7

Termination Analysis of Probabilistic Programs with Martingales

Krishnendu Chatterjee
IST Austria
Hongfei Fu
Shanghai Jiao Tong University
Petr Novotný
Masaryk University

Abstract: Probabilistic programs extend classical imperative programs with random-value generators. For classical non-probabilistic programs, termination is a key question in static analysis of programs, that given a program and an initial condition asks whether it terminates. In the presence of probabilistic behavior there are two fundamental extensions of the termination question, namely, (a) the *almost-sure* termination question that asks whether the termination probability is 1; and (b) the *bounded-time* termination question that asks whether the expected termination time is bounded. While there are many active research directions to address the above problems, one important research direction is the use of martingale theory for termination analysis. We will survey the main techniques related to martingale-based approach for termination analysis of probabilistic programs.

7.1 Introduction

Stochastic models and probabilistic programs. The analysis of stochastic models is a fundamental problem, and randomness plays a crucial role in several disciplines across computer science. Some prominent examples are (a) randomized algorithms (Motwani and Raghavan, 1995); (b) stochastic network protocols (Baier and Katoen, 2008; Kwiatkowska et al., 2011); (c) systems that interact with uncertainty in artificial intelligence (Kaelbling et al., 1996, 1998; Ghahramani, 2015). Programming language support for analysis of such models requires extending the classical non-probabilistic programming models, and the extension of classical imperative programs with *random value generators* that produce random values according to some desired probability distribution gives rise to the class of probabilistic programs (Gordon et al., 2014). The formal analysis of probabilistic systems and probabilistic programs is an active research topic across different disciplines,

[a] From *Foundations of Probabilistic Programming*, edited by Gilles Barthe, Joost-Pieter Katoen and Alexandra Silva published 2020 by Cambridge University Press.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

such as probability theory and statistics (Durrett, 1996; Howard, 1960; Kemeny et al., 1966; Rabin, 1963; Paz, 1971), formal methods (Baier and Katoen, 2008; Kwiatkowska et al., 2011), artificial intelligence (Kaelbling et al., 1996, 1998), and programming languages (Chakarov and Sankaranarayanan, 2013; Fioriti and Hermanns, 2015; Sankaranarayanan et al., 2013; Esparza et al., 2012; Chatterjee et al., 2016a).

Termination questions. One of the most basic, yet fundamental, question in analysis of reactive systems or programs is the *termination* problem. For non-probabilistic program the termination problem asks whether a given program always terminates. The termination problem represents the fundamental notion of liveness for programs, and corresponds to the classical halting problem of Turing machines. While for general programs the termination problem is undecidable, static analysis methods for program analysis aim to develop techniques that can answer the question for subclasses of programs. For non-probabilistic programs, the proof of termination coincides with the construction of *ranking functions* (Floyd, 1967), and many different approaches exist for such construction (Bradley et al., 2005a; Colón and Sipma, 2001; Podelski and Rybalchenko, 2004; Sohn and Gelder, 1991). For probabilistic programs, the presence of randomness requires that the termination questions are extended to handle stochastic aspects. The most natural and basic extensions of the termination problem are as follows: First, the *almost-sure* termination question asks whether the program terminates with probability 1. Second, the *bounded* termination question asks whether the expected termination time is bounded. While the bounded termination implies almost-sure termination, the converse is not true in general. Section 7.2.4 illustrates the concepts on several examples.

Non-determinism. Besides stochastic aspects, another fundamental modeling concept is the notion of *non-determinism*. A classic example of non-determinism in program analysis is *abstraction*: for efficient static analysis of large programs, it is infeasible to track all variables of the program. Abstraction ignores certain variables and replaces them with worst-case behavior modeled as non-determinism. Moreover, non-determinism can be used to replace large portions of a program by overapproximating their effect on variables.

Martingales for probabilistic programs. While there are various different approaches for analyzing probabilistic programs (see Section 7.6 for further discussion), the focus of this chapter is to consider martingale-based approaches. This approach considers martingales (a special type of stochastic processes) and how they can be used to develop algorithmic analysis techniques for analysis of probabilistic programs. The approach brings together various different disciplines, namely, prob-

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

ability theory, algorithmic aspects, and program analysis techniques. Below we present a glimpse of the main results, and then the organization of the chapter.

Glimpse of main results. We present a brief description of main results related to the martingale-based approach.

- *Finite probabilistic choices.* First, for probabilistic programs with non-determinism, but restricted to finite probabilistic choices, quantitative invariants were used to establish termination in McIver and Morgan (2004, 2005).
- *Infinite probabilistic choices without non-determinism.* The approach presented in McIver and Morgan (2004, 2005) was extended in Chakarov and Sankaranarayanan (2013) to *ranking* supermartingales to obtain a sound (but not complete, see Chakarov and Sankaranarayanan (2013, page 10) for a counterexample) approach for almost-sure termination for infinite-state probabilistic programs without non-determinism, but with infinite-domain random variables. The connection between termination of probabilistic programs without non-determinism and *Lyapunov ranking functions* was considered in Bournez and Garnier (2005). For probabilistic programs with countable state space and without non-determinism, Lyapunov ranking functions provide a sound and complete method to prove bounded termination (Bournez and Garnier, 2005; Foster, 1953).
- *Infinite probabilistic choices with non-determinism.* The interaction of non-determinism and infinite probabilistic choice is quite tricky as illustrated in Fioriti and Hermanns (2015). For bounded termination, the ranking supermartingale based approach is sound and complete (Chatterjee and Fu, 2017; Fu and Chatterjee, 2019). As mentioned above, a key goal is to obtain algorithmic methods for automated analysis. Automated approaches for synthesis of linear and polynomial ranking supermartingales have been studied in Chatterjee et al. (2016a,b). Moreover, recently parametric supermartingales, rather than ranking supermartingales, (McIver et al., 2018; Chatterjee and Fu, 2017; Huang et al., 2018) and lexicographic ranking supermartingales (Agrawal et al., 2018) have been considered for proving almost-sure termination of probabilistic programs. The martingale-based approach has also been studied to prove high-probability termination and non-termination of probabilistic programs (Chatterjee et al., 2017).
- *Undecidability characterization.* The problem of deciding termination (almost-sure termination and bounded termination) of probabilistic programs is undecidable, and its precise undecidability characterization has been studied in Kaminski and Katoen (2015).

Organization. The chapter is organized as follows: In Section 7.2 we present the preliminaries (syntax, semantics, and the formal definition of termination problems). In Section 7.3 we present the results related to the theoretical foundations of ranking

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

supermartingales and bounded termination. In Section 7.4 we discuss algorithmic approaches for synthesis of linear and polynomial ranking supermaritngales. In Section 7.5 we consider the martingale-based approach beyond bounded termination: we first consider almost-sure termination and discuss the parametric supermartingale and lexicographic ranking supermartingale based approach, then discuss the approach for high-probability termination. Finally, we discuss related works (Section 7.6), and conclude with future perspective (Section 7.7).

7.2 Preliminaries

7.2.1 Syntax of Probabilistic Programs

We consider a mathematically clean formulation of a simple imperative probabilistic programming language with real-valued numerical variables. An abstract grammar of our probabilistic language is presented in Figure 7.1. There, $\langle pvar \rangle$ stands for program variables, while $\langle expr \rangle$ and $\langle boolexpr \rangle$ represent arithmetic expressions and boolean predicates, respectively.

Expressions and Predicates. We assume that the expressions used in each program satisfy the following: (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; and (2) the function defined by each expression E is Borel-measurable (for definition of Borel-measurability, see, e.g. Billingsley (1995)). This holds in particular for expressions built using standard arithmetic operators $(+, -, *, /)$, provided that expressions evaluating to zero are not allowed as divisors. A *predicate* is a boolean combination of *atomic predicates* of the form $E \leq E'$, where E, E' are expressions. We denote by $\mathbf{x} \models \psi$ the fact that the predicate ψ is satisfied by substituting values from of $\mathbf{x}$ for the corresponding variables in ψ.

Probability and Non-Determinism. Apart from the classical programming constructs, our probabilistic programs also have constructs introducing *probabilistic* and *non-deterministic* behaviour. The former include probabilistic branching (e.g. '**if prob(**$\frac{1}{3}$**) then...**') and sampling of a variable value from a probability distribution (e.g. $x :=$ **sample(**Uniform$[-2, 1]$**)**). We allow both discrete and continuous distributions and we also permit sampling instructions to appear in place of variables within expressions. For the purpose of our analysis, we require that for each distribution d appearing in the program we know the following characteristics: the expected value $\mathbb{E}[d]$ and a set SP_d containing the *support* of d (the support of d is the smallest closed set of real numbers whose complement has probability zero under d). We also allow (demonic) non-deterministic branching represented by $\star$ in the conditional guard. The techniques presented in this chapter can be extended also to

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

$$
\begin{aligned}
\langle stmt \rangle &::= \langle assgn \rangle \mid \textbf{skip} \mid \langle stmt \rangle \,;\, \langle stmt \rangle \\
&\quad \mid \textbf{if } \langle ndboolexpr \rangle \textbf{ then } \langle stmt \rangle \textbf{ else } \langle stmt \rangle \textbf{ fi} \\
&\quad \mid \textbf{while } \langle boolexpr \rangle \textbf{ do } \langle stmt \rangle \textbf{ od} \\
\langle assgn \rangle &::= \langle pvar \rangle := \langle expr \rangle \mid \langle pvar \rangle := \textbf{sample}(\langle dist \rangle) \\
\langle ndboolexpr \rangle &::= \star \mid \textbf{prob}(p) \mid \langle boolexpr \rangle
\end{aligned}
$$

Figure 7.1 Abstract grammar of imperative probabilistic programs.

programs with non-deterministic assignments, but we omit this feature for the sake of simplicity.

Affine Probabilistic Programs. The mathematical techniques presented in this chapter are applicable to a rather general class of probabilistic programs. When considering *automation* of these methods, we restrict our attention to *affine* programs. A probabilistic program $\mathcal{P}$ is affine if each arithmetic expression occurring in $\mathcal{P}$ (i.e. in its loop guards, conditionals, and right-hand sides of assignments) is an *affine expression,* i.e. an expression of the form $b + \sum_{i=1}^{n} a_i x_i$, where $b, a_1, \ldots, a_n$ are real-valued constants.

7.2.2 Semantics of Probabilistic Programs

We now sketch our definition of semantics of PPs with non-determinism. We use the standard operational semantics presented in more detail in (Agrawal et al., 2018).

Basics of Probability Theory. We assume some familiarity with basic concepts 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 $\mathbb{P}(\Omega \setminus A) = 1 - \mathbb{P}(A)$, and (3) for all pairwise disjoint countable set sequences $A_1, A_2, \cdots \in \mathcal{F}$ (i.e., $A_i \cap A_j = \emptyset$ for all $i \neq j$) we have $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}(\bigcup_{i=1}^{\infty} A_i)$.

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 $\{\omega \in \Omega \mid R(\omega) \leq a\}$ belongs to $\mathcal{F}$. We denote by $\mathbb{E}[R]$ the *expected value* of a random variable R (see (Billingsley, 1995, Chapter 5) for a formal definition). A *random vector* in $(\Omega, \mathcal{F}, \mathbb{P})$ is a vector whose every component is a random variable in this probability space. We denote by $\mathbf{X}[j]$ the j-component of a vector $\mathbf{X}$. A *stochastic process* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence of random vectors in this space. A *filtration* in the probability space is an infinite non-decreasing

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

sequence of sigma-algebras $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \cdots \mathcal{F}$ characterizing an increase of available information over time (see (Williams, 1991, Chapter 10)). A process $\{X_n\}_{n\in\mathbb{N}_0}$ is *adapted* to the filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$, if X_n is $\mathcal{F}_n$-measurable for each $n \in \mathbb{N}_0$. We will also use random variables of the form $R\colon \Omega \to S$ for some finite set S, which easily translates to the variables above.

Configurations and Runs. For a program $\mathcal{P}$ we denote by $V_\mathcal{P}$ the set of programs variables used in $\mathcal{P}$ (we routinely drop the subscript when $\mathcal{P}$ is known from the context). A *configuration* of a PP $\mathcal{P}$ is a tuple $(\ell, \mathbf{x})$, where ℓ is a program location (a line of the source code carrying a command) and $\mathbf{x}$ is *valuation*, i.e. a $|V_\mathcal{P}|$-dimensional vector s.t. $\mathbf{x}[t]$ is the current value of variable $t \in V_\mathcal{P}$. A *run* is a finite or infinite sequence of configurations corresponding to a possible execution of the program. A finite run which does not end in the program's terminal location is also called an *execution fragment.*

Schedulers. Non-determinism in a program is resolved via a *scheduler*. Formally, a scheduler is a function σ assigning to every execution fragment that ends in a location containing a command **if** $\star$ **then... else...** a probability distribution over the if- and else branches. 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). In the rest of this work, we refer to measurable schedulers simply as "schedulers."

From a Program to a Stochastic process. A program $\mathcal{P}$ together with a scheduler σ and initial variable valuation $\mathbf{x}_0$ 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. $(\ell_0, \mathbf{x}_0)$, where ℓ_0 corresponds to the first command of $\mathcal{P}$ and ℓ_0 is the initial valuation of variables (from now on, we assume that each program is accompanied by some initial variable valuation denoted $\mathbf{x}_{init}$). Now assume that i steps have elapsed and the program has not yet terminated, and let $\pi_i = (\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\cdots(\ell_i, \mathbf{x}_i)$ be the execution fragment produced so far. Then the next configuration $(\ell_{i+1}, \mathbf{x}_{i+1})$ is chosen as follows:

- If ℓ_i corresponds to a deterministic assignment, the assignment is performed and the program location advances to the next command, which yields the new configuration.
- If ℓ_i corresponds to a probabilistic assignment, the value to assign is first sampled from a given distribution, after which the assignment of the sampled value is performed as above.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

- If ℓ_i corresponds to a command **if** $\star$ **then...**, then a branch to execute is sampled according to scheduler σ, i.e. from the distribution $\sigma(\pi_i)$. The valuation remains unchanged, but ℓ_{i+1} advances to the first command of the sampled branch.
- If ℓ_i corresponds to a command **if prob**(p) **then...**, then we select the if branch with probability p and the else branch with probability $1 - p$. The selected branch is then executed as above.
- Otherwise, ℓ_i contains a standard deterministic conditional (branching or loop guard). We evaluate the truth value of the conditional under the current valuation $\mathbf{x}_i$ to select the correct branch, which is then executed as above.

The above intuition can be formalized by showing that each probabilistic program $\mathcal{P}$ together with a scheduler σ and initial valuation $\mathbf{x}_0$ uniquely determine a certain probability space $(\Omega_{Run}, \mathcal{R}, \mathbb{P}^{\sigma}_{\mathbf{x}_0})$ in which Ω_{Run} is a set of all runs in $\mathcal{P}$, and a stochastic process $C^{\sigma} = \{\mathbf{C}^{\sigma}_i\}_{i=0}^{\infty}$ in this space such that for each run $\varrho \in \Omega_{Run}$ we have that $\mathbf{C}^{\sigma}_i(\varrho)$ is the i-th configuration on ϱ. The formal construction of $\mathcal{R}$ and $\mathbb{P}^{\sigma}_{\mathbf{x}_0}$ proceeds via the standard *cylinder construction* (Ash and Doléans-Dade, 2000, Theorem 2.7.2). We denote by $\mathbb{E}^{\sigma}_{\mathbf{x}_0}$ the expectation operator in the probability space $(\Omega_{Run}, \mathcal{R}, \mathbb{P}^{\sigma}_{\mathbf{x}_0})$.

7.2.3 Termination

Each program $\mathcal{P}$ has a special location ℓ_{out} corresponding to the value of the program counter after finishing the execution of $\mathcal{P}$. We say that a run *terminates* if it reaches a configuration whose first component is ℓ_{out}; such configurations are called *terminal*.

Analysing program termination is one of the fundamental questions already in non-probabilistic program analysis. The question whether (every execution of) a program terminates is really just a re-statement of the classical Halting problem for Turing machines, which is, per one of the first fundamental results in computer science, undecidable (Turing, 1937). While we cannot *decide* whether a given program terminates, we can still aim to *prove* program termination via automated means, i.e. construct an algorithm which proves termination of as many terminating programs as possible, and reports a failure when it is unable to find such a proof (note that failure to find a termination proof does not, per se, prove the program's non-termination).

A classical technique for proving termination of non-probabilistic programs is the synthesis of an appropriate *ranking function* (Floyd, 1967). A ranking function maps program configurations to rational numbers, satisfying the following two properties: (1) each step of the program's execution strictly decreases the value of the ranking function by a value bounded away from zero, say at least by one; and (2) non-terminal configurations are mapped to positive numbers. Due to this strict decrease, the value of the function cannot stay positive *ad infinitum*; hence, the

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

existence of a ranking function shows that the program terminates. Conversely, if we restrict to non-probabilistic programs with *bounded non-determinism* (where the number of non-deterministic choices in every step is bounded by some constant, such as in our syntax), then each such terminating program possesses a ranking function which maps a configuration $(\ell, \mathbf{x})$ to the maximal number of steps the program needs to reach a terminal configuration from $(\ell, \mathbf{x})$. Since termination is undecidable, we cannot have a sound and complete algorithm for synthesis of such ranking functions. We can however employ techniques that are sound and *conditionally complete* in the sense that they search for ranking functions of a restricted form (such as linear ranking functions) and are guaranteed to find such a restricted ranking function whenever it exists (Bradley et al., 2005a; Colón and Sipma, 2001; Podelski and Rybalchenko, 2004).

7.2.4 Termination Questions for Probabilistic Programs

Termination and Termination Time. Recall that ℓ_{out} is a location to a terminated program execution. We define a random variable *Term* such that for each run ϱ the value $Term(\varrho)$ represents the first point in time when the current location is ℓ_{out}. 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:

- *Almost-sure termination:* A probabilistic program $\mathcal{P}$ is almost-surely (a.s.) terminating if under each scheduler σ it holds that $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\varrho \in \Omega_{Run} \mid \varrho \text{ terminates}\}) = 1$, or equivalently, if for each σ it holds $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(Term < \infty) = 1$.
- *Finite and bounded termination:* A probabilistic program $\mathcal{P}$ is said to be *finitely* (aka *positive almost-surely* (Fioriti and Hermanns, 2015)) terminating if under each σ it holds that $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] < \infty$. Furthermore, the program $\mathcal{P}$ is *boundedly* terminating if we have $\sup_{\sigma} \mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] < \infty$.
- *Probabilistic termination:* In this generalization of the a.s. termination problem, we aim to compute a non-trivial lower bound $p \in [0, 1]$ on termination probability, i.e. p s.t. for each σ we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\varrho \in \Omega_{Run} \mid \varrho \text{ terminates}\}) \geq p$. In particular, here we also aim to analyse programs that are not necessarily a.s. terminating.

Remark 7.1 We present some remarks about the above definitions.

- First, finite termination implies almost-sure termination as $\mathbb{E}^{\sigma}_{\mathbf{x}_{init}}[Term] < \infty$ implies $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(Term) = 1$; however, the converse does not hold (see Example 7.2 below).
- Second, there is subtle but important conceptual difference between finite and bounded termination. While the first asks for the expected termination time to be

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

finite for all schedulers, the expected termination time can still grow unbounded with different schedulers. In contrast, the bounded termination asks for the expected termination time to be bounded for all schedulers (but can depend on initial configuration). For probabilistic programs without non-determinism they coincide, since there is no quantification over schedulers. In general bounded termination implies finite termination; however, the converse does not hold (see Example 7.3 below).

It follows that bounded termination provides the strongest termination guarantee among the above questions, and we will focus on bounded termination.

Example 7.2 We present an example program that is almost-sure terminating, but not finite terminating. Consider the probabilistic program depicted in Figure 7.2. The loop models the classical symmetric random walk that hits zero almost-surely, but in infinite expected termination time (see Williams, 1991, Chapter 10). Hence, the loop is a.s. terminating but not finitely terminating.

Example 7.3 We present, in Figure 7.3, an example program that is finitely terminating, but not boundedly terminating. A scheduler for the program can be characterized by how many times the scheduler chooses the program counter 6 from the non-deterministic branch at the program counter 5 (finitely or infinitely), as once the scheduler chooses the program counter 10, the program then jumps out of the while loop at the program counter 4 and terminates after the execution of the while loop at the program counter 12. Under each such scheduler, the expected termination time is finite, so we have that the program is finitely terminating. However, since we have 3^n at the right-hand-side of the program counter 10 and the probability to jump out of the while loop at the non-deterministic branch 4 is 0.5 by the Bernoulli distribution, the expected termination time under a scheduler can be arbitrarily large when the number of times to choose the program counter 6 at the program counter 5 increases. Hence, there is no upper bound on the expected termination time for all schedulers, i.e., the probabilistic program is not boundedly terminating. See Fioriti and Hermanns (2015, Page 2) for details.

We now argue that termination analysis for probabilistic programs is more complex than for non-probabilistic programs.

First, note that the classical ranking functions do not suffice to prove even almost-sure termination. Since ranking functions are designed for non-probabilistic programs, applying them to probabilistic programs would necessitate replacing probabilistic choices and assignments with non-determinism. But Figure 7.2 shows a program which terminates almost-surely in the probabilistic setting, but does not necessarily terminate when the choice on line 3 is replaced by non-determinism (the

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

non-deterministic choice might e.g. alternate between the if- and the else-branch, preventing x from decreasing to 0).

Second, there are deeper theoretical reasons for the hardness of probabilistic termination. The termination of classical programs, i.e. the halting problem, is undecidable but recursively enumerable. As shown by (Kaminski and Katoen, 2015), the problems of deciding almost sure and positive termination in probabilistic programs are complete for the 2nd level of the arithmetic hierarchy.

Hence, the classical analysis is not applicable and new approaches to probabilistic termination are needed.

7.3 Theoretical Foundations for Bounded Termination

In this section, we establish theoretical foundations for proving bounded termination of probabilistic programs. First, we consider probabilistic programs without non-determinism and demonstrate mathematical approaches for proving bounded termination over such programs. Second, we extend the approach to probabilistic programs with non-determinism. Third, we show that our approach is sound and complete for proving bounded termination of non-deterministic probabilistic programs. Finally, we describe algorithms for proving bounded termination.

7.3.1 Probabilistic Programs without Non-determinism

For probabilistic programs without non-determinism, Chakarov and Sankaranarayanan (2013) first proposed a sound approach for proving bounded termination. (Recall that in the absence of non-determinism, finite and bounded termination coincide.) The approach can be described as follows.

- First, a general result on bounded termination of a special class of stochastic processes called *ranking supermartingales* (RSMs) is established.
- Second, program executions are translated into stochastic processes through a notion of *RSM-maps*.
- Third, the existence of RSM-maps that ensure bounded termination of probabilistic programs without non-determinism is established. The central idea of the proof is a construction of RSMs from RSM-maps.

We begin with the notion of *ranking supermartingales* which is the key to the approach. We take the original definition in Chakarov and Sankaranarayanan (2013).

Definition 7.4 (Ranking Supermartingales) A discrete-time stochastic process $\Gamma = \{X_n\}_{n\in\mathbb{N}_0}$ adapted to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$ is a *ranking supermartingale* (RSM) if there exist real numbers $\epsilon > 0$ and $K \leq 0$ such that for all $n \in \mathbb{N}_0$, the following conditions hold:

- (integrability) $\mathbb{E}[|X_n|] < \infty$;
- (lower-bound) it holds a.s. that $X_n \geq K$;

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1:  x := 100;
2:  while x ≥ 0 do
3:      if prob(0.5) then
4:          x := x + 1
        else
5:          x := x − 1
        fi;
    od
6:
```

Figure 7.2 An a.s. (but not finitely) terminating example

```
1: n := 0;  2: i := 0;  3: c := 0;
4: while c = 0 do
5:     if ⋆ then
6:         c := sample(Bernoulli(0.5));
7:         if c = 0 then
8:             n := n + 1
           else
9:             i := n
           fi
       else
10:        i := 3^n;
11:        c := 1
       fi
   od;
12: while i > 0 do
13:    i := i − 1
   od
14:
```

Figure 7.3 A finitely (but not boundedly) terminating example

- (ranking) it holds a.s. that $\mathbb{E}[X_{n+1}|\mathcal{F}_n] \leq X_n - \epsilon \cdot \mathbf{1}_{X_n \geq 0}$, where the random variable $\mathbb{E}[X_{n+1}|\mathcal{F}_n]$ is the conditional expectation of X_{n+1} given the sigma-algebra $\mathcal{F}_n$ (see Williams, 1991, Chapter 9 for details), and the random variable $\mathbf{1}_{X_n \geq 0}$ takes value 1 if the event $X_n \geq 0$ holds and 0 otherwise.

Informally, an RSM is a stochastic process whose values have a lower bound and decrease in expectation when the step increases.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

The random variable Z_Γ. Given an RSM $\Gamma = \{X_n\}_{n\in\mathbb{N}_0}$ adapted to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$, we define the random variable Z_Γ by $Z_\Gamma(\omega) := \min\{n \mid X_n(\omega) < 0\}$ where $\min\emptyset := \infty$. By definition, the random variable Z_Γ measures the amount of steps before the value of the stochastic process Γ drops below zero for the first time.

The following theorem from illustrates the relationship between an RSM Γ and its termination time Z_Γ. There are several versions for the theorem. The original version is Chakarov and Sankaranarayanan (2013) which only asserts almost-sure termination. (Recall that bounded termination implies almost-sure termination, but not vice versa.) Then in Fioriti and Hermanns (2015, Lemma 5.5), the theorem was extended to bounded termination with an explicit upper bound on expected termination time. The version in Fioriti and Hermanns (2015, Lemma 5.5) restricts K to be zero. Here we follow the version in Chatterjee et al. (2018a) that relaxes K to be a non-positive number while deriving an upper bound on the expected termination time.

Proposition 7.5 *Let $\Gamma = \{X_n\}_{n\in\mathbb{N}_0}$ be an RSM adapted to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$ with ϵ, K given as in Definition 7.4. Then $\mathbb{P}(Z_\Gamma < \infty) = 1$ and $\mathbb{E}[Z_\Gamma] \leq \frac{\mathbb{E}(X_0)-K}{\epsilon}$.*

Proof Sketch Using the ranking condition in Definition 7.4, we first prove by induction on $n \geq 0$ that $\mathbb{E}[X_n] \leq \mathbb{E}[X_0] - \epsilon \cdot \sum_{k=0}^{n-1} \mathbb{P}(X_k \geq 0)$. Moreover, we have from the lower-bound condition in Definition 7.4 that $\mathbb{E}[X_n] \geq K$, for all n. Then we obtain that for all n, it holds that

$$\sum_{k=0}^{n} \mathbb{P}(X_k \geq 0) \leq \frac{\mathbb{E}[X_0]-\mathbb{E}[X_{n+1}]}{\epsilon} \leq \frac{\mathbb{E}[X_0]-K}{\epsilon}.$$

Hence, the series $\sum_{k=0}^{\infty} \mathbb{P}(X_k \geq 0)$ converges and is no greater than $\frac{\mathbb{E}[X_0]-K}{\epsilon}$. It follows from $Z_\Gamma(\omega) > k \Rightarrow X_k(\omega) \geq 0$ (for all k, ω) that

- $\mathbb{P}(Z_\Gamma = \infty) = \lim_{k\to\infty} \mathbb{P}(Z_\Gamma > k) = 0$, and
- $\mathbb{E}[Z_\Gamma] = \sum_{k=0}^{\infty} \mathbb{P}(k < Z_\Gamma < \infty) \leq \sum_{k=0}^{\infty} \mathbb{P}(X_k \geq 0) \leq \frac{\mathbb{E}[X_0]-K}{\epsilon}$.

Then the desired result follows. See Fioriti and Hermanns (2015, Lemma 5.5) and Chatterjee et al. (2018a, Proposition 3.2) for details. □

Theorem 7.5 established the first step of the approach. In the next step, we need to relate RSMs with probabilistic programs. To accomplish this, the notion of RSM-maps plays a key role. We first introduce the notion of *pre-expectation*, then that of *RSM-maps*.

Below we fix a non-deterministic probabilistic program P with the set L of locations (values of the program counter), the set V of program variables and the set $\mathcal{D}$ of probability distributions appearing in P. Then the set of variable valuations is $\mathbb{R}^{|V|}$ and the set of configurations is $L \times \mathbb{R}^{|V|}$. Moreover, we say that a *sampling* valuation is a real vector in $\mathbb{R}^{|\mathcal{D}|}$ that represents a vector of sampled values from

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

all probability distributions. Then for each assignment statement in P at a location ℓ, regardless of whether the assignment statement is deterministic or a sampling, we have a function F_ℓ which maps each current variable valuation $\mathbf{x}$ and current sampling valuation $\mathbf{r}$ to the next variable valuation $F_\ell(\mathbf{x}, \mathbf{r})$ after the execution of the assignment statement.

The following definition introduces the notion of *pre-expectation* (Chatterjee et al., 2018a; Chakarov and Sankaranarayanan, 2013; McIver and Morgan, 2005).

Definition 7.6 (Pre-expectation) Let $\eta : L \times \mathbb{R}^{|V|} \to \mathbb{R}$ be a function which maps every configuration to a real number. We define the *pre-expectation* of η as the function $\mathrm{pre}_\eta : L \times \mathbb{R}^{|V|} \to \mathbb{R}$ by:

- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \sum_{\ell' \in L} p_{\ell,\ell'} \cdot \eta(\ell', \mathbf{x})$ if ℓ corresponds to a probabilistic branch and $p_{\ell,\ell'}$ is the probability that the next location is ℓ';
- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell', \mathbf{x})$ if ℓ corresponds to either an if-branch or a while-loop and ℓ' is the next location determined by the current variable valuation $\mathbf{x}$ and the boolean predicate associated with ℓ;
- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell', \mathbb{E}_{\mathbf{r}}(F_\ell(\mathbf{x}, \mathbf{r})))$ if ℓ corresponds to an assignment statement, where ℓ' is the location after the assignment statement and the expectation $\mathbb{E}_{\mathbf{r}}(-)$ is considered when $\mathbf{x}$ is fixed and $\mathbf{r}$ observes the corresponding probability distributions in $\mathcal{D}$.

Intuitively, $\mathrm{pre}_\eta(\ell, \mathbf{x})$ is the expected value of η after the execution of the statement at ℓ with the current configuration $(\ell, \mathbf{x})$.

Remark 7.7 The pre-expectation here is taken from Chatterjee et al. (2018a), and is a *small-step* version that only considers the execution of one individual statement. A *big-step* version is given in Chakarov and Sankaranarayanan (2013) and McIver and Morgan (2005) that consider the execution of a block of statements. The big-step version can be obtained by iterating the small-step version statement by statement in the block.

Invariants. To introduce the notion of ranking-supermartingale maps, we further need the notion of *invariants*. Formally, given an initial configuration $(\ell_0, \mathbf{x}_0)$, an invariant I is a function that assigns to each location ℓ a subset of variable valuations $I(\ell)$ such that in any program execution from the initial configuration and for all configurations $(\ell, \mathbf{x})$ visited in the execution, we have that $\mathbf{x} \in I(\ell)$. Intuitively, an invariant is an over-approximation of the reachable configurations from a specified initial configuration. A trivial invariant is the one that assigns to all locations the set $\mathbb{R}^{|V|}$ of all variable valuations. Usually, we can obtain more precise invariants that tightly approximate the reachable configurations through well-established techniques such as abstract interpretation (Cousot and Cousot, 1977).

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

Now we introduce the notion of ranking-supermartingale maps.

Definition 7.8 (Ranking-supermartingale Maps) A *ranking-supermartingale map* (RSM-map) wrt an invariant I is a function $\eta : L \times \mathbb{R}^{|V|} \to \mathbb{R}$ such that there exist real numbers $\epsilon > 0$ and $K, K' < 0$ such that for all configurations $(\ell, \mathbf{x})$, the following conditions hold:

(C1) if $\ell \neq \ell_{out}$ and $\mathbf{x} \in I(\ell)$, then $\eta(\ell, \mathbf{x}) \geq 0$;
(C2) if $\ell = \ell_{out}$ and $\mathbf{x} \in I(\ell)$, then $K \leq \eta(\ell, \mathbf{x}) \leq K'$;
(C3) if $\ell \neq \ell_{out}$ and $\mathbf{x} \in I(\ell)$, then $\text{pre}_\eta(\ell, \mathbf{x}) \leq \eta(\ell, \mathbf{x}) - \epsilon$.

Intuitively, the condition (C1) specifies that when the program does not terminate then the value of the RSM-map should be non-negative. The condition (C2) specifies that when the program terminates, then the value should be negative and bounded from below. Note that (C1) and (C2) together guarantees that the program terminates iff the value of the RSM-map is negative. Finally, the condition (C3) specifies that the pre-expectation at non-terminal locations should decrease at least by some fixed positive amount, which is related to the ranking condition in the RSM definition (cf. Definition 7.4).

The key role played by RSM-maps is that if we have an RSM-map, then we can assert the bounded termination of the program and an explicit upper bound. In other words, RSM-maps are *sound* for proving bounded termination of probabilistic programs. This is demonstrated by the following proposition from (Chatterjee et al., 2018a, Theorem 3.8).

Proposition 7.9 (Soundness) *If there exists an RSM-map η wrt some invariant I, then we have that* $\sup_\sigma \mathbb{E}^\sigma_{\mathbf{x}_{init}}[Term] \leq \frac{\eta(\ell_0, \mathbf{x}_{init}) - K}{\epsilon}$.

Proof Sketch Suppose that there is an RSM-map η. Let the stochastic process $\Gamma = \{X_n\}_{n \in \mathbb{N}_0}$ be given by: $X_n := \eta(\mathbf{C}_n)$. (Recall that $\mathbf{C}_n$ is the vector of random variables that represents the configuration at the n-th step in a program execution.) Then from (C2) and (C3), we have that Γ is an RSM with the same ϵ, K. Thus we obtain from Proposition 7.5 that $\mathbb{E}[Z_\Gamma] \leq \frac{\mathbb{E}[X_0] - K}{\epsilon}$. Furthermore, from (C1) and (C2), we have that $Term = Z_\Gamma$. It follows that $\mathbb{E}^\sigma_{\mathbf{x}_{init}}[Z_\Gamma] \leq \frac{\mathbb{E}[X_0] - K}{\epsilon}$ for all schedulers σ. See Chatterjee et al. (2018a, Theorem 3.8) for details. □

Below we illustrate the approach of RSM-maps for proving bounded termination of probabilistic programs through a simple example. The example is taken from Chakarov and Sankaranarayanan (2013, Example 2).

Example 7.10 (A Tortoise-Hare Race) Consider a scenario where a tortoise and a hare race against each other. The program representation for such a race is depicted in the left part of Figure 7.4. At the beginning, the hare starts at the position 0

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1: h := 0;
2: t := 30;
3: while h ≤ t do
4:   if prob(0.5) then
5:     r := sample(Uniform(0, 10));
6:     h := h + r
     else
7:     skip
     fi;
8:   t := t + 1
   od
9:
```

ℓ	Invariant I	RSM-map η
1	true	119
2	$h = 0$	118
3	$h \le t + 9$	$3 \cdot (t - h) + 27$
4	$h \le t$	$3 \cdot (t - h) + 26$
5	$h \le t$	$3 \cdot (t - h) + 18$
6	$h \le t \wedge 0 \le r \le 10$	$3 \cdot (t - h - r) + 32$
7	$h \le t$	$3 \cdot (t - h) + 32$
8	$h \le t + 10$	$3 \cdot (t - h) + 31$
9	$t \le h$	-1

Figure 7.4 **Left**: The Probabilistic Program for a Tortoise-Hare Race **Right**: An RSM-map for the Program

(location 1), while the tortoise starts at the position 30 (location 2). Then in each round (an iteration of the loop body from the location 4 to location 8), the hare either stops (location 7) or proceeds with a random distance that observes the uniform distribution over $[0, 10]$ (location 6), both with probability $\frac{1}{2}$, while the tortoise always proceeds with a unit distance (location 8). It is intuitively clear that the hare will eventually catch the tortoise and the program will enter the terminal location $\ell_{out} = 9$ in finite expected time.

The right part of Figure 7.4 illustrates an RSM-map η w.r.t an invariant I for the program, where "ℓ" stands for "location", the invariant I is specified through conditions on program variables for each location (e.g., $I(3)$ is the set of all variable valuations $\mathbf{x}$ where $\mathbf{x}[h] \le \mathbf{x}[t] + 9$), and the RSM-map η is also specified for each location (e.g., $\eta(3, \mathbf{x}) = 3 \cdot (\mathbf{x}[t] - \mathbf{x}[h]) + 27$).

The function η is an RSM-map with $\epsilon = 1, K = K' = -1$ since it satisfies (C1)–(C3). For example, the condition (C1) is satisfied at the location 3 since $\mathbf{x}[h] \le \mathbf{x}[t]+9$ implies that $3 \cdot (\mathbf{x}[t] - \mathbf{x}[h]) + 27 \ge 0$; the condition (C2) is straightforwardly satisfied at the location 9; finally, the condition (C3) at the location 5 is satisfied as we have $\mathbb{E}[\text{uniform}(0, 10)] = 5$ and $3 \cdot (t - h - \mathbb{E}(r)) + 32 \le 3 \cdot (t - h) + 18 - 1$. As a consequence, we obtain that $\mathbb{E}_{\mathbf{x}_{init}}[Term] \le \frac{\eta(1, \mathbf{x}_{init}) - K}{\epsilon} = 120$. □

7.3.2 Probabilistic Programs with Non-determinism

The approach of ranking supermartingales can be directly extended to non-determinism. However, before we illustrate the extension, an important issue to resolve is the

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

operational semantics with non-determinism. There is a diversity in the operational semantics for probabilistic programs with non-determinism. The semantics can be either directly based on random samplings (Fioriti and Hermanns, 2015) or Markov decision processes (MDPs). Below we first describe the result with random-sampling semantics (Fioriti and Hermanns, 2015), then the results with the MDP semantics (Chatterjee et al., 2018a).

Sampling-Based Semantics. In Fioriti and Hermanns (2015), a semantics directly based on samplings is proposed. Under this semantics, a sample point (in the sample space) is an infinite sequence of sampled values from corresponding probability distributions in the program. Then for each scheduler σ, there is a termination-time random variable $Term^{\sigma}$. The advantage of this semantics is that there is only one probability space (i.e., the set of all infinite sequences of sampled values). However, the cost is that there are many random variables $Term^{\sigma}$, and one needs to define a "supremum" random variable $Term^{*} := \sup_{\sigma} Term^{\sigma}$ where σ ranges over all schedulers. As a result, a relative completeness result under such semantics is established in Fioriti and Hermanns (2015, Theorem 5.8) which states that if $\mathbb{E}(Term^{*}) < \infty$ then there exists a ranking supermartingale. As the semantics takes the supremum over all termination-time random variables, it is infeasible to explore the internal effect of an individual scheduler. As a consequence, it is difficult to develop algorithmic approaches based on this semantics.

MDP-Based Semantics. In Chatterjee et al. (2018a), the MDP-semantics is adopted. MDPs are a standard operational model for probabilistic systems with non-determinism. Under the MDP-semantics, a state is a configuration of the program, while the probabilistic transitions between configurations are determined by the imperative semantics of each individual statement. Compared with the sampling-based semantics, there is only one termination-time random variable *Term* and each scheduler determines a probabilistic space. Since the behaviour of an individual scheduler can be manipulated under this semantics, algorithmic approaches can be developed (which will be further illustrated in Section 7.4).

We follow the MDP-based semantics and demonstrate the extension of ranking supermartingales to non-determinism. To introduce the notion of RSM-maps in the context of non-determinism, we first extend the notion of pre-expectation.

Below we fix a non-deterministic probabilistic program P with the set L of locations, the set V of program variables and the set $\mathcal{D}$ of probability distributions appearing in P.

Definition 7.11 (Pre-expectation) Let $\eta : L \times \mathbb{R}^{|V|} \to \mathbb{R}$ be a function which maps

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

every configuration to a real number. We define the *pre-expectation* of η as the function $\mathrm{pre}_\eta : L \times \mathbb{R}^{|V|} \to \mathbb{R}$ by:

- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \sum_{\ell' \in L} p_{\ell,\ell'} \cdot \eta(\ell', \mathbf{x})$ if ℓ corresponds to a probabilistic branch and $p_{\ell,\ell'}$ is the probability that the next location is ℓ';
- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell', \mathbf{x})$ if ℓ corresponds to either an if-branch or a while-loop and ℓ' is the next location given the current variable valuation $\mathbf{x}$;
- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell', \mathbb{E}_{\mathbf{r}}[F_\ell(\mathbf{x}, \mathbf{r})])$ if the location ℓ corresponds to an assignment statement, where ℓ' is the location after the assignment statement and the expectation $\mathbb{E}_{\mathbf{r}}(-)$ is considered when $\mathbf{x}$ is fixed and $\mathbf{r}$ observes the corresponding probability distributions in $\mathcal{D}$;
- $\mathrm{pre}_\eta(\ell, \mathbf{x}) := \max\{\eta(\ell_{\mathrm{th}}, \mathbf{x}), \eta(\ell_{\mathrm{el}}, \mathbf{x})\}$ if ℓ corresponds to a non-deterministic branch where ℓ_{th} and ℓ_{el} are the locations for respectively the then- and else-branch.

Compared with Definition 7.6, the current definition is extended with non-determinism. In the last item of Definition 7.11, the pre-expectation at a non-deterministic branch are defined as the maximum over its then- and else-branches. The reason to have maximum is that the non-deterministic branch in our programming language can be resolved arbitrarily by any scheduler, so we need to consider the worst case at non-deterministic branches regardless of the choice of the scheduler.

Soundness Result. Once we extend pre-expectation with non-determinism, we can keep the definition for RSM-maps the same as in Definition 7.8. Then with similar proofs, the statement of Proposition 7.9 still holds with non-determinism. Thus, RSM-maps are sound for proving bounded termination of probabilistic programs with non-determinism.

Proposition 7.12 (Soundness) *RSM-maps are sound for proving bounded termination of probabilistic programs with non-determinism.*

Completeness Result. In Fu and Chatterjee (2019, Theorem 2), a completeness result is established for RSM-maps and probabilistic programs with integer-valued variables. The result states that if the expected termination time of the probabilistic program is bounded for all schedulers, then there exists an RSM-map. The formal statement is as follows.

Proposition 7.13 (Completeness) *If all program variables in a probabilistic program P are integer-valued and $\sup_\sigma \mathbb{E}^\sigma_{\mathbf{x}_{init}}[Term] < \infty$, then there exists an RSM-map w.r.t some invariant I for P.*

From Proposition 7.12 and Proposition 7.13, we obtain that the approach of RSMs is sound and complete (through RSM-maps).

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

Theorem 7.14 *RSM-maps are sound and complete for proving bounded termination of probabilistic programs.*

Remark 7.15 Note that the termination problems for probabilistic programs generalize the termination problems for non-probabilistic programs (i.e., the halting problem of Turing machines) and is undecidable (for detailed complexity characterization see Kaminski and Katoen (2015)). The above soundness and completeness result does not imply that program termination is decidable, as it only ensures the existence of an RSM-map in general form which is not always computable. Thus the completeness result is orthogonal to the decidability results, however, special classes of RSM-maps can be obtained algorithmically which we consider in the following section.

7.4 Algorithms for Proving Bounded Termination

In the previous sections, we have illustrated that the existence of an RSM-map leads to bounded termination of probabilistic programs. Thus, in order to develop an algorithmic approach to prove bounded termination of probabilistic programs, it suffices to synthesize an RSM-map. Furthermore, since it is infeasible to synthesize an RSM-map in general form, in an algorithmic approach one needs to restrict the form of an RSM-map so as to make the approach feasible. In this section, we illustrate algorithmic approaches that can synthesize linear and polynomial RSM-maps given an input invariant (also in special form). Since the class of linear/polynomial RSM-maps is quite general, the corresponding algorithmic approaches can be applied to typical probabilistic programs such as gambler's ruin, random walks, robot navigation, etc. (see the experimental results in Chakarov and Sankaranarayanan (2013), Chatterjee et al. (2018a) and Chatterjee et al. (2016b) for details).

We first describe the algorithmic approach for synthesizing linear RSM-maps over *affine* probabilistic programs where the right-hand-side of each assignment statement is affine in program variables. A *linear* RSM-map is an RSM-map η such that for each location ℓ, the function $\eta(\ell, -)$ is affine in the program variables of P. For example, the RSM-map at the right part of Figure 7.4 is linear.

To illustrate the algorithm, we need the well-known Farkas' Lemma that characterizes the inclusion of a polyhedron in a halfspace.

Theorem 7.16 (Farkas' Lemma (Farkas, 1894; Schrijver, 2003)) *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ *and* $d \in \mathbb{R}$. *Assume that* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$. *Then we have that*

$$\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\} \subseteq \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{c}^{\mathrm{T}}\mathbf{x} \leq d\}$$

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

iff there exists $\mathbf{y} \in \mathbb{R}^m$ *such that* $\mathbf{y} \geq 0$, $\mathbf{A}^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ *and* $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq d$, *where* $\mathbf{y} \geq 0$ *means that every coordinate of* $\mathbf{y}$ *is non-negative.*

The Farkas' Linear Assertions Φ. Farkas' Lemma transforms the inclusion testing of systems of linear inequalities into an emptiness problem. Given a polyhedron $H = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ as in the statement of Farkas' Lemma (Theorem 7.16), we define the predicate $\Phi[H, \mathbf{c}, d](\xi)$ (which is called a Farkas' linear assertion) for Farkas' Lemma by

$$\Phi[H, \mathbf{c}, d](\xi) := (\xi \geq \mathbf{0}) \wedge \left(\mathbf{A}^{\mathrm{T}}\xi = \mathbf{c}\right) \wedge \left(\mathbf{b}^{\mathrm{T}}\xi \leq d\right)$$

where ξ is a variable representing a column vector of dimension m. Then by Farkas' Lemma, we have that $H \subseteq \{\mathbf{x} \mid \mathbf{c}^{\mathrm{T}}\mathbf{x} \leq d\}$ iff there exists a column vector $\mathbf{y}$ such that $\Phi[H, \mathbf{c}, d](\mathbf{y})$ holds.

Linear Invariants. We also need the notion of linear invariants. Informally, A *linear invariant* is an invariant I such that for all locations ℓ we have that $I(\ell)$ is a finite union of polyhedra.

Now we illustrate the algorithm for synthesizing linear RSM-maps w.r.t a given linear invariant. The description of the algorithm is as follows.

(i) First, the algorithm establishes a linear template for an RSM map. The linear template specifies that at each location, the function is affine in program variables with unknown coefficients. Besides, the algorithm also sets up three unknown parameters ϵ, K, K' which correspond to the counterparts in the definition of RSM-maps (cf Definition 7.8).

(ii) Second, the algorithm transforms the conditions (C1)–(C3) equivalently into Farkas' linear assertions through Farkas' Lemma.

(iii) Third, since the Farkas' linear assertions refer to the emptiness problem over polyhedra, we can use linear programming to solve those assertions. If a linear programming solver eventually finds the concrete values for the unknown coefficients in the template, then the algorithm finds a linear RSM-map that witnesses the bounded termination of the input program. Otherwise, the algorithm outputs "fail", meaning that the algorithm does not know whether the input program is boundedly terminating or not.

Since linear programming can be solved in polynomial time, our algorithm also runs in polynomial time, as is illustrated by the following theorem.

Theorem 7.17 (Chatterjee et al., 2018a, Theorem 4.1) *The problem to synthesize a linear RSM-map over non-deterministic affine probabilistic programs where all loop guards are in disjunctive normal form can be solved in polynomial time.*

Below we illustrate the details on how the algorithm works on Example 7.10.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

Example 7.18 We illustrate our algorithm on Example 7.10, where the input invariant is the same as given by the right part of Figure 7.4.

(i) First, the algorithm establishes a template η for a linear RSM-map so that $\eta(i,-) = a_i \cdot h + b_i \cdot t + c_i \cdot r + d_i$ for $1 \le i \le 9$, where a_i, b_i, c_i, d_i are unknown coefficients. The algorithm also sets up the three unknown parameters ϵ, K, K'.

(ii) Second, the algorithm transforms the conditions (C1)–(C3) at all locations into Farkas' linear assertions. We present two examples for such transformation.

- The condition (C1) at location 6 says that $\eta(6,-)$ should be non-negative over the polyhedron $H' := \{\mathbf{x} \mid \mathbf{x}[h] \le \mathbf{x}[t] \wedge 0 \le \mathbf{x}[r] \le 10\}$. Then from Farkas' Lemma, we construct the Farkas' linear assertion $\Phi[H', (-a_6, -b_6, -c_6)^{\mathrm{T}}, d_6](\xi)$ where ξ is a column vector of fresh variables.
- The condition (C3) at location 4 says that $0.5 \cdot \eta(5,-) + 0.5 \cdot \eta(7,-) + \epsilon \le \eta(4,-)$ holds over the polyhedron $H'' := \{\mathbf{x} \mid \mathbf{x}[h] \le \mathbf{x}[t]\}$. Note that $0.5 \cdot \eta(5,-) + 0.5 \cdot \eta(7,-) + \epsilon - \eta(4,-) = (\mathbf{c}') \cdot (h,t,r)^{\mathrm{T}} - d'$ where $\mathbf{c}' = (0.5 \cdot (a_5 + a_7) - a_4, 0.5 \cdot (b_5 + b_7) - b_4, 0.5 \cdot (c_5 + c_7) - c_4)$ and $d' = -0.5 \cdot (d_5 + d_7) + d_4 - \epsilon$. So we construct the Farkas' linear assertion $\Phi[H'', \mathbf{c}', d'](\xi')$ with fresh variables in ξ'. Note that this assertion is linear in both the unknown coefficients (i.e., a_i, b_i, c_i, d_i's), the unknown parameters ϵ, K, K' and the variables in ξ'.

(iii) Third, the algorithm collects all the Farkas' linear assertions constructed from the second step in the conjunctive fashion. Then, together with the constraint $\epsilon \ge 1$ and $K, K \le -1$ (which is equivalent to $\epsilon > 0$ and $K, K < 0$ as we can always multiply them with a large enough factor), the algorithm calls a linear programming solver (e.g. Cplex, 2010, Lpsolve, 2016) to get the solution to the unknown coefficients in the template.

Remark 7.19 (Synthesis of Polynomial RSM-maps) In several situations, linear RSM-maps do not suffice to prove bounded termination of probabilistic programs. To extend the applicability of RSM-maps, Chatterjee et al. (2016b) proposed an efficient sound approach to synthesize polynomial RSM-maps. The approach is through Positivstellensatz's (Scheiderer, 2008), an extension of Farkas' Lemma to polynomial case, and linear/semidefinite programming. This sound approach gives polynomial-time algorithms. Moreover, it is shown in Chatterjee et al. (2016b) that the existence of polynomial RSM-maps is decidable through the first-order theory of reals.

Remark 7.20 (Angelic Non-determinism) In this chapter, all non-deterministic branches are *demonic* in the sense that they cannot be controlled and we need to consider the worst-case. In contrast to demonic non-deterministic branches, *angelic* non-deterministic branches are branches that can be controlled in order to fulfill a prescribed aim. Similar to the demonic case, theoretical and algorithmic

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

approaches for angelic branches have been considered. The differences for angelic non-determinism as compared to demonic non-determinism are follows: (i) Motzkin's Transposition Theorem is used instead of Farkas' Lemma in the algorithm, and (ii) the problem to decide the existence of a linear RSM-map over affine probabilistic programs with angelic non-determinism is NP-hard and in PSPACE (see Chatterjee et al. (2018a) for details).

Remark 7.21 (Concentration Bound) A key advantage of martingales is that with additional conditions sharp concentration results can be obtained. For example, in Chatterjee et al. (2018a), it is shown that the existence of a *difference-bounded* RSM can derive a *concentration bound* beyond which the probability of non-termination within a given number of steps decreases exponentially. Informally, an RSM is difference-bounded if its change of value is bounded from the current step to the next step. The key techniques for such concentration bounds are Azuma's or Hoeffding's inequality; for a detailed discussion see Chapter 8 of this book.

7.5 Beyond Bounded Termination

As shown above, ranking supermartingales provide a sound and complete method of proving bounded termination. In this section, we present several martingale techniques capable of proving *a.s.* termination of programs that do not necessarily terminate in bounded or even finite expected time. Moreover, already for programs that *do* terminate boundedly, some of the techniques we present here provide a computationally more efficient approach to termination proving. For succinctness, we will from now on omit displaying the terminal location when presenting program examples.

7.5.1 Zero Trend and Zeno Behaviour

Consider a program modelling a symmetric random walk (Figure 7.5, here in a discrete variant where the change in each step is either -1 or $+1$, with equal probability). It is well-known that such a program is a.s. terminating. At the same time it does not admit any ranking supermartingale. This is because ranking supermartingales require that the "distance" to termination strictly decreases (in expectation) in every step, while the expected one-step change of the symmetric random walk is zero. Another scenario in which the standard ranking supermartingales are not applicable is when there is a progress towards termination, but the magnitude of this progress decreases over the runtime of the program, as is the case in Figure 7.6.

McIver et al. (2018) give a martingale-based proof rule which can handle the above issues. Here we present a re-formulation of the rule within the scope of our

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1:  while x ≥ 1 do
2:      x := x + sample(Uniform{−1, 1})
    od
```

Figure 7.5 Symmetric random walk.

```
1:  while x ≥ 1 do
2:      p := 1/(x + 1)
3:      t := sample(Uniform[0, 1])
4:      if t ≤ p then
5:          x := 0
        else
6:          x := x + 1
        fi od
```

Figure 7.6 Escaping spline program from (McIver et al., 2018).

syntax and semantics of PPs. In the following, we say that a real function f is *antitone* (or, alternatively, *non-increasing*) if $f(x) \leq f(y) \Leftrightarrow y \leq x$.

Definition 7.22 A non-negative discrete-time stochastic process $\Gamma = \{X_n\}_{n\in\mathbb{N}_0}$ adapted to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$ is a *parametric ranking supermartingale* (PRSM) if there exist functions d (for "decrease") of type $d\colon \mathbb{R} \to \mathbb{R}_{\geq 0}$, and p (for "probability") of type $\mathbb{R} \to [0, 1]$, both of them antitone and strictly positive on positive reals, such that the following conditions hold:

(i) for each $n \in \mathbb{N}_0$, $\mathbb{E}[X_{n+1} \mid \mathcal{F}_n] \leq X_n$; and
(ii) for each $n \in \mathbb{N}_0$, $\mathbb{P}(X_{n+1} \leq X_n - d(X_n) \mid \mathcal{F}_n) \geq p(X_n)$.

In PRSMs, the constraint on expected change is relaxed so that we prohibit an expected increase of the value (i.e., Γ has to be a supermartingale). On the other hand, in each step, there is a positive probability of a strict decrease, and this probability as well as the magnitude of the decrease can only get larger as the value of the process approaches zero (this is to avoid a possible "Zeno behaviour", when the process would approach zero but never reach it).

Theorem 7.23 *Let $\Gamma = \{X_n\}_{n\in\mathbb{N}_0}$ be a PRSM adapted to some filtration. Then $\mathbb{P}(Z_\Gamma < \infty) = 1$, i.e. with probability 1 the process reaches a zero value.*

Proof (sketch). Let $H \in \mathbb{N}$ be arbitrary, and and let T_H be a random variable returning the first point in time in which Γ jumps out of the interval $(0, H]$. Then has $\mathbb{P}(T_H < \infty) = 1$. This is because within the interval $(0, H]$ both the probability and magnitude of decrease of Γ are bounded away from zero (as p and d are

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

antitone on positive reals), so Γ cannot stay within this interval forever with positive probability. Hence, we can apply the optional stopping theorem for non-negative supermartingales Williams (1991, Section 10.10 d)), which says that the expected value $\mathbb{E}[X_{T_H}]$ of Γ at time T_H satisfies $\mathbb{E}[X_{T_H}] \leq \mathbb{E}[X_0]$. But at the same time $\mathbb{E}[X_{T_H}] \geq H \cdot \mathbb{P}(X_{T_H} \geq H)$, so $\mathbb{P}(X_{T_H} \geq H) \leq E[X_0]/H$. Hence, the probability that Γ "escapes" through the upper boundary of $(0, H]$ decreases as H increases. It follows that, denoting ℓ_H the probability that Γ escapes through the lower boundary, we have $\ell_H \to 1$ as $H \to \infty$. But each ℓ_H is a lower bound on $\mathbb{P}(Z_\Gamma < \infty)$, from which the result follows. □

One way to apply this theorem to a concrete program $\mathcal{P}$ equipped with an invariant I is to find positive antitone functions p_ℓ, d_ℓ (one per each location) along with a function η mapping $\mathcal{P}$'s configurations to non-negative real numbers, such that the following holds whenever $\mathbf{x} \in I(\ell)$: $\eta(\ell, \mathbf{x}) > 0$ if ℓ is not the terminal configuration; $\mathrm{pre}_\eta(\ell, \mathbf{x}) \leq \eta(\ell, \mathbf{x})$; and, denoting $P_{\eta,\mathbf{x},\ell}$ the function mapping $(\ell', \mathbf{y})$ to 0 if $\eta(\ell', \mathbf{y}) \leq \eta(\ell, \mathbf{x}) - d_\ell(\eta(\ell, \mathbf{x}))$, and to 1 otherwise, we have $\mathrm{pre}_{P_{\eta,\mathbf{x},\ell}}(\ell, \mathbf{x}) \leq 1 - p_\ell(\eta(\ell, \mathbf{x}))$. We call such a function η a *PRSM map*. Existence of such a PRSM map guarantees that the program terminates almost-surely. (Note that allowing separate d and p functions for each location is acceptable, since there are only finitely many locations and a minimum of finitely many positive antitone functions is again positive and antitone.) However, finding PRSM maps might be an intricate process. To illustrate this, consider the symmetric ransom walk in Figure 7.5. Looking at a definition of a PRSM, it would seem natural to choose x itself as the required function, since its expected change is non-positive and with probability $\frac{1}{2}$ the value of x decreases by 1 in every loop iteration. However, a mapping η assigning x to each of the two program locations is not a PRSM map, since at the beginning of each loop iteration, when transitioning from location 1 to location 2, there is not a positive probability of decrease of x. Indeed, a simple computation shows that there is no linear PRSM map for the program. Nevertheless, a PRSM map exists, as the following example shows.

Example 7.24 Take η such that $\eta(1, x) = \sqrt{x+1}$ and $\eta(2, x) = \frac{1}{2} \cdot \sqrt{x} + \frac{1}{2} \cdot \sqrt{x+2}$. Indeed, such η only takes positive values for $x \geq 0$ and furthermore, $\mathrm{pre}_\eta(2, \mathbf{x}) = \eta(2, \mathbf{x})$ (by definition) and $\mathrm{pre}_\eta(1, \mathbf{x}) = \frac{1}{2} \cdot \sqrt{x} + \frac{1}{2} \cdot \sqrt{x+2} \leq \sqrt{x+1}$ the last inequality following by a straightforward application of calculus. As for the decrease function, when making a step from 2 to 1, there is a $p_2 = \frac{1}{2}$ probability of the value decreasing by $d_2(\eta(2, x)) = \frac{1}{2}\sqrt{x+2} - \frac{1}{2}\sqrt{x}$; while a step from 1 to 2 entails decrease by $d_1(\eta(1, x)) = \sqrt{x+1} - \frac{1}{2}\sqrt{x+2} - \frac{1}{2}\sqrt{x}$ with probability $p_1 = 1$. A straightforward analysis reveals that both d_1 and d_2 are positive and antitone on positive reals.

An alternative "loop-based" approach to usage of PRSMs was proposed in (McIver

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

et al., 2018). Imagine that our aim is to prove almost-sure termination of a probabilistic loop 1 : **while** ψ **do** $\mathcal{P}$ **od**, and that we are provided with an invariant $I(1)$ for the head of the loop. Assume that for each configuration $\mathbf{x}$ such that $\mathbf{x} \in I(1)$ and $\mathbf{x} \models \psi$, the body $\mathcal{P}$ of the loop terminates when started with variables set according to $\mathbf{x}$. (Such a guarantee might be obtained by recursively analysing $\mathcal{P}$. If $\mathcal{P}$ is loopless, the guarantee holds trivially.) Let f be a non-negative function mapping variable valuations to real numbers. Since $\mathcal{P}$ is guaranteed to terminate a.s., we can define a stochastic process $\{X_i^f\}_{i\in\mathbb{N}_0}$ such that for a run ϱ, $X_i^f(\varrho)$ returns the value $f(\tilde{\mathbf{x}}_i)$, where $\tilde{\mathbf{x}}_i$ is the valuation of variables immediately after the i-th iteration of the loop along ϱ (if ϱ traverses the loop less than i times, we put $X_i^f(\varrho) = 0$). If the process $\{X_i^f\}_{i\in\mathbb{N}_0}$ is a PRSM, then with the help of Theorem 7.23 it can be easily shown that the loop indeed terminates almost surely.

Example 7.25 Returning to the symmetric random walk (Figure 7.5), let $f(x) = x$. In each iteration of the loop, the value of x has zero expected change, and with probability $p = \frac{1}{2}$ it decreases by $d = 1$. Hence, $\{X_i^f\}_{i\in\mathbb{N}_0}$ is a PRSM and the walk terminates a.s.

Example 7.26 Consider the escaping spline in Figure 7.6 and set $f(x, p) = x$. Fix any point in which the program's execution passes through the loop head and let a be the value of x at this moment. Then the expected value of x after performing one loop iteration is $0 \cdot \frac{1}{a+1} + (a+1) \cdot \frac{a}{a+1} = a$, so the expected change of x in each loop iteration is zero. Moreover, in each iteration the value of x decreases by at least 1 with probability $p = \frac{1}{x+1}$. Since p is antitone, it follows that $\{X_i^f\}_{i\in\mathbb{N}_0}$ is a PRSM, and hence the program terminates a.s.

This loop-based use of PRSMs is non-local: we have to analyse the behaviour of f along one whole loop iteration, as opposed to single computational steps. For complex loops, finding the right f and checking its properties might be an intricate process. In McIver et al. (2018), the authors propose proving required properties of f in the *weakest pre-expectation logic,* a formal calculus which extends the classical weakest-precondition reasoning to probabilistic programs. While falling short of automated termination analysis, formalizing the proofs in the formal logic makes use of interactive proof assistants possible, with a potential to achieve provably correct results with significantly decreased human workload.

Remark 7.27 A similar martingale-based approach for proving almost-sure termination of probabilistic while loops is proposed in Huang et al. (2018). Compared with McIver et al. (2018), the martingale-based approach in Huang et al. (2018) can derive asymptotically optimal bounds on *tail probabilities* of program non-termination within a given number of steps, while McIver et al. (2018) cannot derive such probabilities. On the other hand, the approach in McIver et al. (2018)

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1: while x ≥ 1 and y ≥ 1 do
2:     if ⋆ then
3:         x := x + sample(Uniform{−3, 1})
       else
4:         y := y − 1
5:         x := 2x + sample(Uniform{−1, 1})
       fi
```

Figure 7.7 A program without a linear RSM but admitting a LexRSM.

refines that in Huang et al. and can prove the almost-sure termination of probabilistic programs that Huang et al. cannot. Another related approach in Huang et al. (2018) uses Central Limit Theorem to prove almost-sure termination.

7.5.2 Lexicographic Ranking Supermartingales

For some programs (even for those that do terminate in finite expected time), it might be difficult to find an RSM because of a complex control flow structure, which makes the computation go through several phases, each with a different program behaviour.

Example 7.28 Consider the program in Figure 7.7 with an invariant I s.t. $I(1) = \{(x,y) \mid x \geq -2 \wedge y \geq 0\}$, $I(2) = I(3) = I(4) = \{(x,y) \mid x \geq 1 \wedge y \geq 1\}$ and $I(5) = \{(x,y) \mid x \geq 1 \wedge y \geq 0\}$. The program terminates in bounded expected time, as shown by the existence of the following (non-linear) RSM map η: $\eta(i) = (x+2) \cdot 2^y \cdot y - \frac{(i-1)}{2}$ for $i \in \{1,\ldots,4\}$ and $\eta(5) = (x+2) \cdot 2^{y+1} \cdot y + 1$. Next, it is easy to verify, that there is no *linear* RSM map for the program. Intuitively, in the else branch, executing the decrement of y can decrease the value of a linear function only by some constant, and this cannot compensate for the possibly unbounded increase of x caused by doubling.

The absence of a termination certificate within the scope of linear arithmetic is somewhat bothersome, as non-linear reasoning can become computationally hard. In non-probabilistic setting, similar issues were addressed by considering *multi-dimensional* termination certificates. The crucial idea is to consider functions that map the program configurations to real-valued vectors instead of just numbers, such that the value of the vector-valued function strictly decreases in every step w.r.t. some well-founded ordering of the vectors. This in essence entails a certain "decomposition" of the termination certificate: it might happen that a program admits a multi-dimensional certificate where each component is linear, even when no one-dimensional linear certificates exist. Such certificates can often be found

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

via fully automated linear-arithmetic reasoning. A prime example of this concept are *lexicographic* ranking functions (Cook et al., 2013), where the well-founded ordering used is typically the lexicographic ordering on non-negative real vectors.

In the context of probabilistic programs, the lexicographic extension of ranking supermartingales was introduced in Agrawal et al. (2018). We again start with a general mathematical definition and a correctness theorem. In the following, an n-dimensional stochastic process is a sequence $\{\mathbf{X}_i\}_{i=0}^{\infty}$ of n-dimensional random vectors, i.e. each $\mathbf{X}_i$ is a vector whose component is a random variable. We denote by $\mathbf{X}_i[j]$ the j-component of $\mathbf{X}_i$.

Definition 7.29 An n-dimensional real-valued stochastic process $\{\mathbf{X}_i\}_{i=0}^{\infty}$ is a *lexicographic ϵ-ranking supermartingale (ϵ-LexRSM)* adapted to a filtration $\{\mathcal{F}_i\}_{i=0}^{\infty}$ if the following conditions hold:

(i) For each $1 \leq j \leq n$ the 1-dimensional stochastic process $\{\mathbf{X}_i[j]\}_{i=0}^{\infty}$ is adapted to $\{\mathcal{F}_i\}_{i=0}^{\infty}$.

(ii) For each $i \in \mathbb{N}_0$ and $1 \leq j \leq n$ it holds $\mathbf{X}_i[j] \geq 0$, i.e. the process takes values in non-negative real vectors.

(iii) For each $i \in \mathbb{N}_0$ there exists a partition of the set $\{\omega \in \Omega \mid \forall 1 \leq j \leq n, \mathbf{X}_i[j](\omega) > 0\}$ into n subsets $L_1^i, \ldots, L_n^i$, all of them $\mathcal{F}_i$-measurable, such that for each $1 \leq j \leq n$:
- $\mathbb{E}[\mathbf{X}_{i+1}[j] \mid \mathcal{F}_i](\omega) \leq \mathbf{X}_i[j](\omega) - \epsilon$ for each $\omega \in L_j^i$;
- for all $1 \leq j' < j$ we have $\mathbb{E}[\mathbf{X}_{i+1}[j'] \mid \mathcal{F}_i](\omega) \leq \mathbf{X}_i[j'](\omega)$ for each $\omega \in L_j^i$.

Note that we dropped the integrability condition from Definition 7.4. This is because integrability is only needed to ensure that the conditional expectations in the definiton of a (Lex)RSM 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 Agrawal et al. (2018) for details. This is exactly the case in LexRSMs. Waiving the integrability condition might simplify 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 full proof of the following theorem is provided in Agrawal et al. (2018).

Theorem 7.30 *Let $\{\mathbf{X}_i\}_{i=0}^{\infty}$ be a LexRSM adapted to some filtration. Then with probability* 1 *at least one component of the process eventually attains a zero value.*

To apply LexRSMs to a.s. termination proving, let $\mathcal{P}$ be a program and I be an invariant for $\mathcal{P}$.

Definition 7.31 (Lexicographic Ranking Supermartingale Map) Let $\epsilon > 0$. An n-dimensional *lexicographic ϵ-ranking supermartingale map* (ϵ-LexRSM map) for a program $\mathcal{P}$ with an invariant I is a vector function $\overrightarrow{\eta} = (\eta_1, \ldots, \eta_n)$, where each

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

η_i maps configurations of $\mathcal{P}$ to real numbers, such that for each configuration $(\ell, \mathbf{x})$ where $\mathbf{x} \in I(\ell)$ the following conditions are satisfied:

- for all $1 \leq j \leq n$: $\eta_j(\ell, \mathbf{x}) \geq 0$, and if $\ell \neq \ell_{out}$, then $\eta_j(\ell, \mathbf{x}) > 0$; and
- if $\ell \neq \ell_{out}$ and ℓ does not contain a non-deterministic choice, then there exists $1 \leq j \leq n$ such that
  - $\text{pre}_{\eta_j}(\ell, \mathbf{x}) \leq \eta_j(\ell, \mathbf{x}) - \epsilon$, and
  - for all $1 \leq j' < j$ we have $\text{pre}_{\eta_{j'}}(\ell, \mathbf{x}) \leq \eta_{j'}(\ell, \mathbf{x})$;
- $\ell \neq \ell_{out}$ and ℓ contains a non-deterministic choice, then for each $\tilde{\ell} \in \{\ell_{\text{th}}, \ell_{\text{el}}\}$ (where $\ell_{\text{th}}, \ell_{\text{el}}$ are the successor locations in the corresponding branches) there is $1 \leq j \leq n$ such that
  - $\eta_j(\tilde{\ell}, \mathbf{x}) \leq \eta_j(\ell, \mathbf{x}) - \epsilon$, and
  - for all $1 \leq j' < j$ we have $\eta_{j'}(\tilde{\ell}, \mathbf{x}) \leq \eta_{j'}(\ell, \mathbf{x})$.

If additionally each η_i is a linear expression map, then we call $\overrightarrow{\eta}$ a linear ϵ-LexRSM map (ϵ-LinLexRSM).

Using Theorem 7.30, we get the following.

Theorem 7.32 *Let $\mathcal{P}$ be a probabilistic program and I its invariant. Assume that there exists an $\epsilon > 0$ and an n-dimensional ϵ-LexRSM map for $\mathcal{P}$ and I. Then $\mathcal{P}$ terminates almost surely.*

Example 7.33 Consider again the program in Figure 7.7, together with the invariant I from Example 7.28. Then the following 3-dimensional 1-LexRSM map $\overrightarrow{\eta}$ proves that the program terminates a.s.: $\overrightarrow{\eta}(1, \mathbf{x}) = (y + 1, x + 3, 4)$, $\overrightarrow{\eta}(2, \mathbf{x}) = (y + 1, x + 3, 3)$, $\overrightarrow{\eta}(3, \mathbf{x}) = \overrightarrow{\eta}(4, \mathbf{x}) = (y + 1, x + 3, 2)$, $\overrightarrow{\eta}(5, \mathbf{x}) = (y + 2, x + 3, 1)$, and $\overrightarrow{\eta}(\ell_{out}, \mathbf{x}) = (0, 0, 0)$.

(Agrawal et al., 2018) presented an algorithm for synthesis of linear LexRSM maps in affine probabilistic programs with pre-computed invariants. The algorithm is based on a method for finding lexicographic ranking functions presented in Alias et al. (2010). The method attempts to find a LinLexRSM map by computing one component at a time, iteratively employing the algorithm for synthesis of 1-dimensional RSMs (Section 7.4) as a sub-procedure. The method is complete in the sense that if there exists a LinLexRSM map for a program $\mathcal{P}$ with a given invariant I, then the algorithm finds such a map. If guards of all conditional statements and loops in the program are linear assertions (i.e. conjunctions of linear inequalities), then the algorithm runs in time polynomial in the size of $\mathcal{P}$ and I.

We now show that LexRSMs are indeed capable of proving a.s. termination of programs that terminate in infinite expected number of steps.

Example 7.34 Consider the program in Figure 7.8, together with an invariant I such that $I(1) = \{(x, c) \mid x \geq 1 \wedge c \geq 0\}$, $I(2) = I(3) = I(4) = \{(x, c) \mid x \geq 1 \wedge c \geq 1\}$,

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1:  while c ≥ 1 and x ≥ 1 do
2:      if prob(0.5) then
3:          x := 2 · x
        else
4:          c := 0
        fi
    od;
5:  while x ≥ 1 do
6:      x := x − 1
    od
```

Figure 7.8 An example program that is a.s. terminating but with infinite expected termination time.

$I(5) = \{(x,c) \mid x \geq 0\}$, and $I(6) = \{(x,c) \mid x \geq 1\}$. The a.s. termination of the program is witnessed by a linear 1-LexRSM map $\overrightarrow{\eta}$ such that $\overrightarrow{\eta}(1,\mathbf{x}) = (6c+5, 2x+2)$, $\overrightarrow{\eta}(2,\mathbf{x}) = (6c+4, 2x+2)$, $\overrightarrow{\eta}(3,\mathbf{x}) = (6c+6, 2x+2)$, $\overrightarrow{\eta}(4,\mathbf{x}) = (6c, 2x+2)$, $\overrightarrow{\eta}(5,\mathbf{x}) = (1, 2x+2)$, and $\overrightarrow{\eta}(5,\mathbf{x}) = (1, 2x+1)$. However, the program terminates in an infinite expected number of steps: to see this, note that that the expected value of variable x upon reaching the second loop is $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 4 + \cdots = \infty$, and that the time needed to get out of the second loop is equal to the value of x upon entering the loop.

Finally, we remark that (Agrawal et al., 2018) introduced further uses of LexRSMs, such as compositional termination proving (where we prove a.s. termination one loop at a time, proceeding from the innermost ones) and the use of special type of linear LexRSMs for obtaining polynomial bounds on expected termination time.

7.5.3 Quantitative Termination and Safety

Consider the program in Figure 7.9. Due to lines 5–6, the program does not terminate a.s., because there is a positive probability that x hits zero before y falls below 1. However, a closer look shows that such an event, while possible, is unlikely, since x tends to increase and y tends to decrease on average. (Chatterjee et al., 2017) studied martingale-based techniques that can provide lower bounds on termination probabilities of such programs.

First, the paper introduced the concept of *stochastic invariants.*

Definition 7.35 Let (PI, p) be a tuple such that PI is a function mapping each program location to a set of variable valuations and $p \in [0,1]$ is a probability. The tuple (PI, p) is a *stochastic invariant* for a program $\mathcal{P}$ if the following holds: if we

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

```
1: x := 150, y := 100
2: while y ≥ 1 do
3:     x := x + sample(Uniform[−1/4, 1])
4:     y := y + sample(Uniform[−1, 1/4])
5:     while x ≤ 0 do
6:         skip od
   od
```

Figure 7.9 A program with infinitely many reachable configurations which terminates with high probability, but not almost surely, together with a sketch of its pCFG.

denote by $Fail(PI)$ the set of all runs that reach a configuration of the form $(\ell, \mathbf{x})$ with $\mathbf{x} \notin PI(\ell)$, then for all schedulers σ it holds $\mathbb{P}^\sigma(Fail(PI)) \leq 1 - p$.

Example 7.36 Consider the example in Figure 7.9 and a tuple (PI, p) for the program such that $PI(5) = \{(x, y) \mid x \geq \frac{1}{9}\}$, $PI(\ell) = \mathbb{R}^2$ for all the other locations, and $p = 10^{-5}$. Using techniques for analysis of random walks, one can prove that (PI, p) is a stochastic invariant for the program. Below, we will sketch a martingale-based technique that can be used to prove this formally (and automatically).

Intuitively, unlike their classical counterparts, stochastic invariants are not over-approximations of the set of reachable configurations. However, for small p, they can be viewed as good probabilistic approximations of this set, in the sense that the probability of reaching a configuration not belonging to this approximation is small (smaller than p). The following theorem illustrates a possible use of stochastic invariants in probabilistic termination analysis.

Theorem 7.37 *Let $\mathcal{P}$ be a probabilistic program, I a (classical) invariant, and (PI, p) a stochastic invariant for $\mathcal{P}$. Further, let $\eta \colon L \times \mathbb{R}^{|V|} \to \mathbb{R}$ be a mapping such that there exists $\epsilon > 0$ for which the following holds in each configuration $(\ell, \mathbf{x})$ of $\mathcal{P}$:*

- *if $\mathbf{x} \in I(\ell)$, then $\eta(\ell, \mathbf{x}) \geq 0$, and*
- *if $\ell \neq \ell_{out}$ and $\mathbf{x} \in I(\ell) \cap PI(\ell)$, then $\mathrm{pre}_\eta(\ell, \mathbf{x}) \leq \eta(\ell, \mathbf{x}) - \epsilon$.*

Then, under each scheduler σ, the program $\mathcal{P}$ terminates with probability at least $1 - p$.

Proof (Sketch). The map η can be viewed as an RSM map for a modified version of $\mathcal{P}$ which immediately terminates whenever PI is violated. Such a modified program therefore terminates with probability 1. Since (PI, p) is a stochastic invariant, violations of PI can occur with probability at most p, so with probability at least $1 - p$ the modified (and thus also the original) program terminates in an orderly way. □

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

Example 7.38 Let $(PI, 10^{-5})$ be the stochastic invariant from Example 7.36 (concerning Figure 7.9). For the corresponding program we have a classical invariant I such that $I(1) = \{(30, 20)\}$, $I(2) = \{(x, y) \mid x \geq 0 \wedge y \geq 0\}$, $I(3) = \{(x, y) \mid x \geq 0 \wedge y \geq 1\}$, $I(4) = \{(x, y) \mid x \geq -\frac{1}{4} \wedge y \geq 1\}$, $I(5) = \{(x, y) \mid x \geq -\frac{1}{4} \wedge y \geq 0\}$, and $I(6) = \{(x, y) \mid 0 \geq x \geq -\frac{1}{4} \wedge y \geq 0\}$. Consider a map η defined as follows: $\eta(1) = \eta(5) = 16y+3, \eta(2) = 16y+2, \eta(3) = 16y+1, \eta(4) = 16y$, and $\eta(6) = 16y+4$. Then η satisfies the conditions of Theorem 7.37, from which it follows that the program terminates with probability at least 0.99999.

Given an affine probabilistic program and its classical and stochastic invariants, I and (PI, p) (both I and PI being linear), we can check whether there exists a linear RSM map satisfying Theorem 7.37 using virtually the same linear system as in Section 7.4. We just need to take the location-wise intersection I' of I and PI as the input invariant used to construct the linear constraints. Although I' is not a classical invariant, the linear RSM map obtained from solving the constraints satisfies the requirements of Theorem 7.37.

The question, then, is how to prove that a tuple (PI,p) is a stochastic invariant. In (Chatterjee et al., 2017), a concept of *repulsing supermartingales* (RepSMs) was introduced, which can be used to compute upper bounds on the probability of violating PI. RepSMs are inspired by use of martingale techniques in the analysis of one-counter probabilistic systems (Brázdil et al., 2013), and they are in some sense dual to RSMs: they show that a computation is probabilistically *repulsed* away from (rather than attracted to) some set of configurations. As was the case in the preceding martingale-based concepts, RepSMs are defined abstractly as a certain class of stochastic processes, and then applied to program analysis via the notion of *RepSM maps*. For the sake of succinctness, we present here only the latter concept.

Definition 7.39 (Linear repulsing supermartingales) Let $\mathcal{P}$ be a PP with an initial configuration $(\ell_{init}, \mathbf{x}_{init})$, I its invariant, and $C \subseteq L \times \mathbb{R}^{|V|}$ some set of configurations of $\mathcal{P}$. An ϵ-repulsing supermartingale (ϵ-RepSM) map for C supported by I is a mapping $\eta \colon L \times \mathbb{R}^{|V|} \to V$ such that for all configurations $(\ell, \mathbf{x})$ of $\mathcal{P}$ the following holds:

- if $(\ell, \mathbf{x}) \in C$ and $\mathbf{x} \in I(\ell)$, then $\eta(\ell, \mathbf{x}) \geq 0$
- if $(\ell, \mathbf{x}) \notin C$ and $\mathbf{x} \in I(\ell)$, then $pre_\eta(\ell, \mathbf{x}) \leq \eta(\ell, \mathbf{x}) - \epsilon$,
- $\eta(\ell_{init}, \mathbf{x}_{init}) < 0$.

An ϵ-RepSM map supported by I has c-bounded differences if for each pair of locations ℓ, ℓ' and each pair of configurations $(\ell, \mathbf{x})$, $(\ell', \mathbf{x}')$ such that $\mathbf{x} \in I(\ell)$ and $(\ell', \mathbf{x}')$ can be produced by performing a step of computation from $(\ell, \mathbf{x})$ it holds $|\eta(\ell, \mathbf{x}) - \eta(\ell', \mathbf{x}')| \leq c$.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

The following theorem is proved using Azuma's inequality, a concentration bound from martingale theory.

Theorem 7.40 *Let C be a set of configurations of a PP $\mathcal{P}$. Suppose that there exist $\epsilon > 0$, $c > 0$, and a linear ϵ-RepSM map η for C supported by some invariant I such that η has c-bounded differences. Then under each scheduler σ, the probability p_C that the program reaches a configuration from C satisfies*

$$p_C \leq \alpha \cdot \frac{\gamma^{\lceil |\eta(\ell_{init}, \mathbf{x}_{init})|/c \rceil}}{1-\gamma}, \tag{7.1}$$

where $\gamma = \exp\left(-\frac{\epsilon^2}{2(c+\epsilon)^2}\right)$ *and* $\alpha = \exp\left(\frac{\epsilon \cdot |\eta(\ell_{init}, \mathbf{x}_{init})|}{(c+\epsilon)^2}\right)$.

Example 7.41 Consider again the program in Figure 7.9, with the same invariant I as in Example 7.36. Let $C = \{(\ell, (x, y)) \mid \ell = 5 \wedge x \leq \frac{1}{8}\}$. Then the following map η is a 13-bounded 1-RepSM map for C: $\eta(1) = \eta(5) = -16x + 2$, $\eta(2) = 16x + 1$, $\eta(3) = -16x$, and $\eta(4) = \eta(6) = -16x + 3$. Applying Theorem 7.40 yields that C is reached with probbaility at most $\exp\left(\frac{-116154}{392} - \frac{1}{392} \cdot \lceil \frac{116154}{14} \rceil\right) / (1 - \exp(-1/392)) \approx 1.2 \cdot 10^{-6} \leq 10^{-5}$. Now for the map PI in Example 7.36 it holds that violating PI entails reaching C, which shows that $(PI, 10^{-5})$ is indeed a stochastic invariant.

Checking whether there is a linear RepSM map (supported by a given linear invariant) for a set of configurations defined by a given system of linear constraints can be again performed by linear constraint solving, using techniques analogous to Section 7.4.

Finally, we mention that RepSM maps can be used to *refute* almost-sure and finite termination.

Theorem 7.42 *Let C be a set of terminal configurations of a program $\mathcal{P}$, i.e. of those configurations where the corresponding location is terminal. Suppose that there exist $\epsilon \geq 0$, $c > 0$, and a linear ϵ-RepSM map η for C supported by some invariant I such that η has c-bounded differences. Then, no matter which scheduler is used, $\mathcal{P}$ does not terminate in finite expected time. Moreover, if $\epsilon > 0$, then $\mathcal{P}$ terminates with probability less than 1 under every scheduler.*

Example 7.43 Consider the symmetric random walk (Figure 7.5) together with an invariant $x \geq 0$ in every location. Assuming that initially $x > 1$, the mapping which to each non-terminal configuration (ℓ, x) assigns the number $-x + 1$, while each terminal configuration is assigned zero, is a 0-RepSM for the set of terminal configurations, with 1-bounded differences. Hence, the symmetric random walk indeed does not terminate in finite expected time.

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

7.6 Related Works

Termination approaches. In Sharir et al. (1984) the termination of concurrent probabilistic programs with finite-state space was considered as a fairness problem, and the precise probabilities did not play a role in termination. A sound and complete method for proving termination of finite state programs was given in Esparza et al. (2012). The above approaches do not apply to programs with countable state space in general. For countable state space and almost-sure termination a characterization through fixed-point theory was presented in Hart and Sharir (1985). The analysis of non-probabilistic program and the termination problem has also been extensively studied (Bradley et al., 2005a; Colón and Sipma, 2001; Podelski and Rybalchenko, 2004; Sohn and Gelder, 1991; Bradley et al., 2005b; Cook et al., 2013; Lee et al., 2001). The focus of this chapter was to present the key aspects of martingale-based approaches for termination of infinite-state probabilistic programs.

Proof-rule based approach. In this work we consider the supermartingale based approach for probabilistic programs, and an alternative approach is based on the notion of proof rules (Kaminski et al., 2016; Hesselink, 1993; Olmedo et al., 2016). These two approaches complement each other, and have their own advantages. The proof-rule based approach itself does not depend on classical invariants (see for exampleColón et al., 2003; Cousot, 2005) and is capable of establishing quantitative invariants, whereas the supermartingale approach usually requires classical invariants to achieve automation (Chakarov and Sankaranarayanan, 2013; Chatterjee et al., 2016b,a). In contrast, the advantages of the supermartingale-based approach are: (a) the supermartingale based approach leads to automated and algorithmic approaches; (b) tail bounds can be obtained through supermartingales using the mathematical results such as Azuma's inequality or Hoeffding's inequality (Chatterjee et al., 2016a), and (c) in presence of conditioning, proof-rules cannot be applied to non-deterministic programs as the schedulers are not necessarily local, whereas ranking supermartingales can consider non-determinism as the semantics is through general MDPs and general schedulers.

Other results. Martingales can also be used for analysis of properties other than termination over probabilistic programs (e.g., probabilistic invariants (Barthe et al., 2016b) or proving recurrence/persistence/reactivity properties (Chakarov et al., 2016; Dimitrova et al., 2016; Chatterjee et al., 2017)). Other prominent approaches for analyzing probabilistic programs include: (a) techniques based on coupling proofs and their applications in analysis of differential privacy and probabilistic sensitivity (Barthe et al., 2017, 2018, 2016a); (b) static-analysis based approaches (Sankaranarayanan et al., 2013; Cusumano-Towner et al., 2018; Wang et al., 2018); (c)

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

potential-function based approaches for cost analysis (Chatterjee et al., 2018b; Ngo et al., 2018). Moreover, the semantics of probabilistic programs is studied in Bichsel et al. (2018) and Staton et al. (2016).

7.7 Conclusion and Future Directions

In this chapter we present the main results related to martingale-based approach for termination analysis of probabilistic programs. There are several interesting directions of future work. First, for analysis of probabilistic programs with angelic non-determinism there is a complexity gap for linear RSMs and an interesting theoretical question is to close the complexity gap. Second, while the martingale-based approach and other approaches such as proof-rule based approach each has its own advantages, techniques for combining them is another interesting direction of future work. Finally, practical directions of building scalable tools using algorithmic results for martingales in conjunction with other methods such as compositional analysis are also largely unexplored.

Acknowledgements

Krishnendu Chatterjee is supported by the Austrian Science Fund (FWF) NFN Grant No. S11407-N23 (RiSE/SHiNE), and COST Action GAMENET. Hongfei Fu is supported by the National Natural Science Foundation of China (NSFC) Grant No. 61802254. Petr Novotný is supported by the Czech Science Foundation grant No. GJ19-15134Y.

References

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

Alias, Christophe, Darte, Alain, Feautrier, Paul, and Gonnord, Laure. 2010. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. Pages 117–133 of: *Proceedings of the 17th International Conference on Static Analysis*. SAS'10. Berlin, Heidelberg: Springer-Verlag.

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

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking*. MIT Press.

Barthe, Gilles, Gaboardi, Marco, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2016a. Proving Differential Privacy via Probabilistic Couplings. *In:* Grohe et al. (2016).

https://doi.org/10.1017/9781108770750.008 Published online by Cambridge University Press

Barthe, Gilles, Espitau, Thomas, Fioriti, Luis María Ferrer, and Hsu, Justin. 2016b. Synthesizing Probabilistic Invariants via Doob's Decomposition. Pages 43–61 of: Chaudhuri, Swarat, and Farzan, Azadeh (eds), *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*. Lecture Notes in Computer Science, vol. 9779. Springer.

Barthe, Gilles, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2017. Coupling proofs are probabilistic product programs. Pages 161–174 of: Castagna, Giuseppe, and Gordon, Andrew D. (eds), *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. ACM.

Barthe, Gilles, Espitau, Thomas, Grégoire, Benjamin, Hsu, Justin, and Strub, Pierre-Yves. 2018. Proving expected sensitivity of probabilistic programs. *PACMPL*, **2**(POPL), 57:1–57:29.

Bichsel, Benjamin, Gehr, Timon, and Vechev, Martin T. 2018. Fine-Grained Semantics for Probabilistic Programs. Pages 145–185 of: Ahmed, Amal (ed), *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. Lecture Notes in Computer Science, vol. 10801. Springer.

Billingsley, P. 1995. *Probability and Measure*. 3rd edn. Wiley.

Bournez, O., and Garnier, F. 2005. Proving Positive Almost-Sure Termination. Pages 323–337 of: *International Conference on Rewriting Techniques and Applications, RTA'05*. Springer.

Bradley, A. R., Manna, Z., and Sipma, H. B. 2005a. Linear Ranking with Reachability. Pages 491–504 of: *International Conference on Computer Aided Verification, CAV'05*. Springer.

Bradley, A. R., Manna, Z., and Sipma, H. B. 2005b. The Polyranking Principle. Pages 1349–1361 of: *International Colloquium on Automata, Languages, and Programming, ICALP'05*. Springer.

Brázdil, Tomáš, Brožek, Václav, Etessami, Kousha, and Kučera, Antonín. 2013. Approximating the termination value of one-counter MDPs and stochastic games. *Information and Computation*, **222**, 121–138.

Chakarov, A., and Sankaranarayanan, S. 2013. Probabilistic Program Analysis with Martingales. Pages 511–526 of: *CAV*.

Chakarov, Aleksandar, Voronin, Yuen-Lam, and Sankaranarayanan, Sriram. 2016. Deductive proofs of almost sure persistence and recurrence properties. Pages 260–279 of: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'16*. Springer.

Chatterjee, K., Fu, H., Novotný, P., and Hasheminezhad, R. 2016a. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. Pages 327–342 of: *POPL*.

Chatterjee, K., Fu, H., and Goharshady, A. K. 2016b. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. Pages 3–22 of: *CAV*.

Chatterjee, K., Novotný, P., and Žikelić, Đ. 2017. Stochastic Invariants for Probabilistic Termination. Pages 145–160 of: *POPL*.

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

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

Chatterjee, Krishnendu, Fu, Hongfei, Goharshady, Amir Kafshdar, and Okati, Nastaran. 2018b. Computational Approaches for Stochastic Shortest Path on Succinct MDPs. Pages 4700–4707 of: Lang, Jérôme (ed), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* ijcai.org.

Colón, M., and Sipma, H. 2001. Synthesis of Linear Ranking Functions. Pages 67–81 of: *TACAS*.

Colón, M., Sankaranarayanan, S., and Sipma, H. 2003. Linear Invariant Generation Using Non-linear Constraint Solving. Pages 420–432 of: *CAV*.

Cook, B., See, A., and Zuleger, F. 2013. Ramsey vs. Lexicographic Termination Proving. Pages 47–61 of: *TACAS*.

Cousot, P. 2005. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. Pages 1–24 of: *VMCAI*.

Cousot, Patrick, and Cousot, Radhia. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. Pages 238–252 of: Graham, Robert M., Harrison, Michael A., and Sethi, Ravi (eds), *POPL*. ACM.

Cplex. 2010. *IBM ILOG CPLEX Optimizer Interactive Optimizer Community Edition 12.6.3.0*. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Cusumano-Towner, Marco, Bichsel, Benjamin, Gehr, Timon, Vechev, Martin T., and Mansinghka, Vikash K. 2018. Incremental inference for probabilistic programs. *In:* Foster and Grossman (2018).

Dimitrova, Rayna, Fioriti, Luis María Ferrer, Hermanns, Holger, and Majumdar, Rupak. 2016. Probabilistic CTL*: The Deductive Way. Pages 280–296 of: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'16*. Springer.

Durrett, R. 1996. *Probability: Theory and Examples (Second Edition)*. Duxbury Press.

Esparza, J., Gaiser, A., and Kiefer, S. 2012. Proving Termination of Probabilistic Programs Using Patterns. Pages 123–138 of: *CAV*.

Farkas, J. 1894. A Fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikaiés Természettudományi Értesitö*, **12**, 457–472.

Fioriti, L. M. F., and Hermanns, H. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. Pages 489–501 of: *POPL*.

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

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

Foster, Jeffrey S., and Grossman, Dan (eds). 2018. *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. ACM.

Fu, Hongfei, and Chatterjee, Krishnendu. 2019. Termination of Nondeterministic Probabilistic Programs. Pages 468–490 of: Enea, Constantin, and Piskac, Ruzica (eds), *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings*. Lecture Notes in Computer Science, vol. 11388. Springer.

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

Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. 2014. Probabilistic programming. Pages 167–181 of: *FOSE*.

Grohe, Martin, Koskinen, Eric, and Shankar, Natarajan (eds). 2016. *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. ACM.

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

Hesselink, W. H. 1993. Proof Rules for Recursive Procedures. *Formal Asp. Comput.*, **5**(6), 554–570.

Howard, H. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Huang, Mingzhang, Fu, Hongfei, and Chatterjee, Krishnendu. 2018. New Approaches for Almost-Sure Termination of Probabilistic Programs. Pages 181–201 of: Ryu, Sukyoung (ed), *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*. Lecture Notes in Computer Science, vol. 11275. Springer.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. 1996. Reinforcement learning: A survey. *JAIR*, **4**, 237–285.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, **101**(1), 99–134.

Kaminski, B. L., and Katoen, J.-P. 2015. On the Hardness of Almost-Sure Termination. Pages 307–318 of: *MFCS*.

Kaminski, B. L., Katoen, J.-P., Matheja, C., and Olmedo, F. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. Pages 364–389 of: *ESOP*.

Kemeny, J.G., Snell, J.L., and Knapp, A.W. 1966. *Denumerable Markov Chains*. D. Van Nostrand Company.

Kwiatkowska, M. Z., Norman, G., and Parker, D. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. Pages 585–591 of: *CAV*. LNCS 6806.

Lee, C. S., Jones, N. D., and Ben-Amram, A. M. 2001. The size-change principle for program termination. Pages 81–92 of: *POPL*.

Lpsolve. 2016. *lp_solve 5.5.2.3*. http://lpsolve.sourceforge.net/5.5/.

McIver, A., and Morgan, C. 2004. Developing and Reasoning About Probabilistic Programs in *pGCL*. Pages 123–155 of: *PSSE*.

McIver, A., and Morgan, C. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer.

McIver, A., Morgan, C., Kaminski, B. L., and Katoen, J.-P. 2018. A new proof rule for almost-sure termination. *PACMPL*, **2**(POPL), 33:1–33:28.

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

Neuhäußer, M., Stoelinga, M., and Katoen, J.-P.. 2009. Delayed Nondeterminism in Continuous-Time Markov Decision Processes. Pages 364–379 of: *Foundations of Software Science and Computational Structures (FOSSACS 2009)*. Lecture Notes in Computer Science, vol. 5504. Springer.

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

Ngo, Van Chan, Carbonneaux, Quentin, and Hoffmann, Jan. 2018. Bounded expectations: resource analysis for probabilistic programs. *In:* Foster and Grossman (2018).

Olmedo, F., Kaminski, B. L., Katoen, J.-P., and Matheja, C. 2016. Reasoning about Recursive Probabilistic Programs. Pages 672–681 of: *LICS*.

Paz, A. 1971. *Introduction to probabilistic automata (Computer science and applied mathematics)*. Academic Press.

Podelski, A., and Rybalchenko, A. 2004. A Complete Method for the Synthesis of Linear Ranking Functions. Pages 239–251 of: *VMCAI*.

Rabin, M.O. 1963. Probabilistic automata. *Inf. & Control*, **6**, 230–245.

Sankaranarayanan, S., Chakarov, A., and Gulwani, S. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. Pages 447–458 of: *PLDI*.

Scheiderer, Claus. 2008. Positivity and Sums of Squares: A Guide to Recent Results. *The IMA Volumes in Mathematics and its Applications*, **149**, 271–324.

Schrijver, Alexander. 2003. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

Sharir, M., Pnueli, A., and Hart, S. 1984. Verification of Probabilistic Programs. *SIAM J. Comput.*, **13**(2), 292–314.

Sohn, K., and Gelder, A. V. 1991. Termination Detection in Logic Programs using Argument Sizes. Pages 216–226 of: *PODS*.

Staton, Sam, Yang, Hongseok, Wood, Frank D., Heunen, Chris, and Kammar, Ohad. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. *In:* Grohe et al. (2016).

Turing, Alan Mathison. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, **2**(1), 230–265.

Wang, Di, Hoffmann, Jan, and Reps, Thomas W. 2018. PMAF: an algebraic framework for static analysis of probabilistic programs. *In:* Foster and Grossman (2018).

Williams, D. 1991. *Probability with Martingales*. Cambridge University Press.