuration $(\ell, \mathbf{x})$ where $\ell \neq \ell_{term}$ and $\mathbf{x} \in I(\ell)$ the following conditions are satisfied:

- for all $1 \leq j \leq n$, $\eta_j(\ell, \mathbf{x}) \geq 0$; and
- for each generalized transition $\tilde{\tau}$ outgoing from ℓ such that $\mathbf{x} \models G(\tilde{\tau})$ (we say that such a $\tilde{\tau}$ is enabled in $(\ell, \mathbf{x})$) there exists $1 \leq j \leq n$ such that
  - $\tilde{\tau}$ is ϵ-ranked by η_j from $(\ell, \mathbf{x})$
  - for all $1 \leq j' < j$ we have that $\tilde{\tau}$ is unaffected by $\eta_{j'}$ from $(\ell, \mathbf{x})$.

If additionally η is a linear expression map, then we call it a linear ϵ-LexRSM map (ϵ-LinLexRSM).

Example 4.7. Consider the program shown in Figure 2. A 2-dimensional 1-LinLexRSM map for the program is given on the right, along with the supporting invariants in square brackets. The invariants and a LinLexRSM on each line belong to the program location in which the program is *before* executing the command on that line. The function is indeed a 1-LinLexRSM, since in the probabilistic branching location ℓ we have the LHS of (3) equal to $3c + 3$ and $3c + 3 \leq 6c + 1 - 1$ for all $c \geq 1$.

The main result is the soundness of ϵ-LexRSM maps for proving a.s. termination.

THEOREM 4.8. *Let $\mathcal{P}$ be a probabilistic program. Assume that there exists an $\epsilon > 0$ and an n-dimensional ϵ-LexRSM map $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$ for $\mathcal{P}$ supported by some invariant map I. Then $\mathcal{P}$ terminates almost surely.*

PROOF. Let σ be any measurable scheduler and $\mathbf{x}_{init} \in \Xi_{init}$ any initial variable valuation in $\mathcal{P}$. We make the proof under the assumption that σ is location-pure, which means that in each nondeterministic branching location it selects (depending on the current execution fragment) a concrete transition to execute with probability 1 (rather than randomizing between "if" and "else" branches). The general proof is more technical and we defer it to [Agrawal et al. 2017].

We define an n-dimensional stochastic process $\{\mathbf{X}_i\}_{i=0}^{\infty}$ on the probability space $(\Omega_{Run}, \mathcal{R}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$ such that for each $i \geq 0$ and $1 \leq j \leq n$ and each run ϱ we put $\mathbf{X}_i[j](\varrho) = \eta_j(\mathbf{C}_i(\varrho))$. We claim that $\{\mathbf{X}_i\}_{i=0}^{\infty}$ is a strict n-dimensional ϵ-LexRSM for the termination time *Term* of $\mathcal{P}$. Clearly, the process is real-valued, componentwise non-negative, and adapted to the canonical filtration of $\mathcal{R}$ (see Remark 1). It remains to prove that condition (3) in Definition 3.3 is satisfied. To this end, for each $i \geq 0$ we define a partition of the set $\{\varrho \in \Omega_{Run} \mid Term(\varrho) > i\}$ into sets $L_1^i, \ldots, L_n^i$ as follows: we put L_j^i to be the set of all runs ϱ such that $Term(\varrho) > i$ and the index j is the smallest one such that the unique generalized transition enabled in $\mathbf{C}_i(\varrho)$ is ranked by η_j from $\mathbf{C}_i(\varrho)$ (here we use that σ is location pure, as in such a case there is indeed a unique enabled generalized transition). Due to definition of an ϵ-LexRSM map such a j exists for all $\varrho \in \{Term > i\}$ and hence we indeed have a partition (so $L_{n+1}^i = \emptyset$ for all i). It remains to prove that irrespective of the initial choice of

```
x := 1 ;  c := 1 ;
while  c ≥ 1  do                    (6c + 2, x)   [c ≥ 0 ∧ x ≥ 1]
   if  prob ( 0.5 )  then           (6c + 1, x)   [c ≥ 1 ∧ x ≥ 1]
      x := 2 · x                    (6c + 3, x)   [c ≥ 1 ∧ x ≥ 1]
   else
      c := 0                        (3, x)        [c ≥ 1 ∧ x ≥ 1]
   fi
od ;
while  x ≥ 0  do  x := x − 1  od   (0, x)        [x ≥ 0]
```

Fig. 2. An example program that is a.s. terminating but with infinite expected termination time.

σ and $\mathbf{x}_{init}$ it holds, for each $1 \leq j \leq n$ and $j' < j$, that $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[\mathbf{X}_{i+1}[j] \mid \mathcal{R}_i] \leq X_i[j] - \epsilon$ on L^i_j and $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[\mathbf{X}_{i+1}[j] \mid \mathcal{R}_i] \leq X_i[j]$ on $L^i_{j'}$. This follows easily from the definition of $L^i_1, \ldots, L^i_n$ and from the definition of a transition being ϵ-ranked by η_j.

Since $\{\mathbf{X}_i\}^{\infty}_{i=0}$ is an ϵ-LexRSM for *Term*, from Theorem 4.8 it follows that $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\textit{Term} < \infty) = 1$, irrespective of σ and $\mathbf{x}_{init}$. □

For 1-dimensional RSMs, their existence imply both a.s. termination as well as positive termination. In contrast, we show that (Lin)LexRSMs can prove a.s. termination of programs whose expected termination time is infinite.

Example 4.9. Consider the program in Figure 2. By Example 4.7 and Theorem 4.8 it follows that the program terminates a.s., but the expected termination time is infinite: to see this, note that that the expected value of variable x upon reaching the second loop is $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 4 + \cdots = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots = \infty$ and that the time needed to get out of the second loop is equal to the value of x upon entering the loop.

Also, similarly to non-probabilistic programs, LexRSMs are more powerful than 1-dimensional LexRSMs already on single-loop programs.

Example 4.10. Consider the program in Figure 1 and, for simplicity, assume that initial values of variables are non-negative. One can rather straightforwardly check that there is no 1-dimensional Lin(Lex)RSM map for the program, given any invariant. Intuitively, this is because any such RSM map has to decrease as x decreases, since x is the only variable modified in the "if" branch of the non-deterministic choice, but x can increase unboundedly in the "else" branch, and this cannot be compensated by a bounded decrease of y. However, there is the following 3-dimensional $\frac{1}{2}$-LinLexRSM map $\overrightarrow{\eta}$: $\overrightarrow{\eta}(\ell_0) = (y+1, x+2, 3)$, $\overrightarrow{\eta}(\ell_1) = (y+1, x+2, 2)$, $\overrightarrow{\eta}(\ell_2) = \overrightarrow{\eta}(\ell_3) = (y+1, x+2, 1)$, $\overrightarrow{\eta}(\ell_4) = (y+1, x+3, 0)$, $\overrightarrow{\eta}(\ell_5) = (y+1, x+1, 0)$, $\overrightarrow{\eta}(\ell_6) = (y+\frac{1}{2}, x+2, 1)$, $\overrightarrow{\eta}(\ell_7) = (0, 0, 0)$. (Here $\overrightarrow{\eta}$ is supported by invariant I s.t. $I(\ell_0) = x \geq -2 \wedge y \geq -1$, $I(\ell_i) = x \geq 0 \wedge y \geq 0$ for $i = 1, \ldots, 6$, and $I(\ell_7) = \textit{true}$.) A bit more intricate LinLexRSM map exists also without the assumption that the initial valuation is non-negative.

5   ALGORITHMIC ASPECTS

In this section we describe a polynomial-time algorithm for synthesizing linear ϵ-LexRSM maps in affine probabilistic programs supported by a given linear invariant map I. A fully automated

method of termination proving can be obtained by pre-processing a given program using a suitable invariant generation tool, and inputting the program along with the computed invariants into our algorithm. This is also what we do in our experiments (Section 8).

The algorithm, based on iterative solving of linear constraints, is a generalization of an algorithm for finding lexicographic ranking functions in non-probabilistic programs [Alias et al. 2010]. Hence, we provide only a high-level description, focusing on the new aspects. For clarity of presentation, we show an algorithm which works for programs whose pCFGs have the property that a guard of each transition is a linear assertion (this particularly holds if each guard of while loop or a conditional branch in the program is a linear constraint). We also assume that we work with invariant maps such that each $I(\ell)$ is a polyhedron, i.e. a set defined by a linear assertion. This is exactly the setting in which non-probabilistic termination was considered in [Alias et al. 2010]. In the full version [Agrawal et al. 2017], we show how to extend the algorithm to general guards and invariants.

The main idea is to iteratively synthesize 1-dimensional *linear expression maps (LEMs)* that 1-rank a subset of generalized transitions. These maps form the individual components of the sought-after 1-LinLexRSM map. In each iteration, we start with a set U of the yet-unranked generalized transitions. We seek a 1-dimensional LEM which ranks the maximal number of elements of U, and is unaffected by the remaining elements of U (here, by ranking a generalized transition $\tilde{\tau}$ we mean ranking it from each configuration $(\ell, \mathbf{x})$, where ℓ is the source location of $\tilde{\tau}$ and $\mathbf{x} \in I(\ell)$). If no 1-dimensional LEM that would rank at least one element in U exists, then there is no LinLexRSM map for the program. Otherwise, we remove the newly ranked elements from U and continue into the next iteration, until U becomes empty. The process is summarized in Algorithm 1.

Hence, the main computational task of the algorithm is to check, for a given set of generalized transitions U, whether there exists a 1-dimensional LEM η such that:

(1) for each location $\ell \in L$ and all $\mathbf{x} \in I(\ell)$ it holds $\eta(\ell, \mathbf{x}) \geq 0$;
(2) for each $\tilde{\tau} \in U$ and each configuration $(\ell, \mathbf{x})$ where ℓ is the source of $\tilde{\tau}$ and $\mathbf{x} \in I(\ell) \cap \{\mathbf{x}' \mid \mathbf{x}' \models G(\tilde{\tau})\}$ we have that $\tilde{\tau}$ is unaffected by η from $(\ell, \mathbf{x})$; and
(3) there is $\tilde{\tau} \in U$ that is 1-ranked by η, from each configuration $(\ell, \mathbf{x})$ where ℓ is the source of $\tilde{\tau}$ and $\mathbf{x} \in I(\ell) \cap \{\mathbf{x}' \mid \mathbf{x}' \models G(\tilde{\tau})\}$; we then say that η ranks $\tilde{\tau}$ w.r.t. I.

Moreover, if such an LEM η exists, the algorithm has to find one that maximizes the number of gen. transitions in U ranked by it. Both these tasks can be accomplished by the standard method of linear constraints based on the use of Farkas's lemma, which was widely use for synthesis of termination proofs in both probabilistic and non-probabilistic programs [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016b; Colón and Sipma 2001; Podelski and Rybalchenko 2004a]. That is, the algorithm first constructs, for each location ℓ a *template* for η, i.e. an expression of the form $a_1^\ell x_1 + \cdots + a_{|V|}^\ell x_{|V|} + b^\ell$, where $x_1, \ldots, x_{|V|}$ are program variables and $a_1^\ell, \ldots, a_{|V|}^\ell, b^\ell$ are yet unknown coefficients. Supplying concrete values for all the unknown coefficients yields an LEM. Now the conditions (1) and (2) above can be expressed using linear constraints on the coefficients. More precisely, using the construction provided e.g. in [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016b] (which includes a use of the Farkas's lemma) we construct, in polynomial time, for each generalized transition $\tilde{\tau}$, a system of linear constraints $\mathcal{L}_{\tilde{\tau}}$ over set of variables $\{a_1^\ell, \ldots, a_{|V|}^\ell, b^\ell \mid \ell \in L\} \cup \{\epsilon_{\tilde{\tau}}\} \cup F$, where F is the set of fresh variables (not appearing in any template) and $\epsilon_{\tilde{\tau}}$ is constrained to be non-negative. Each solution of the system $\mathcal{L}_{\tilde{\tau}}$ yields a LEM which satisfies the constraints (1) and (2) for $\tilde{\tau}$. Moreover, each solution of $\mathcal{L}_{\tilde{\tau}}$ yields a LEM which $\epsilon_{\tilde{\tau}}$-ranks $\tilde{\tau}$. To find a LEM which satisfies all constraints (1)–(3) as well as maximizes the number

of 1-ranked elements of U it is sufficient to construct $\mathcal{L}_{\tilde{\tau}}$ for each $\tilde{\tau} \in U$ and solve the following linear program $\mathcal{LP}_U$:

$$\text{maximize} \sum_{\tilde{\tau} \in U} \epsilon_{\tilde{\tau}} \text{ subject to constraints}$$

$$\mathcal{L}_{\tilde{\tau}} \wedge 0 \leq \epsilon_{\tilde{\tau}} \leq 1; \quad \tilde{\tau} \in U$$

Each system $\mathcal{L}_{\tilde{\tau}}$ is constructed in such a way that if it admits a solution with some $\epsilon_{\tilde{\tau}}$ positive, then decreasing the value of $\epsilon_{\tilde{\tau}}$ in that solution to any non-negative value still yields a valid solution (this corresponds to the fact that if some transition is ϵ-ranked by η, than it is ϵ'-ranked by η for each $0 \leq \epsilon' \leq \epsilon$). Moreover, each solution where $\epsilon_{\tilde{\tau}}$ is positive can be rescaled into another solution in which $\epsilon_{\tilde{\tau}}$ is at least 1. It follows that if $\mathcal{LP}_U$ has at least one feasible solution, then it has an optimal solution in which each $\epsilon_{\tilde{\tau}}$ is either 0 or 1. If the system does not have a solution or all the $\epsilon_{\tilde{\tau}}$ are equal to zero then there is no LEM satisfying (1)–(3). Otherwise, the optimal solution of $\mathcal{LP}_U$ yields a LEM η which satisfies (1)–(3) and maximizes the number of 1-ranked elements of U.

This polynomial-time linear-programming step is used as a sub-procedure in Algorithm 1 for LinLexRSM synthesis.

Algorithm 1: Synthesis of LinLexRSMs for Apps

    input : An App $\mathcal{P}$ together with an invariant map I.
    output: A multi-dimensional LinLexRSM if it exists, otherwise "No LinLexRSM"
1 $U \leftarrow$ all generalized transitions of $\mathcal{C}_{\mathcal{P}}$
2 $d \leftarrow 0$
3 **while** U *is non-empty* **do**
4     $d \leftarrow d + 1$
5   construct and solve $\mathcal{LP}_U$
6   **if** $\mathcal{LP}_U$ *does not have a feasible solution or optimal value is* 0 **then**
7   **return** *No LinLexRSM*
8   **else**
9         $sol \leftarrow$ optimal solution of $\mathcal{LP}_U$
10         $\eta_d \leftarrow$ the LEM η induced by sol
11         $U \leftarrow U \setminus \{\tilde{\tau} \mid \epsilon_{\tilde{\tau}} = 1 \text{ in } sol\}$
12 **return** $(\eta_1, \ldots, \eta_d)$

Both soundness and relative completeness of Algorithm 1 can be proved by generalization of arguments presented in [Alias et al. 2010].

THEOREM 5.1. *Suppose that Algorithm 1 is run on an App $\mathcal{P}$ together with linear invariant map I. If the algorithm returns a d-dimensional LEM $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_d)$, then $\overrightarrow{\eta}$ is a 1-LinLexRSM map for $\mathcal{P}$ supported by I. Conversely, if the algorithm returns "No LinLexRSM", then for any $d' \in \mathbb{N}$ and $\epsilon > 0$ there is no d'-dimensional ϵ-LinLexRSM for $\mathcal{P}$ supported by I. If guards of all transitions in $\mathcal{C}_{\mathcal{P}}$ are linear assertions, then the algorithm runs in time polynomial in size of $\mathcal{P}$ and I.*

PROOF (KEY IDEAS). For soundness, let U_i denote the content of U just before the i-th iteration of the while loop. Then each $\tilde{\tau} \in U_i \setminus U_{i+1}$ is 1-ranked by η_i, for each $1 \leq i \leq d$. Since $U_d = \emptyset$, each generalized transition of $\mathcal{C}_{\mathcal{P}}$ is 1-ranked by some component of $\overrightarrow{\eta}$. Non-negativity of each η_i is

ensured directly by $\mathcal{LP}_{U_i}$. Hence, it remains to show that each $\tilde{\tau} \in U_i$ is unaffected by $\eta_1, \ldots, \eta_{i-1}$. But this follows from $U_i \subseteq U_{i-1} \subseteq \cdots \subseteq U_1$ and from the fact that each gen. transition in U_j is unaffected by η_j.

Proving completeness is more intricate; as pointed out in [Alias et al. 2010], one needs to show that the greedy strategy of selecting LEM that ranks maximal number of remaining transitions does not cut off some possible LinLexRSMs. In [Alias et al. 2010] the completeness of the greedy strategy is proved by using several geometric arguments that exploit the fact that the underlying programs are affine. The same geometric properties hold for our generalization to Apps (all ranking conditions in Definitions 4.2 and 4.3 are linear in program variables), so the result is easily transferable. The complexity argument is rather standard and we present it in the full version [Agrawal et al. 2017]. □

6 COMPOSITIONALITY OF RANKING SUPERMARTINGALES REVISITED

In this section we discuss compositional aspects of LexRSMs. We start with the one-dimensional case, and then present the multidimensional case.

6.1 One-Dimensional Compositional Proofs of Almost-Sure Termination

Compositionality in the context of termination proving means providing the proof of termination step-by-step, handling one loop at a time, rather than attempting to construct the proof (in our case, a LexRSM) at once [Kroening et al. 2010]. In the context of probabilistic programs, the work [Fioriti and Hermanns 2015] introduced a compositional notion of almost-sure termination proof based on the *probabilistic variant rule (V-rule),* which we explain in a more detail below. However, for the method to work, [Fioriti and Hermanns 2015] impose a technical *uniform integrability* condition, whose checking is hard to automatize. In this section we show that using our insights into LexRSMs we can obtain a different notion of a probabilistic V-rule which is sound without any additional assumptions, and which can be used to compositionally prove termination of programs that the previous method cannot handle.

Let $\mathcal{P}$ be a PP of the form **while** Ψ **do** $\mathcal{P}_{body}$ **od**, and let $\mathcal{C}_{\mathcal{P}}$ be the associated pCFG, whose set of locations we denote by L. We denote by $loops(\mathcal{P})$ the set of all locations of $\mathcal{C}_{\mathcal{P}}$ that belong to a sub-pCFG of $\mathcal{C}_{\mathcal{P}}$ corresponding to some nested loop of $\mathcal{P}$. We also define $slice(\mathcal{P})$ to be the set $L \setminus loops(\mathcal{P})$ of locations that do not belong to any nested sub-loop. A formal definition of both functions is given in the full version [Agrawal et al. 2017], we illustrate them in the following example.

Example 6.1. Consider the program $\mathcal{P}$ in Figure 4a and its associated pCFG. Then $loops(\mathcal{P}) = \{\ell_2, \ell_3\}$ and $slice(\mathcal{P}) = \{\ell_0, \ell_1, \ell_4, \ell^{out}\}$.

Given an invariant map I, we say that a 1-dimensional measurable map η for $\mathcal{C}_{\mathcal{P}}$ is ϵ-I-ranking/unaffecting in location ℓ, if for each $\mathbf{x} \in I(\ell)$ each generalized transition $\tilde{\tau}$ outgoing from ℓ is ϵ-ranked/unaffected by η from $(\ell, \mathbf{x})$.

We recall the notion of compositional ranking supermartingale as introduced in [Fioriti and Hermanns 2015]. We call it a PV supermartingale, as it is based on so called probabilistic variant rule. Due to differences in syntax and semantics, the definition is syntactically slightly different

from [Fioriti and Hermanns 2015], but the essence is the same. A measurable map η is propositionally linear, if each function $\eta(\ell)$ is of the form $1_{G_1} \cdot E_1 + \cdots 1_{G_k} \cdot E_k$, where each 1_{G_i} is an indicator function of some polyhedron and each E_i is a linear expression.

Definition 6.2 (PV-supermartingale [Fioriti and Hermanns 2015, Definition 7.1.]). A 1-dimensional propositionally linear map η is a PV supermartingale (PVSM) for a program $\mathcal{P}$ supported by an invariant map I if there exists $\epsilon > 0$ such that η is ϵ-I-ranking an I-non-negative (i.e. non-negative on all configurations within I) in each location $\ell \in slice(\mathcal{P})$ and I-unaffected in each $\ell \in loops(\mathcal{P})$.

The condition that η should be non-negative in locations of $slice(\mathcal{P})$ is not explicitly mentioned in [Fioriti and Hermanns 2015]. However, it is implicitly used in some of the proofs and one can easily construct an example where, if the non-negativity in $slice(\mathcal{P})$ is not required, the Theorem 6.3 below, which also comes from [Fioriti and Hermanns 2015], does not hold. Hence, we state the condition explicitly.

In [Fioriti and Hermanns 2015], they present an intriguing example showing that even if all nested loops were already proved to terminate a.s. and there is a PVSM for the program, then the program itself might not terminate a.s. This is in stark contrast with the non-probabilistic setting, where an analogous argument would be sound for proving termination. To remedy this, they impose a *uniform integrability* constraint on the PVSM under which a PVSM together with a proof of a.s. termination of each nested sub-loop of $\mathcal{P}$ entails termination of the whole program $\mathcal{P}$. Uniform integrability is a deep concept from probability and measure theory: a sequence $X_0, X_1, X_2, \ldots$ of random variables is uniformly integrable if for each $\delta > 0$ there exists a $K \in \mathbb{N}$ such that for all $n \geq 0$ it holds $\mathbb{E}[|X_n| \cdot 1_{X_n \geq K}] \leq \delta$. Apart from uniform integrability being somewhat restrictive in itself, already in [Fioriti and Hermanns 2015] it is argued that proving uniform integrability is beyond the capability of state-of-the-art automated theorem provers. As a substitute for these, [Fioriti and Hermanns 2015] introduce a type system that can be used to automatically prove uniform integrability of ranking supermartingales for a restricted class of programs. We do not repeat the precise definition of the typesystem here, we just say that a PVSM satisfying the condition imposed by the type system *typechecks* correctly. The following result about PVSMs was proved:

THEOREM 6.3 ([FIORITI AND HERMANNS 2015]). *Let $\mathcal{P}$ be a PP of the form **while** Ψ **do** $\mathcal{P}_{body}$ **od**. Assume that each nested loop of $\mathcal{P}$ terminates in finite expected time from each reachable configuration, and that there exists a PVSM for $\mathcal{P}$ that typechecks correctly. Then $\mathcal{P}$ terminates almost surely.*

The intricacies of uniform integrability are shown in the following example.

Example 6.4. Consider the two Apps in Figure 3. For the inner loop there exists (in both cases) a 1-dimensional linear ranking supermartingale whose value in each location is equal to $c + d_\ell$, where d_ℓ is a location-specific constant. Since the expected change of c in each loop step is -0.5, this is indeed a LRSM. Also, in both cases, a LEM of the form $x + d'_\ell$, again for some suitable location-specific constants d'_ℓ, is a PVSM for the outer loop. However, the variable x is uniformly integrable within the inner loop of the right program while for the left program this does not hold: we show this in the full version [Agrawal et al. 2017]. The example shows that proving uniform integrability requires intricate reasoning about quantitative behaviour of the program and dependencies of its variables. Moreover, as shown below, none of the two programs have a PVSM that typechecks.

Indeed, taking a closer look at typesystem in [Fioriti and Hermanns 2015], there are several reasons for typechecking of a PVSM to fail. The major ones are:

```
while  x ≥ 0  do
   c := 1 ;
   while  c ≥ 1  do
      if  prob ( 0.5 )  then
         x := 0 ;  c := 0
      else  x := 2x  fi ;
   od ;
   x := x − 1
od
```

```
while  x ≥ 0  do
   c := 1 ;
   while  c ≥ 1  do
      if  prob ( 0.5 )  then  x := 0
      else  x := 2x  fi ;
      if  prob ( 0.5 )  then  c := 0
      else  skip  fi
   od ;
   x := x − 1
od
```
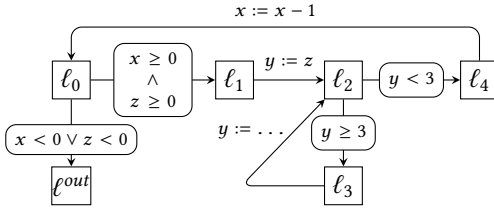
Fig. 3. Examples of programs with (right) and without (left) uniformly integrable PVSMs.

```
while  x ≥ 0  and  z ≥ 0  do
   y := z ;
   while  y ≥ 3  do
      y := y+sample(Uniform[−3, 1])
   od ;
   x := x − 1
od
```



```
while  x ≥ 0  do
   while  y ≥ 0  do
      z := x ;
      while  z ≥ 0  do
         z := z − 1 ;
         x := x − 1
      od ;
      y := y − 1
   od ;
   x := x+sample(Uniform[−3, 1])
od
```

(b) Program where the outer loop does not have
a PVSM that typechecks.

(a) Example for slicing illustration: program and
its pCFG.

Fig. 4. Program illustrations.

(1) A PVSM η for PP $\mathcal{P}$ will not typecheck if $\mathcal{P}$ has a nested loop in which the value of η can change unboundedly in a single step (see Figure 3).

(2) A PVSM η for PP $\mathcal{P}$ will not typecheck if $\mathcal{P}$ has a nested loop which itself has a nested loop in which some variable appearing in some expression of η is modified, see Figure 4b.[3]

Thus, the typechecking algorithm may rule out programs where the termination-controlling variable represents e.g. a length of an array, which can be doubled/halved in some sub-program due to (de)allocation, merging, or splitting. To overcome the typechecking, we use the results on LexRSMs to define a new notion of compositional ranking supermartingales, which we call *non-negative*

---

[3]Both these statements regarding typechecking failure are somewhat simplified, even in these two cases the PVSM might sometimes typecheck correctly, in case where the nested loops are followed by assignments which completely overwrite the effect of these loops, e.g. if the program in Figure 4b contained an assignment $x := 0$. However, the statements intuitively summarize the major reasons for typechecking failure.

compositional (NC) supermartingales. For the sake of generality, we allow NC martingales to be general measurable maps, not necessarily propositionally linear.

Definition 6.5. A 1-dimensional measurable map η is an NC supermartingale (NCSM) for a program $\mathcal{P}$ supported by an invariant map I if there exists $\epsilon > 0$ such that η is:

(1) non-negative in each $(\ell, \mathbf{x})$ where ℓ is a location of $C_{\mathcal{P}}$ and $\mathbf{x} \in I(\ell)$;
(2) ϵ-I-ranking in each location $\ell \in slice(\mathcal{P})$; and
(3) I-unaffecting in each $\ell \in loops(\mathcal{P})$.

A (propositionally) linear NCSM (LinNCSM) is a NCSM which is also a (propositionally) linear expression map.

We can prove that NCSMs are a sound method for proving a.s. termination in a compositional way, without any additional assumptions.

THEOREM 6.6. *Let $\mathcal{P}$ be a PP of the form* **while** Ψ **do** $\mathcal{P}_{body}$ **od**. *Assume that each nested loop of $\mathcal{P}$ terminates almost surely from each reachable configuration, and that there exists a NCSM for $\mathcal{P}$ supported by some invariant map. Then $\mathcal{P}$ terminates almost surely from each initial configuration.*

PROOF (KEY IDEA). Let $\{X_i\}_{i=0}^{\infty}$ be a stochastic process returning the value of NCSM η in step i. Then $\{X_i\}_{i=0}^{\infty}$ is a (non-strict) 1-dimensional ϵ-LexRSM for the termination time of the program, for some $\epsilon > 0$. Since all sub-loops of $\mathcal{P}$ terminate, with probability one each run has level < 2 in infinitely many steps. From Theorem 3.4 it follows that $\mathbb{P}_{\mathbf{x}_{init}}^{\sigma}(Term < \infty) = 1$, for all σ and $\mathbf{x}_{init}$. □

For Apps, synthesizing linear NCSM entails synthesizing sufficient program invariants (for which there is a good automated tool support [Feautrier and Gonnord 2010]) and encoding the ranking, unaffection, and non-negativity conditions into a collection of linear constraints (as for general LinLexRSMs in Section 5). Figures 3 and 4b show instances where attempts to prove a.s. termination via PVSMs fail while proofs via LinNCSMs work.

Example 6.7. All programs in Figures 3 and 4b have LinNCSMs for all their loops, which shows that the programs terminate a.s. In Figure 3, for both programs the inner loops have LinNCSMs of the form $c + d_\ell$, for d_ℓ a location-specific constant, while the outer loops have LinNCSMs of the form $x + d'_\ell$. In Figure 4b the program similarly has LinNCSMs defined, proceeding from the innermost loop and neglecting the location-specific constants, by variables z, y, x.

Using LinNCSMs, we can devise a simple algorithm for compositional proving of almost-sure termination of Apps, which is pictured in Algorithm 2.

The soundness of the algorithm follows from Theorems 6.6 and 6.3. We can use the PVSM-based algorithm of [Fioriti and Hermanns 2015] in the case when LinNCSM-based proof fails. Hence, Algorithm 2 can compositionally prove a.s. termination of strictly larger class of programs than the PVSM-based algorithm alone.

Example 6.8. A compositional proof of almost-sure termination for the left program in Figure 3 would go as follows: First, the algorithm takes the inner loop and synthesizes its one-dimensional RSM of the form $c + d_\ell$ (clearly, c is non-negative inside the loop and its value decreases by an average of 0.5 in each loop execution). This proves a.s. termination of the inner loop. Then, the algorithm takes the whole program and synthesizes a function of the form $x + d'_\ell$, which, in addition to being a one-dimensional RSM for the outer loop, is also non-negative and non-increasing (on

---

Algorithm 2: Compositional Termination Proving

---

    input : An APP $\mathcal{P}$ together with an invariant map I.

1 **for** $i \leftarrow$ *depth of loop nesting in* $\mathcal{P}$ **to** 0 **do**

2 | **foreach** *sub-loop* $\mathcal{P}'$ *of* $\mathcal{P}$ *nested i levels below the main loop* **do**

3 | | $\mathcal{L} \leftarrow$ system of lin. constraints encoding the existence of LinNCSM for $\mathcal{P}'$ supported by I

4 | | **if** $\mathcal{L}$ *not solvable* **then**

5 | | | check termination of $\mathcal{P}$ via PVSM-algorithm from [Fioriti and Hermanns 2015]

6 **return** *"$\mathcal{P}$ terminates a.s."*

---

average) in the inner loop, i.e. it is a NCSM for the program. Paired with the already proved a.s. termination of the inner loop, this proves a.s. termination of the whole program.

To summarize, the novelty of NCSMs is the following:

(1) NCSMs allow compositional, fully automated proofs of a.s. termination without the need for reasoning about uniform integrability.
(2) LinNCSMs are capable of proving a.s. termination of programs for which no uniformly integrable PVSMs exist (and hence the PVSM method cannot be used at all on such programs).
(3) LinNCSMs are capable of proving a.s. termination of programs for which the PVSM method cannot be applied in an automated way, due to failure of the typechecking procedure.
(4) Unlike the PVSM method, NCSMs do not require the nested loops to terminate in finite expected time in order to prove a.s. termination.

6.2 Multidimensional Compositional Ranking

Above, we defined NCSMs as one-dimensional objects, to make them analogous to PVSMs for better comparison. However, we can also define a multi-dimensional version of NCSMs, to take advantage of the fact that LexRSMs can handle loops for which no 1-dimensional linear RSM exists (see Example 4.10). We say that, given an invariant map I, an n-dimensional measurable map is ϵ-I-ranking in a location ℓ if for each $\mathbf{x} \in I(\ell)$ and each gen. transition $\tilde{\tau}$ outgoing from ℓ there exists $1 \le j \le n$ such that $\tilde{\tau}$ is ϵ-ranked by η_j from $(\ell, \mathbf{x})$ and for each $j' < j$ we have that $\tilde{\tau}$ is unaffected by $\eta_{j'}$.

Definition 6.9. An n-dimensional measurable map $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$ is an NC supermartingale (NCSM) for a program $\mathcal{P}$ supported by an invariant map I if there exists $\epsilon > 0$ such that:

(1) for each $1 \le i \le n$, η_i is non-negative in each location of $C_{\mathcal{P}}$;
(2) $\overrightarrow{\eta}$ is ϵ-I-ranking in each location $\ell \in slice(\mathcal{P})$; and
(3) for each $1 \le i \le n$, η_i is unaffected in each $\ell \in loops(\mathcal{P})$.

An n-dimensional linear NCSM (LinNCSM) is an n-dimensional NCSM which is also a linear expression map. The following theorem can be proved in essentially the same way as Theorem 6.6.

THEOREM 6.10. *Let $\mathcal{P}$ be a PP of the form **while** Ψ **do** $\mathcal{P}_{body}$ **od**. Assume that each nested loop of $\mathcal{P}$ terminates almost surely from each reachable configuration, and that there exists a NCSM for $\mathcal{P}$ supported by some invariant map. Then $\mathcal{P}$ terminates almost surely.*

For Apps, we can generalize Algorithm 2 by changing line 2 to "check existence of a multi-dimensional LinNCSM for $\mathcal{P}'$" and line 3 to "**if** a multi-dimensional LinNCSM does not exist." The check of existence of a multi-dimensional LinNCSM for $\mathcal{P}'$ can be done by algorithm presented in Section 5, modified so as to only pursue ranking for generalized transitions outgoing from locations belonging to $slice(\mathcal{P}')$. (I.e., only these gen. transitions have ϵ_τ included in the objective function, and algorithm terminates once all such transitions are ranked.)

7 BOUNDS ON EXPECTED TERMINATION TIME

As shown in Example 4.9, already LinLexRSM maps are capable of proving almost-sure termination of programs whose expected termination time is infinite. However, it is often desirable to obtain bounds on expected runtime of a program. In this section, we present a LexRSM-based method for obtaining bounds on expected runtime using a restricted class of LexRSMs.

As in the case of a.s. termination we start with general mathematical statement about LexRSMs. We define a restricted class of strict LexRSMs with *bounded expected conditional increase* property. Recall from Definition 3.3 that strict LexRSM for a stopping time T is characterized by the possibility to a.s. partition, for each $i \in \mathbb{N}_0$, the set $\{\omega \in \Omega \mid T(\omega) > i\}$ into n sets $L_1^i, \ldots, L_d^i$ such that, intuitively, on L_j^i the conditional expectation of $\mathbf{X}_{i+1}[j]$ given $\mathcal{F}_i$ is smaller than $\mathbf{X}_i[j]$, and for all $j' < j$, on $L_{j'}^i$ the conditional expectation of $\mathbf{X}_{i+1}[j]$ given $\mathcal{F}_i$ is no larger than $\mathbf{X}_i[j]$. This leaves the opportunity of conditional expectation of $\mathbf{X}_{i+1}[j]$ being larger than $\mathbf{X}_i[j]$ on $L_{j''}^i$ with $j'' > j$. The expected conditional increase (ECI) property bounds the possibility of this increase.

Definition 7.1. Let $\{\mathbf{X}_i\}_{i=0}^\infty$ be an n-dimensional strict LexRSM for some stopping time T, defined w.r.t. some filtration $\{\mathcal{F}_i\}_{i=0}^\infty$. We say that $\{\mathbf{X}_i\}_{i=0}^\infty$ has $\mathbf{c}$-*bounded expected conditional increase (ECI)*, for some non-negative vector $\mathbf{c} \in \mathbb{R}^d$, if there exists an instance $(\{\mathbf{X}_i\}_{i=0}^\infty, \{L_1^i, \ldots, L_{n+1}^i\}_{i=0}^\infty)$ of the strict LexRSM (i.e. $L_{n+1}^i = \emptyset$ for all i) such that for each $i \in \mathbb{N}_0$ and each $1 \leq j \leq n$ it holds $\mathbb{E}[\mathbf{X}_{i+1}[j] \mid \mathcal{F}_i] \leq \mathbf{X}_i[j] + \mathbf{c}[j]$ on $L_{j''}^i$, for all $j'' > j$ (here $L_1^i, \ldots, L_n^i$ are as in Definition 3.3).

For strict LexRSMs with $\mathbf{c}$-bounded ECI we have the following result. For simplicity, we formulate the result for 1-LexRSMs, though it is easy to prove analogous result for general ϵ-LexRSMs, $\epsilon > 0$, at the cost of obtaining less readable formula.

THEOREM 7.2. *Let $\{\mathbf{X}_i\}_{i=0}^\infty$ be an n-dimensional strict LexRSM with c-bounded ECI for some stopping time T. Then $\mathbb{E}[T] \leq \sum_{j=1}^n \mathbb{E}[\mathbf{X}_0[j]] \cdot (1 + \sum_{j < j' \leq j'' \leq n} \prod_{k=j'}^{j''} \mathbf{c}[k])$.*

PROOF. Fix an instance $(\{\mathbf{X}_i\}_{i=0}^\infty, \{L_1^i, \ldots, L_{n+1}^i\}_{i=0}^\infty)$ satisfying Definition 7.1. Denote $\sharp lev_j(\omega)$ the number of steps i in which $\omega \in L_j^i$. Since by Theorem 3.4 the existence of strict LexRSM entails $\mathbb{P}(T < \infty) = 1$, the value $\sharp lev_j(\omega)$ is a.s. finite for all $1 \leq j \leq n$. We prove that for each $1 \leq j \leq n$ it holds $\mathbb{E}[\sharp lev_j] \leq \mathbf{c}[j] \cdot \left(\sum_{j' < j} \mathbb{E}[\sharp lev_{j'}] \right) + \mathbb{E}[\mathbf{X}_0[j]]$. Since $T(\omega) = \sum_{1 \leq j \leq n} \sharp lev_j(\omega)$, for each $\omega \in \Omega$ (and hence, due to linearity of expectation $\mathbb{E}[T] = \sum_{1 \leq j \leq n} E[\sharp lev_j]$), the statement of the Theorem follows by an easy induction.

To prove the required inequality, let $\sharp_k lev_j(\omega)$ be the number of steps i *within the first k* steps such that $\omega \in L_j^i$. We prove, by induction on k, that for each k it holds $\mathbb{E}[\sharp_k lev_j] \leq \mathbf{c}[j] \cdot \left(\sum_{j' < j} \mathbb{E}[\sharp_k lev_{j'}] \right) + \mathbb{E}[\mathbf{X}_0[j]] - \mathbb{E}[\mathbf{X}_k[j]]$. Once this is proved, the desired inequality follows by taking k to ∞, since $\lim_{k \to \infty} \mathbb{E}[\sharp_k lev_j] = \mathbb{E}[\sharp lev_j]$ and $\lim_{k \to \infty} \mathbb{E}[\mathbf{X}_k[j]] \geq 0$. The inductive proof is somewhat technical and deferred to [Agrawal et al. 2017]. □

To transfer this mathematical result to probabilistic programs, we want to impose a restriction on LexRSM maps that ensures that all components of a LexRSM map have, from each reachable configuration, an expected one-step increase of at most c. Here c can be a constant, but it can also be a value that depends on the initial configurations: this is to handle cases where some variables are periodically reset to a value related to the initial variable values, such as variable y in Figure 4a. To this end, let $\mathcal{P}$ be a PP with a pCFG $\mathcal{C}_{\mathcal{P}}$ and let $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$ be an n-dimensional 1-LexRSM map for $\mathcal{P}$. Consider an n-dimensional vector $\bar{\mathbf{c}} = (\bar{c}_1, \ldots, \bar{c}_n)$ whose each component is an expression over variables of $\mathcal{P}$. We say that $\overrightarrow{\eta}$ has $\bar{\mathbf{c}}$-bounded ECI w.r.t. invariant map I if the following holds for each initial configuration $(\ell_{init}, \mathbf{x}_{init})$ with $\mathbf{x}_{init} \in \Xi_{init}$: for each configuration $(\ell, \mathbf{x})$ with $\mathbf{x} \in I(\ell)$ and generalized transition $\tilde{\tau}$ of $\mathcal{C}_{\mathcal{P}}$ outgoing from $(\ell, \mathbf{x})$ it holds that if j is the smallest index such that $\tilde{\tau}$ is 1-ranked by η_j from $(\ell, \mathbf{x})$, then for all $j' > j$ the gen. transition $\tilde{\tau}$ is f-ranked by $\eta_{j'}$ from $(\ell, \mathbf{x})$, where $f = -c_{j'}(\mathbf{x}_{init})$. From Theorem 7.2 we have the following:

COROLLARY 7.3. *Let $\mathcal{P}$ be a probabilistic program. Assume that there exists an n-dimensional ϵ-LexRSM map $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_d)$ for $\mathcal{P}$ supported by some invariant map I, such that $\overrightarrow{\eta}$ has $\bar{\mathbf{c}}$-bounded ECI (w.r.t. I) for some vector of expressions $\bar{\mathbf{c}} = (\bar{c}_1, \ldots, \bar{c}_n)$. Then under each scheduler σ and for each initial valuation of program variables $\mathbf{x}_{init} \in \Xi_{init}$ it holds $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] \leq 2 \cdot \sum_{j=1}^{n} \eta_j(\ell_{init}, \mathbf{x}_{init}) \cdot (1 + \sum_{j < j' \leq j'' \leq n} \prod_{k=j'}^{j''} \bar{c}_k(\mathbf{x}_{init}))$.*

REMARK 3. *Somewhat crisper statement of our result (weaker than Corollary 7.3) is as follows: Consider the class of programs where each assignment is of the form $x := x + c$, where c is a real constant; or $x := a_0 \cdot init(x_0) + \ldots + a_n \cdot init(x_n) + c$, where $a_0, \ldots, a_n, c$ are real constants and $init(x_i)$ is the initial value of variable x_i. For such programs, if a LinLexRSM exists, then the LinLexRSM implies a (computable) bound B on expected termination time which is a polynomial function of the initial variable values.*

REMARK 4. *Previous works testify that analogous "bounded expected change" constraints naturally emerge when one aims to obtain additional information about termination time of probabilistic programs [Chatterjee et al. 2016b, 2017; Fioriti and Hermanns 2015]. However, all previous works consider bounded expected change to use well-known tail-bound inequalities for martingales such as Azuma's inequality to obtain concentration bounds on termination time, but not asymptotic bounds on expected termination time. In contrast, the major novelty of our bounded ECI condition is that we show it can be used to obtain asymptotic bounds on expected termination time, and in particular, they can be used to obtain polynomial bounds on expected runtime using* linear *supermartingales.*

Example 7.4. Consider the program in Figure 4a, with initial condition $x_{init} \geq 0$. There is an invariant map I with $I(\ell_0) = x \geq -1$, $I(\ell_1) = I(\ell_4) = x \geq 0 \wedge z \geq 0$, $I(\ell_2) = I(\ell_3) = x \geq 0 \wedge y \geq 0 \wedge z \geq 0$, and $I(\ell^{out}) = true$. Further, there exists the following 3-dimensional 1-LinLexRSM map $\overrightarrow{\eta}$ supported by I: $\overrightarrow{\eta}(\ell_0) = (2x + 3, 0, 2)$, $\overrightarrow{\eta}(\ell_1) = (2x + 3, 0, 1)$, $\overrightarrow{\eta}(\ell_2) = (2x + 2, y, 1)$, $\overrightarrow{\eta}(\ell_3) = (2x + 2, y, 0)$, $\overrightarrow{\eta}(\ell_4) = (2x + 2, 0, 0)$, and $\overrightarrow{\eta}(\ell^{out}) = (0, 0, 0)$. It is easy to check that $\overrightarrow{\eta}$ is has $(0, z, 2)$-bounded ECI. Applying Corollary 7.3 yields $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] \leq 2 \cdot (2x_{init} + 3) \cdot (3 + 3z_{init}) + 0 + 4 = 12x_{init}z_{init} + 12x_{init} + 9z_{init} + 22$. Note that this quadratic bound is asymptotically optimal for the program.

REMARK 5 (POLYNOMIAL BOUNDS). *Now consider that we have synthesized a 1-LexRSM map $\overrightarrow{\eta}$ and we want to check if there exists $\bar{\mathbf{c}}$ such that $\overrightarrow{\eta}$ has $\bar{\mathbf{c}}$-bounded ECI. In the linear setting (i.e. the program is an APP, maps η and I are linear, and we seek $\bar{\mathbf{c}}$ which is a vector of affine expressions) we can encode the existence of $\bar{\mathbf{c}}$ into a system of linear inequalities in a similar way as the existence of linear LexRSM maps was encoded in Section 4 (some static analysis might be required to identify "resets to the initial value"). I.e., we set up a linear template with unknown coefficients for each component of $\bar{\mathbf{c}}$ and using Farkas's lemma we set up a system of linear constraints, which includes the unknown coefficients as*

variables, encoding the fact that $\overrightarrow{\eta}$ has $\bar{c}$-bounded ECI. In this way, we can check in polynomial time if
Corollary 7.3 can be applied to $\overrightarrow{\eta}$, and if yes, we can synthesize the witness vector $\bar{c}$. Since $\bar{c}$ consists of
affine expressions, Corollary 7.3 provides a polynomial (in the size of initial variable valuation) upper
bound on expected runtime.

8 EXPERIMENTAL RESULTS

We implemented the algorithm of Section 5 (in C++) and present two sets of experimental results.
For all the experimental results we use the tool Aspic [Feautrier and Gonnord 2010] for invariant
generation, and our algorithm requires linear-programming solver for which we use CPlex [cpl
2010]. The results were obtained on the following platform: Ubuntu16.04, 7.7GB, Intel-Core i3-4130
CPU 3.40GHz QuadCore 64-bit.

Benchmarks adapted from previous work. For developing our benchmark suite, we build on a bench-
mark suite of non-probabilistic programs from [Alias et al. 2010] (see also [Chawdhary et al. 2008]).
To the best of our knowledge, there are no abstraction tools available for probabilistic programs.
Hence, we construct our benchmark suite by taking abstractions of non-probabilistic programs in
the benchmark suite of [Alias et al. 2010] and extending these programs with probabilities. Given
these non-probabilistic programs, we obtain probabilistic programs in two ways: (a) *probabilistic
loops,* where the existing while loops are made probabilistic by executing the existing statements
with probability 1/2, and with remaining probability executing skip statements; and (b) *probabilistic
assignments,* where the existing assignments are perturbed uniformly in range $[-1, 1]$ (i.e., we
consider additional variables whose value is, in each loop iteration, generated by probabilistic
assignment uniformly in the range $[-1, 1]$, and we add such variable to the RHS of an existing
assignment). We report our results on twenty five benchmarks in Table 1 (we consider around fifty
benchmark examples and the results on the remaining ones, along with detailed description of the
table, are presented in the full version [Agrawal et al. 2017]). The experimental results show that
the time taken by our approach is almost always less than 1/10-th of a second. In the table, along
with the benchmark name, and time in seconds, we show whether a solution exists or not (i.e.,
whether linear lexicographic RSMs exist or not), and if the solution exists we present the dimension
of the lexicographic RSM we obtain. The final two columns of the table represent whether the
non-probabilistic program is extended with probabilistic loops and/or probabilistic assignments.

Synthetic examples of large programs. The programs in the aforementioned benchmark suite have
between 10-100 lines of code. To test the how does our approach scale with larger codes we consider
synthetic examples of large probabilistic programs generated as follows. Given n Boolean variables,
we consider probabilistic while loops, with some nondeterministic conditional branches, and
generate all possible 2^n if conditions based on the Boolean variables. Hence, given n variables we
have probabilistic programs of size $O(2^n)$. For such programs, we first run an invariant generation
tool, followed by our algorithm. In all these examples lexicographic RSMs exist, and have dimension
at most 3. Even for programs with around 12K lines of code the total time taken is around one hour,
where the invariant generation (i.e., running Aspic) takes the maximum time, and our algorithm
requires around two minutes. The results are presented in Table 2 where we present the number of
variables, then lines of code, followed by the time taken for invariant generation by Aspic, then the
time taken by our algorithm, and finally the total time.

Table 1. Experimental results for benchmarks from [Alias et al. 2010] extended with probabilistic loops/assignments.

Benchmark	Time (s)	Solution	Dimension	Prob. loops	Prob. Assignments
alain	0.11	yes	3	yes	yes
catmouse	0.08	yes	2	yes	yes
counterex1a	0.10	no	-	no	no
counterex1c	0.11	yes	2	yes	yes
easy1	0.09	yes	1	yes	yes
exmini	0.09	yes	2	yes	yes
insertsort	0.10	yes	3	yes	yes
ndecr	0.09	yes	2	yes	yes
perfect	0.11	yes	3	yes	yes
perfect2	0.10	yes	3	yes	no
	0.11	no	-	yes	yes
real2	0.09	no	-	no	no
realbubble	0.22	yes	3	yes	yes
realselect	0.11	yes	3	yes	yes
realshellsort	0.09	no	-	yes	no
serpent	0.10	yes	3	yes	yes
sipmabubble	0.10	yes	3	yes	yes
speedDis2	0.09	no	-	no	no
speedNestedMultiple	0.10	yes	3	yes	yes
speedpldi2	0.09	yes	2	yes	yes
speedpldi4	0.09	yes	2	yes	yes
speedSimpleMultipleDep	0.09	no	-	no	no
speedSingleSingle2	0.12	yes	2	yes	no
	0.10	no	-	yes	yes
unperfect	0.10	yes	2	yes	no
	0.16	no	-	yes	yes
wcet1	0.11	yes	2	yes	yes
while2	0.10	yes	3	yes	yes

9 RELATED WORK

Probabilistic programs and termination. In early works the termination for concurrent probabilistic programs was studied as fairness [Sharir et al. 1984], which ignored precise probabilities. For countable state space a sound and complete characterization of almost-sure termination was presented in [Hart and Sharir 1985], but nondeterminism was absent. A sound and complete method for proving termination of finite-state programs was given in [Esparza et al. 2012]. For probablistic programs with countable state space and without nondeterminism, the *Lyapunov ranking functions* provide a sound and complete method to prove positive termination [Bournez and Garnier 2005; Foster 1953]. For probabilistic programs with nondeterminism, but restricted to discrete probabilistic choices, the termination problem was studied in [McIver and Morgan 2004, 2005]. The RSM-based (ranking supermatingale-based) approach extending ranking functions was first presented in [Chakarov and Sankaranarayanan 2013] for probabilistic programs without non-determinism, but with real-valued variables, and its extension for probabilistic programs with

Table 2. Experimental results for synthetic examples.

Variables	LOC	Inv Time (s)	Our Time (s)	Total Time (s)
2	20	0.06	0.03	0.08
3	32	0.07	0.03	0.09
4	56	0.08	0.04	0.11
5	104	0.14	0.06	0.19
6	200	0.36	0.10	0.46
7	392	1.31	0.3	1.61
8	776	7.56	0.7	8.25
9	1544	33.07	2.5	35.57
10	3080	164.09	8.77	172.86
11	6152	817.92	35.37	853.29
12	12296	4260.96	145.18	4406.14

non-determinism has been studied in [Chatterjee and Fu 2017; Chatterjee et al. 2016a,b, 2017; Fioriti and Hermanns 2015; McIver and Morgan 2016]. Supermartingales were also considered for other liveness and safety properties [Barthe et al. 2016a; Chakarov et al. 2016]. While all these results deeply clarify the role of RSMs for probabilistic programs, the notion of lexicographic RSMs to obtain an efficient approach for termination analysis for probabilistic programs has not been studied before, which we consider in this work. A compositional rule for proving almost-sure termination was studied in [Fioriti and Hermanns 2015] under the uniform integrability assumption. In [McIver and Morgan 2005], a soundness of the probabilistic variant rule is proved for programs with finitely many configurations.

Other approaches. Besides RSMs, other approaches have also been considered for probabilistic programs. Logical calculi for reasoning about properties of probabilistic programs (including termination) were studied in [Feldman 1984; Feldman and Harel 1982; Kozen 1981, 1983] and extended to programs with demonic non-determinism in [Gretz et al. 2014; Kaminski et al. 2016; Katoen et al. 2010; McIver and Morgan 2004, 2005; Olmedo et al. 2016]. However, none of these approaches is readily automatizable. A sound approach [Monniaux 2001] for almost-sure termination is to explore the exponential decrease of probabilities upon bounded-termination through abstract interpretation [Cousot and Cousot 1977]. A method for a.s. termination of weakly finite programs (where number of reachable configurations is finite from each initial configuration) based on *patterns* was presented in [Esparza et al. 2012].

Non-probabilistic programs. Termination analysis of non-probabilistic programs has also been extensively studied [Bradley et al. 2005a,b; Colón and Sipma 2001; Cook et al. 2006, 2011; Kroening et al. 2010; Lee et al. 2001; Podelski and Rybalchenko 2004a,b; Sohn and Gelder 1991]. Ranking functions are at the heart of the termination analysis, and lexicographic ranking functions have emerged as one of the most efficient and practical approaches for termination analysis [Alias et al. 2010; Bradley et al. 2005a; Cook et al. 2013; Gonnord et al. 2015], being used e.g. in the prominent T2 temporal prover [Brockschmidt et al. 2016]. In this work, we extend lexicographic ranking functions to probabilistic programs, and present lexicographic RSMs for almost-sure termination analysis of probabilistic programs with non-determinism. Theoretical complexity of synthesizing lexicographic ranking functions in non-probabilistic programs was studied in [Ben-Amram and Genaim 2013, 2015].

10 CONCLUSION AND FUTURE WORK

In this work we considered lexicographic RSMs for termination analysis of probabilistic programs with non-determinism. We showed that lexicographic RSMs present a sound approach for almost-sure termination, that they enable compositional reasoning about termination, and can be used, under certain conditions, to obtain polynomial bounds on expected runtime. There are several interesting directions of future work. First, lexicographic ranking functions have been considered in several works to provide different practical methods for analysis of non-probabilistic programs. While our work presents the foundations of lexicographic RSMs for probabilistic programs, it would be interesting to extend other practical methods based on lexicographic ranking functions to lexicographic RSMs in order to provide better scalability and termination analysis of real-world and large-scale probabilistic code. Developing supporting tools for such an analysis, such as automated tools for computing abstractions of probabilistic programs, would be an indispensable part of such an effort. Second, while our algorithmic approaches focus on the linear case, it would be interesting to consider non-linear functions in the future.

REFERENCES

2010. IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2017. Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs. *CoRR* abs/1709.04037 (2017). http://arxiv.org/abs/1709.04037

Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. 2010. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *Proceedings of the 17th International Conference on Static Analysis (SAS'10)*. Springer-Verlag, Berlin, Heidelberg, 117–133. http://dl.acm.org/citation.cfm?id=1882094.1882102

R.B. Ash and C. Doléans-Dade. 2000. *Probability and Measure Theory*. Harcourt/Academic Press.

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. The MIT Press, Cambridge, Massachusetts. 984 pages.

Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016a. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer International Publishing, 43–61. https://doi.org/10.1007/978-3-319-41528-4_3

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. Proving Differential Privacy via Probabilistic Couplings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*. ACM, New York, NY, USA, 749–758. https://doi.org/10.1145/2933575.2934554

Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin Pierce. 2016c. Programming Language Techniques for Differential Privacy. *ACM SIGLOG News* 3, 1 (Feb. 2016), 34–53. https://doi.org/10.1145/2893582.2893591

Amir M. Ben-Amram and Samir Genaim. 2013. On the Linear Ranking Problem for Integer Linear-constraint Loops. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*. ACM, New York, NY, USA, 51–62. https://doi.org/10.1145/2429069.2429078

Amir M. Ben-Amram and Samir Genaim. 2015. Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions. In *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, Daniel Kroening and Corina S. Păsăreanu (Eds.). Springer International Publishing, 304–321. https://doi.org/10.1007/978-3-319-21668-3_18

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

Olivier Bournez and Florent Garnier. 2005. Proving Positive Almost-Sure Termination. In *RTA*. 323–337.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2005a. Linear Ranking with Reachability. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings (Lecture Notes in Computer Science)*, Kousha Etessami and Sriram K. Rajamani (Eds.), Vol. 3576. Springer, 491–504. https://doi.org/10.1007/11513988_48

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2005b. The Polyranking Principle. In *ICALP*. 1349–1361.

Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf, and Nir Piterman. 2016. T2: Temporal Property Verification. In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, Marsha Chechik and Jean-François Raskin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 387–393. https://doi.org/10.1007/978-3-662-49674-9_22

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

Aleksandar Chakarov, Yuen-Lam Voronin, and Sriram Sankaranarayanan. 2016. Deductive Proofs of Almost Sure Persistence and Recurrence Properties. In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, Marsha Chechik and Jean-François Raskin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 260–279. https://doi.org/10.1007/978-3-662-49674-9_15

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

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

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

Krishnendu Chatterjee, Petr Novotný, and Djordje Žikelić. 2017. Stochastic Invariants for Probabilistic Termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, New York, NY, USA, 145–160. https://doi.org/10.1145/3009837.3009873

Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv, and Hongseok Yang. 2008. Ranking Abstractions. In *Programming Languages and Systems: 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, Sophia Drossopoulou (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 148–162. https://doi.org/10.1007/978-3-540-78739-6_13

Michael Colón and Henny Sipma. 2001. Synthesis of Linear Ranking Functions. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings (Lecture Notes in Computer Science)*, Tiziana Margaria and Wang Yi (Eds.), Vol. 2031. Springer, 67–81. https://doi.org/10.1007/3-540-45319-9_6

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006. Termination Proofs for Systems Code. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06)*. ACM, New York, NY, USA, 415–426. https://doi.org/10.1145/1133981.1134029

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2011. Proving program termination. *Commun. ACM* 54, 5 (2011), 88–98. https://doi.org/10.1145/1941487.1941509

Byron Cook, Abigail See, and Florian Zuleger. 2013. Ramsey vs. Lexicographic Termination Proving. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*. Springer-Verlag, Berlin, Heidelberg, 47–61. https://doi.org/10.1007/978-3-642-36742-7_4

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, Robert M. Graham, Michael A. Harrison, and Ravi Sethi (Eds.). ACM, 238–252. https://doi.org/10.1145/512950.512973

Devdatt Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms* (1st ed.). Cambridge University Press, New York, NY, USA.

Javier Esparza, Andreas Gaiser, and Stefan Kiefer. 2012. Proving Termination of Probabilistic Programs Using Patterns. In *CAV*. 123–138.

Paul Feautrier and Laure Gonnord. 2010. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. *Electronic Notes in Theoretical Computer Science* 267, 2 (2010), 3 – 13. https://doi.org/10.1016/j.entcs.2010.09.014 Proceedings of the Tools for Automatic Program AnalysiS (TAPAS).

Yishai A. Feldman. 1984. A decidable propositional dynamic logic with explicit probabilities. *Information and Control* 63, 1 (1984), 11–38. https://doi.org/10.1016/S0019-9958(84)80039-X

Yishai A Feldman and David Harel. 1982. A probabilistic dynamic logic. In *Proceedings of the fourteenth annual ACM Symposium on Theory of computing*. ACM, 181–195.

Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 489–501. https://doi.org/10.1145/2676726.2677001

Robert W. Floyd. 1967. Assigning meanings to programs. *Mathematical Aspects of Computer Science* 19 (1967), 19–33.

F. G. Foster. 1953. On the Stochastic Matrices Associated with Certain Queuing Processes. *The Annals of Mathematical Statistics* 24, 3 (1953), pp. 355–360.

Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521, 7553 (2015), 452–459.

Laure Gonnord, David Monniaux, and Gabriel Radanne. 2015. Synthesis of Ranking Functions Using Extremal Counterexamples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '15)*. ACM, New York, NY, USA, 608–618. https://doi.org/10.1145/2737924.2737976

Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2014. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Performance Evaluation* 73 (2014), 110 – 132. https://doi.org/10.1016/j.peva.2013.11.004 Special Issue on the 9th International Conference on Quantitative Evaluation of Systems.

Sergiu Hart and Micha Sharir. 1985. Concurrent Probabilistic Programs, Or: How to Schedule if You Must. *SIAM J. Comput.* 14, 4 (1985), 991–1012.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run–Times of Probabilistic Programs. In *Programming Languages and Systems: 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, Peter Thiemann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 364–389. https://doi.org/10.1007/978-3-662-49498-1_15

Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. 2010. Linear-Invariant Generation for Probabilistic Programs: - Automated Support for Proof-Based Methods. In *SAS*, Vol. LNCS 6337, Springer. 390–406.

Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

Dexter Kozen. 1983. A Probabilistic PDL. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83)*. ACM, New York, NY, USA, 291–297. https://doi.org/10.1145/800061.808758

Daniel Kroening, Natasha Sharygina, Aliaksei Tsitovich, and Christoph M. Wintersteiger. 2010. Termination Analysis with Compositional Transition Invariants. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, Tayssir Touili, Byron Cook, and Paul Jackson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 89–103. https://doi.org/10.1007/978-3-642-14295-6_9

Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV (LNCS 6806)*. 585–591.

Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. 2001. The size-change principle for program termination. In *POPL*. 81–92.

Annabelle McIver and Carroll Morgan. 2004. Developing and Reasoning About Probabilistic Programs in *pGCL*. In *PSSE*. 123–155.

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer.

Annabelle McIver and Carroll Morgan. 2016. A new rule for almost-certain termination of probabilistic and demonic programs. *CoRR* abs/1612.01091 (2016). http://arxiv.org/abs/1612.01091

David Monniaux. 2001. An Abstract Analysis of the Probabilistic Termination of Programs. In *Static Analysis, 8th International Symposium, SAS 2001, Paris, France, July 16-18, 2001, Proceedings (Lecture Notes in Computer Science)*, Patrick Cousot (Ed.), Vol. 2126. Springer, 111–126. https://doi.org/10.1007/3-540-47764-0_7

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA.

M. Neuhäußer, M. Stoelinga, and J.-P. Katoen. 2009. Delayed Nondeterminism in Continuous-Time Markov Decision Processes. In *Proceedings of FoSSaCS 2009*, Vol. 5504. 364–379.

Martin R Neuhäußer and Joost-Pieter Katoen. 2007. Bisimulation and logical preservation for continuous-time Markov decision processes. In *International Conference on Concurrency Theory (CONCUR 2007)*. Springer, 412–427.

Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning About Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*. ACM, New York, NY, USA, 672–681. https://doi.org/10.1145/2933575.2935317

Andreas Podelski and Andrey Rybalchenko. 2004a. A Complete Method for the Synthesis of Linear Ranking Functions. In *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January*

11-13, 2004, Proceedings (Lecture Notes in Computer Science), Bernhard Steffen and Giorgio Levi (Eds.), Vol. 2937. Springer, 239–251. https://doi.org/10.1007/978-3-540-24622-0_20

Andreas Podelski and Andrey Rybalchenko. 2004b. Transition Invariants. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS '04)*. IEEE Computer Society, Washington, DC, USA, 32–41. https://doi.org/10.1109/LICS.2004.50

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

Micha Sharir, Amir Pnueli, and Sergiu Hart. 1984. Verification of Probabilistic Programs. *SIAM J. Comput.* 13, 2 (1984), 292–314.

Kirack Sohn and Allen Van Gelder. 1991. Termination Detection in Logic Programs using Argument Sizes. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado, USA*, Daniel J. Rosenkrantz (Ed.). ACM Press, 216–226. https://doi.org/10.1145/113413.113433