

# Termination Analysis of Probabilistic Programs through Positivstellensatz's<sup>\*</sup>

Krishnendu Chatterjee<sup>1</sup>, Hongfei Fu<sup>1,2</sup>, and Amir Kafshdar Goharshady<sup>1</sup>

<sup>1</sup> IST Austria

<sup>2</sup> State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, P.R. China

**Abstract.** We consider nondeterministic probabilistic programs with the most basic liveness property of termination. We present efficient methods for termination analysis of nondeterministic probabilistic programs with polynomial guards and assignments. Our approach is through synthesis of polynomial ranking supermartingales, that on one hand significantly generalizes linear ranking supermartingales and on the other hand is a counterpart of polynomial ranking-functions for proving termination of nonprobabilistic programs. The approach synthesizes polynomial ranking-supermartingales through Positivstellensatz's, yielding an efficient method which is not only sound, but also semi-complete over a large subclass of programs. We show experimental results to demonstrate that our approach can handle several classical programs with complex polynomial guards and assignments, and can synthesize efficient quadratic ranking-supermartingales when a linear one does not exist even for simple affine programs.

## 1 Introduction

*Probabilistic Programs.* Classic imperative programs extended with *random-value generators* gives rise to probabilistic programs. Probabilistic programs provide the appropriate framework to model applications ranging from randomized algorithms [39,18], to stochastic network protocols [5,35], to robot planning [34,31], etc. Nondeterminism plays a crucial role in modeling, such as, to model behaviors over which there is no control, or for abstraction. Thus nondeterministic probabilistic programs are crucial in a huge range of problems, and hence their formal analysis has been studied across disciplines, such as probability theory and statistics [19,29,33,43,40], formal methods [5,35], artificial intelligence [32,31], and programming languages [11,22,44,20].

*Basic Termination Questions.* Besides safety properties, the most basic property for analysis of programs is the liveness property. The most basic and widely used notion of liveness for programs is *termination*. In absence of probability (i.e., for

---

\* A conference version of the paper appears in [12]

nonprobabilistic programs), the synthesis of *ranking functions* and proof of termination are equivalent [23], and numerous approaches exist for synthesis of ranking functions for nonprobabilistic programs [9,14,41,49]. The most basic extension of the termination question for probabilistic programs is the *almost-sure termination* question which asks whether a program terminates with probability 1. Another fundamental question is about *finite termination* (aka positive almost-sure termination [22,8]) which asks whether the expected termination time is finite. The next interesting question is the *concentration* bound computation problem that asks to compute a bound  $M$  such that the probability that the termination time is below  $M$  is concentrated, or in other words, the probability that the termination time exceeds the bound  $M$  decreases exponentially.

*Previous Results.* We discuss the relevant previous results for termination analysis of probabilistic programs.

- *Probabilistic Programs.* First, quantitative invariants was introduced to establish termination of discrete probabilistic programs with demonic nondeterminism [36,37], This was extended in [11] to *ranking supermartingales* resulting in a sound (but not complete) approach to prove almost-sure termination of probabilistic programs without nondeterminism but with integer- and real-valued random variables from distributions like uniform, Gaussian, and Poisson, etc. For probabilistic programs with countable state-space and without nondeterminism, the Lyapunov ranking functions provide a sound and complete method for proving finite termination [8,24]. Another sound method is to explore bounded-termination with exponential decrease of probabilities [38] through abstract interpretation [16]. For probabilistic programs with nondeterminism, a sound and complete characterization for finite termination through ranking-supermartingale is obtained in [22]. Ranking supermartingales thus provide a very powerful approach for termination analysis of probabilistic programs.
- *Ranking Functions/supermartingales Synthesis.* Synthesis of linear ranking-functions/ranking-supermartingales has been studied extensively in [41,14,11,13]. In context of probabilistic programs, the algorithmic study of synthesis of linear ranking supermartingales for probabilistic programs (cf. [11]) and probabilistic programs with nondeterminism (cf. our previous result [13]) has been studied. The major technique adopted in these results is Farkas’ Lemma [21] which serves as a complete reasoning method for linear inequalities. Beyond linear ranking functions, polynomial ranking functions have also been considered. Heuristic synthesis method of polynomial ranking-functions is studied in [4,10]: Cook *et al.* [4] checked termination of deterministic polynomial programs by detecting divergence on program variables and Bradley *et al.* [10] extended to nondeterministic programs through an analysis on finite differences over transitions. More general methods for deterministic polynomial programs are given by [15,48] where Cousot [15] uses Lagrangian Relaxation, and Shen *et al.* [48] use Putinar’s Positivstellensatz [42]. Complete methods of synthesizing polynomial

ranking-functions for nondeterministic programs are studied by Yang *et al.* [52], where a complete method through root classification/real root isolation of semi-algebraic systems and quantifier elimination is proposed.

To summarize, while many different approaches has been studied, the algorithmic study of synthesis of ranking supermartingales for probabilistic programs has only been limited to linear ranking supermartingales. For example, [11] presents a method of synthesis of linear ranking supermartingales for probabilistic programs without nondeterminism, and identifies synthesis of more general nonlinear supermartingales, or extension to probabilistic programs with nondeterminism as important challenges. While the approach of [11] has been extended to probabilistic programs with nondeterminism in our previous result [13], it is restricted to linear ranking supermartingales. Hence there is no algorithmic approach to handle nonlinear ranking supermartingales even for probabilistic programs without nondeterminism.

*Our Contributions.* Our contributions are as follows:

1. *Polynomial Ranking Supermartingales.* First, we extend the notion of linear ranking supermartingales (LRSM) to polynomial ranking supermartingales (pRSM). We show (by a straightforward extension of LRSM) that pRSM implies both almost-sure as well as finite termination.
2. *Positivstellensatz's.* Second, we conduct a detailed investigation on the application of Positivstellensatz's (German for "positive-locus-theorem" which is related to polynomials over semialgebraic sets) (cf. Sect. 5.1) to synthesis of pRSMs over nondeterministic probabilistic programs. To the best of our knowledge, this is the first result which demonstrates the synthesis of a polynomial subclass of ranking supermartingales through Positivstellensatz's.
3. *New Approach for Non-probabilistic Programs.* Our results also extend existing results for nonprobabilistic programs. We present the first result that uses Schmüdgen's Positivstellensatz [46] and Handelman's Theorem [26] to synthesize polynomial ranking-functions for nonprobabilistic programs.
4. *Efficient Approach.* The previous complete method [52] suffers from high computational complexity due to the use of quantifier elimination. In contrast, our approach (sound but not complete) is efficient since the synthesis can be accomplished through linear or semi-definite programming, which can mostly be solved in polynomial time in the problem size [25]. In particular, our approach does not require quantifier elimination, and works for nondeterministic probabilistic programs.
5. *Experimental Results.* We demonstrate the effectiveness of our approach on several classical examples. We show that on classical examples, such as Gambler's Ruin, and Random Walk, our approach can synthesize a pRSM efficiently. For these examples, LRSMs do not exist, and many of them cannot be analysed efficiently by previous approaches.

*Technical Contributions and Novelty.* The main technical contributions and novelty are:

1. While Farkas' Lemma and Motzkin's Transposition Theorem are standard techniques to linear ranking functions or linear ranking supermartingales, they are not sufficient for pRSMs. Instead, our technical contributions is to use various Positivstellensatz's to synthesize pRSMs.
2. Even for nonprobabilistic programs, only a limited number of Positivstellensatz's have been used, e.g., [48]; some of the Positivstellensatz's we use (such as Schmüdgen's Positivstellensatz [46] and Handelman's Theorem [26]) have not even been used in the context of nonprobabilistic programs.

In summary, we study the use of Positivstellensatz's for the first time for probabilistic programs, and for some of them, even for the first time for nonprobabilistic programs, and show that how they can be used for efficient algorithms for program analysis.

*Organization of the Paper.* In Sect. 2, we present the syntax and semantics of probabilistic programs. In Sect. 3, we define the problems to be studied. In Sect. 4, we develop the notion of polynomial ranking supermartingale. Then in Sect. 5, we present Positivstellensatz and our algorithm to synthesize polynomial ranking supermartingales. In Sect. 6, we present our experimental results. Finally, Sect. 7 concludes the paper.

## 2 Probabilistic Programs

### 2.1 Basic Notations and Concepts

For a set  $A$ , we denote by  $|A|$  the cardinality of  $A$ . We denote by  $\mathbb{N}$ ,  $\mathbb{N}_0$ ,  $\mathbb{Z}$ , and  $\mathbb{R}$  the sets of all positive integers, non-negative integers, integers, and real numbers, respectively. We use boldface notation for vectors, e.g.  $\mathbf{x}$ ,  $\mathbf{y}$ , etc, and we denote an  $i$ -th component of a vector  $\mathbf{x}$  by  $\mathbf{x}[i]$ .

*Polynomial Predicates.* Let  $X$  be a finite set of variables endowed with a fixed linear order under which we have  $X = \{x_1, \dots, x_{|X|}\}$ . We denote the set of real-coefficient polynomials by  $\mathfrak{R}[x_1, \dots, x_{|X|}]$  or  $\mathfrak{R}[X]$ . A *polynomial constraint* over  $X$  is a logical formula of the form  $g_1 \bowtie g_2$ , where  $g_1, g_2$  are polynomials over  $X$  and  $\bowtie \in \{<, \leq, >, \geq\}$ . A *propositional polynomial predicate* over  $X$  is a propositional formula whose all atomic propositional literals are either *true*, *false* or polynomial constraints over  $X$ . The validity of the satisfaction assertion  $\mathbf{x} \models \phi$  between a vector  $\mathbf{x} \in \mathbb{R}^{|X|}$  (interpreted in the way that the value for  $x_j$  ( $1 \leq j \leq |X|$ ) is  $\mathbf{x}[j]$ ) and a propositional polynomial predicate  $\phi$  is defined in the standard way w.r.t polynomial evaluation and normal semantics for logical connectives (cf. Appendix A). The satisfaction set of a propositional polynomial predicate  $\phi$  is defined as  $\llbracket \phi \rrbracket := \{\mathbf{x} \in \mathbb{R}^{|X|} \mid \mathbf{x} \models \phi\}$ . For more on polynomials (e.g., polynomial evaluation and arithmetic over polynomials), we refer to the textbook [30, Chapter 3].

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

*Random Variables and Filtrations.* A *random variable*  $X$  in a probability space  $(\Omega, \mathcal{F}, \mathbb{P})$  is an  $\mathcal{F}$ -measurable function  $X: \Omega \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ , i.e., a function satisfying the condition that for all  $d \in \mathbb{R} \cup \{+\infty, -\infty\}$ , the set  $\{\omega \in \Omega \mid X(\omega) \leq d\}$  belongs to  $\mathcal{F}$ . The *expected value* of a random variable  $X$ , denote by  $\mathbb{E}(X)$ , is defined as the Lebesgue integral of  $X$  with respect to  $\mathbb{P}$ , i.e.,  $\mathbb{E}(X) := \int X \, d\mathbb{P}$ ; the precise definition of Lebesgue integral is somewhat technical and is omitted here (cf. [7, Chapter 5] for a formal definition). A *filtration* of a probability space  $(\Omega, \mathcal{F}, \mathbb{P})$  is an infinite sequence  $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$  of  $\sigma$ -algebras over  $\Omega$  such that  $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$  for all  $n \in \mathbb{N}_0$ .

## 2.2 Probabilistic Programs

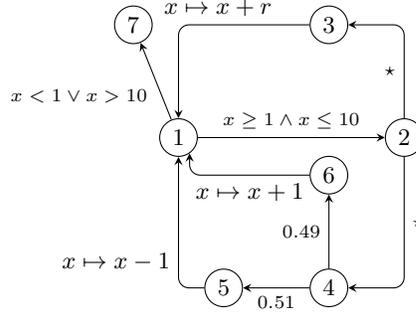
**The Syntax.** The class of probabilistic programs we consider encompasses basic programming mechanisms such as assignment statement (indicated by ‘:=’), while-loop, if-branch, basic probabilistic mechanisms such as probabilistic branch (indicated by ‘prob’) and random sampling, and demonic nondeterminism indicated by ‘ $\star$ ’. Variables (or identifiers) of a probabilistic program are of *real* type, i.e., values of the variables are real numbers; moreover, variables are classified into *program* and *sampling* variables, where program variables receive their values through assignment statements and sampling variables do through random samplings. We consider that each sampling variable  $r$  is *bounded*, i.e., associated with a one-dimensional cumulative distribution function  $\mathcal{Y}_r$  and a non-empty bounded interval  $\text{supp}_r$  such that any random variable  $z$  which respects  $\mathcal{Y}_r$  satisfies that  $z$  lies in the bounded interval with probability 1. Due to space limit, we put details (e.g., grammar) in Appendix B. An example probabilistic program is illustrated in Example 1.

*Example 1.* Consider the running example depicted in Fig. 1, where  $r$  is a sampling variable with the two-point distribution  $\{1 \mapsto 0.5, -1 \mapsto 0.5\}$  where the probability to take values 1 and  $-1$  are both 0.5. The probabilistic program models a scenario of Gambler’s Ruin where the gambler has initial money  $x$  and repeats gambling until he wins more than 10 or lose all his money. The result of a gamble is nondeterministic: either win 1 with probability 0.5 (nondeterministic branch); or lose with probability 0.51 (the probabilistic branch). The numbers 1 – 7 on the left are the program counters for the program, where 1 is the initial program counter and 7 the terminal program counter.

```

1: while  $1 \leq x \wedge x \leq 10$  do
2:   if  $\star$  then
3:      $x := x + r$ 
4:   else
5:     if prob(0.51) then
6:        $x := x - 1$ 
7:     else
8:        $x := x + 1$ 
9:     fi
10:  fi
11: od

```



**Fig. 2.** The CFG of the Running Example

**Fig. 1.** The Running Example: Gambler’s Ruin

**The semantics.** We use control flow graphs to capture the semantics of probabilistic programs, which we define below.

**Definition 1 (Control Flow Graph).** A control flow graph (CFG) is a tuple  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  with the following components:

- $L$  is a finite set of labels partitioned into four pairwise-disjoint subsets  $L_d$ ,  $L_p$ ,  $L_c$  and  $L_a$  of demonic, probabilistic, conditional-branching (branching for short) and assignment labels, resp.; and  $\perp$  is a special label not in  $L$  called the terminal label;
- $X$  and  $R$  are disjoint finite sets of real-valued program and sampling variables respectively ;
- $\mapsto$  is a transition relation in which every member (called transition) is a tuple of the form  $(\ell, \alpha, \ell')$  for which  $\ell$  (resp.  $\ell'$ ) is the source label (resp. target label) in  $L$  and  $\alpha$  is either a real number in  $(0, 1)$  if  $\ell \in L_p$ , or  $\star$  if  $\ell \in L_d$ , or a propositional polynomial predicate if  $\ell \in L_c$ , or an update function  $f: \mathbb{R}^{|X|} \times \mathbb{R}^{|R|} \rightarrow \mathbb{R}^{|X|}$  if  $\ell \in L_a$ .

W.l.o.g, we assume that  $L \subseteq \mathbb{N}_0$ . Intuitively, labels in  $L_d$  correspond to demonic statements indicated by ‘ $\star$ ’; labels in  $L_p$  correspond to probabilistic-branching statements indicated by ‘**prob**’; labels in  $L_c$  correspond to conditional-branching statements indicated by some propositional polynomial predicate; labels in  $L_a$  correspond to assignments indicated by ‘:=’ and the terminal label  $\perp$  denotes the termination of a program. The transition relation  $\mapsto$  specifies the transitions between labels together with the additional information specific to different types of labels. The update functions are interpreted as follows: we first fix two linear orders on  $X$  and  $R$  so that  $X = \{x_1, \dots, x_{|X|}\}$  and  $R = \{r_1, \dots, r_{|R|}\}$ , interpreting each vector  $\mathbf{x} \in \mathbb{R}^{|X|}$  (resp.  $\mathbf{r} \in \mathbb{R}^{|R|}$ ) as a *valuation* of program (resp. sampling) variables in the sense that the value of  $x_j$  (resp.  $r_j$ ) is  $\mathbf{x}[j]$  (resp.  $\mathbf{r}[j]$ );

then each update function  $f$  is interpreted as a function which transforms a valuation  $\mathbf{x} \in \mathbb{R}^{|X|}$  before the execution of an assignment statement into  $f(\mathbf{x}, \mathbf{r})$  after the execution of the assignment statement, where  $\mathbf{r}$  is the valuation on  $R$  obtained from a sampling before the execution of the assignment statement.

It is intuitively clear that any probabilistic program can be naturally transformed into a CFG. Informally, each label represents a program location in an execution of a probabilistic program for which the statement of the program location is the next to be executed. A detailed construction is provided in Appendix C.

*Example 2.* The control flow graph of the running example (Example 1) is depicted in Fig. 2, where vertices correspond to labels specified in Fig. 1.

Now we present the semantics of probabilistic programs. In the rest of the section, we fix a probabilistic program  $P$  with the set  $X = \{x_1, \dots, x_{|X|}\}$  of program variables and the set  $R = \{r_1, \dots, r_{|R|}\}$  of sampling variables, and let  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  be its associated CFG. We also fix  $\ell_0$  and resp.  $\mathbf{x}_0$  to be the label corresponding to the first statement to be executed in  $P$  and resp. the initial valuation of program variables.

*The Semantics.* A *configuration* (for  $P$ ) is a tuple  $(\ell, \mathbf{x})$  where  $\ell \in L \cup \{\perp\}$  and  $\mathbf{x} \in \mathbb{R}^{|X|}$ . A *finite path* (of  $P$ ) is a finite sequence of configurations  $(\ell_0, \mathbf{x}_0), \dots, (\ell_k, \mathbf{x}_k)$  such that for all  $0 \leq i < k$ , either (i)  $\ell_{i+1} = \ell_i = \perp$  and  $\mathbf{x}_i = \mathbf{x}_{i+1}$  (i.e., the program terminates); or (ii) there exist  $(\ell_i, \alpha, \ell_{i+1}) \in \mapsto$  and  $\mathbf{r} \in \{\mathbf{r}' \mid \forall r \in R. \mathbf{r}'(r) \in \text{supp}_r\}$  such that one of the following conditions hold: (a)  $\ell_i \in L_p \cup L_d$  and  $\mathbf{x}_i = \mathbf{x}_{i+1}$  (probabilistic or demonic transitions), (b)  $\ell_i \in L_c$ ,  $\mathbf{x}_i = \mathbf{x}_{i+1}$  and  $\mathbf{x}_i \models \alpha$  (conditional-branch transitions), (c)  $\ell_i \in L_a$  and  $\mathbf{x}_{i+1} = \alpha(\mathbf{x}_i, \mathbf{r})$  (assignment transitions). A *run* (of  $P$ ) is an infinite sequence of configurations whose all finite prefixes are finite paths over  $P$ . A configuration  $(\ell, \mathbf{x})$  is *reachable* from the initial configuration  $(\ell_0, \mathbf{x}_0)$  if there exists a finite path  $(\ell_0, \mathbf{x}_0), \dots, (\ell_k, \mathbf{x}_k)$  such that  $(\ell, \mathbf{x}) = (\ell_k, \mathbf{x}_k)$ .

The probabilistic feature of  $P$  can be captured by constructing a suitable probability measure over the set of all its runs. However, before this can be done, nondeterminism in  $P$  needs to be resolved by some *scheduler*.

**Definition 2 (Scheduler).** A scheduler (for  $P$ ) is a function which assigns to every finite path  $(\ell_0, \mathbf{x}_0), \dots, (\ell_k, \mathbf{x}_k)$  with  $\ell_k \in L_d$  a transition  $\ell_k \mapsto$  with source label  $\ell_k$ .

The behaviour of  $P$  under a scheduler  $\sigma$  is standard: at each step,  $P$  first samples a real number for each sampling variable and then evolves to the next step according to its CFG or the scheduler choice (the details are in Appendix D). In this way, the scheduler and random choices/samplings produce a run over  $P$ . Moreover, each scheduler  $\sigma$  induces a unique probability measure  $\mathbb{P}^\sigma$  over the

runs of  $P$ . In the rest of the paper, we will use  $\mathbb{E}^\sigma(\cdot)$  to denote the expected values of random variables under  $\mathbb{P}^\sigma$ .

*Random Variables and Filtrations over Runs.* We define the following (vectors of) random variables on the set of runs of  $P$ :  $\{\theta_n^P\}_{n \in \mathbb{N}_0}$ ,  $\{\bar{\mathbf{x}}_n^P\}_{n \in \mathbb{N}_0}$  and  $\{\bar{\mathbf{r}}_n^P\}_{n \in \mathbb{N}_0}$ : each  $\theta_n^P$  is the random variable representing the (integer-valued) label at the  $n$ -th step; each  $\bar{\mathbf{x}}_n^P$  is the vector of random variables such that each  $\bar{\mathbf{x}}_n^P[i]$  is the random variable representing the value of the program variable  $x_i$  at the  $n$ -th step; and each  $\bar{\mathbf{r}}_n^P[i]$  is the random variable representing the sampled value of the sampling variable  $r_i$  at the  $n$ -th step. The filtration  $\{\mathcal{H}_n^P\}_{n \in \mathbb{N}_0}$  is defined such that each  $\sigma$ -algebra  $\mathcal{H}_n^P$  is the smallest  $\sigma$ -algebra that makes all random variables in  $\{\theta_k^P\}_{0 \leq k \leq n}$  and  $\{\bar{\mathbf{x}}_k^P\}_{0 \leq k \leq n}$  measurable. We will omit the superscript  $P$  in all the notations above if it is clear from the context.

*Remark 1.* Under the condition that each sampling variable is bounded, using an inductive argument it follows that each  $\bar{\mathbf{x}}_n$  is a vector of bounded random variables. Thus  $\mathbb{E}^\sigma(|\bar{\mathbf{x}}_n[i]|)$  exists for each random variable  $\bar{\mathbf{x}}_n[i]$ .

Below we define the notion of *polynomial invariants* which logically captures all reachable configurations. A polynomial invariant may be obtained through abstract interpretation [16].

**Definition 3 (Polynomial Invariant).** A polynomial invariant (for  $P$ ) is a function  $I$  assigning a propositional polynomial predicate over  $X$  to every label in  $\mathcal{G}$  such that for all configurations  $(\ell, \mathbf{x})$  reachable from  $(\ell_0, \mathbf{x}_0)$  in  $\mathcal{G}$ , it holds that  $\mathbf{x} \models I(\ell)$ .

### 3 Termination over Probabilistic Programs

In this section, we first define the notions of almost-sure/finite termination and concentration bounds over probabilistic programs, and then describe the computational problems studied in this paper. Below we fix a probabilistic program  $P$  with its associated CFG  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  and an initial configuration  $(\ell_0, \mathbf{x}_0)$  for  $P$ .

**Definition 4 (Termination [8,22,13]).** A run  $\omega = \{(\ell_n, \mathbf{x}_n)\}_{n \in \mathbb{N}_0}$  over  $P$  is terminating if  $\ell_n = \perp$  for some  $n \in \mathbb{N}_0$ . The termination time of  $P$  is a random variable  $T_P$  such that for each run  $\omega = \{(\ell_n, \mathbf{x}_n)\}_{n \in \mathbb{N}_0}$ ,  $T_P(\omega)$  is the least number  $n$  such that  $\ell_n = \perp$  if such  $n$  exists, and  $\infty$  otherwise. The program  $P$  is said to be almost-sure terminating (resp. finitely terminating) if  $\mathbb{P}^\sigma(T_P < \infty) = 1$  (resp.  $\mathbb{E}^\sigma(T_P) < \infty$ ) for all schedulers  $\sigma$  (for  $P$ ).

Note that  $\mathbb{E}^\sigma(T_P) < \infty$  implies that  $\mathbb{P}^\sigma(T_P < \infty) = 1$ , but the converse does not necessarily hold (see [11, Example 5] for an example). i.e., finite-termination

implies almost-sure termination, but not vice-versa. To measure the expected values of the termination time under all (demonic) schedulers, we further define the quantity  $\text{ET}(P) := \sup_{\sigma} \mathbb{E}^{\sigma}(T_P)$ .

**Definition 5 (Concentration on Termination Time [38,13]).** A concentration bound for  $P$  is a non-negative integer  $M$  such that there exist real constants  $c_1 \geq 0$  and  $c_2 > 0$ , and for all  $N \geq M$  we have  $\mathbb{P}(T_P > N) \leq c_1 \cdot e^{-c_2 \cdot N}$ .

Informally, a concentration bound characterizes exponential decrease of probability values of non-termination beyond the bound. On one hand, it can be used to give an upper bound on probability of non-termination beyond a large step; and on the other hand, it leads to an algorithm that approximates  $\text{ET}(P)$  (cf. [13, Theorem 5]).

In this paper, we consider the algorithmic analysis of the following problems:

- **Input:** a probabilistic program  $P$ , a polynomial invariant  $I$  for  $P$  and an initial configuration  $(\ell_0, \mathbf{x}_0)$  for  $P$ ;
- **Output: (Almost-Sure/Finite Termination)** “yes” if the algorithm finds that  $P$  is almost-sure/finite terminating and “fail” otherwise;
- **Output: (Concentration on Termination)** a concentration bound if the algorithm finds one and “fail” otherwise.

## 4 Polynomial Ranking-Supermartingale

In this section, we develop the notion of polynomial ranking-supermartingale which is an extension of linear ranking-supermartingale [11,13]. We fix a probabilistic program  $P$ , a polynomial invariant  $I$  for  $P$  and an initial configuration  $(\ell_0, \mathbf{x}_0)$  for  $P$ . Let  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  be the associated CFG of  $P$ , with  $X = \{x_1, \dots, x_{|X|}\}$  and  $R = \{r_1, \dots, r_{|R|}\}$ . We first present the general notion of *ranking supermartingale*, and then define that of *polynomial ranking supermartingale*.

**Definition 6 (Ranking Supermartingale [22,13]).** A discrete-time stochastic process  $\{X_n\}_{n \in \mathbb{N}_0}$  w.r.t a filtration  $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$  is a ranking supermartingale (RSM) if there exist  $K < 0$  and  $\epsilon > 0$  such that for all  $n \in \mathbb{N}_0$ , we have  $\mathbb{E}(|X_n|) < \infty$  and it holds almost surely (with probability 1) that  $X_n \geq K$  and  $\mathbb{E}(X_{n+1} \mid \mathcal{F}_n) \leq X_n - \epsilon \cdot \mathbf{1}_{X_n \geq 0}$ , where  $\mathbb{E}(X_{n+1} \mid \mathcal{F}_n)$  is the conditional expectation of  $X_{n+1}$  given  $\mathcal{F}_n$  (cf. [51, Chapter 9]).

Informally, a polynomial ranking-supermartingale over  $P$  is a polynomial instantiation of an RSM through certain function  $\eta : (L \cup \{\perp\}) \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  which satisfies that each  $\eta(\ell, \cdot)$  (for all  $\ell \in L \cup \{\perp\}$ ) is essentially a polynomial function over  $X$ . Given such a function  $\eta$ , the intuition is to have conditions that

make the stochastic process  $X_n = \eta(\theta_n, \bar{\mathbf{x}}_n)$  an RSM. To ensure this, we consider the conditional expectation  $\mathbb{E}^\sigma(X_{n+1} \mid \mathcal{H}_n)$ ; this is captured by an extension of *pre-expectation* [11,13] from the linear to the polynomial case. Below we define  $L_\perp := L \cup \{\perp\}$ . For a function  $g : \mathbb{R}^{|X|} \times \mathbb{R}^{|R|} \rightarrow \mathbb{R}$ , we let  $\mathbb{E}_R(g, \cdot) : \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  be the function such that each  $\mathbb{E}_R(g, \mathbf{x})$  is the expected value  $\mathbb{E}(g(\mathbf{x}, \hat{\mathbf{r}}))$ , where  $\hat{\mathbf{r}}$  is any vector of independent random variables such that each  $\hat{r}[i]$  is a random variable that respects the cumulative distribution function  $\mathcal{T}_{r_i}$ .

**Definition 7 (Pre-Expectation).** *Let  $\eta : L_\perp \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  be a function such that each  $\eta(\ell, \cdot)$  (for all  $\ell \in L_\perp$ ) is a polynomial function over  $X$ . The function  $\text{pre}_\eta : L_\perp \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  is defined by:*

- $\text{pre}_\eta(\ell, \mathbf{x}) := \sum_{(\ell, z, \ell') \in \mapsto} z \cdot \eta(\ell', \mathbf{x})$  if  $\ell \in L_p$  (probabilistic transitions);
- $\text{pre}_\eta(\ell, \mathbf{x}) := \max_{(\ell, \star, \ell') \in \mapsto} \eta(\ell', \mathbf{x})$  if  $\ell \in L_d$  (nondeterministic transitions);
- $\text{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell', \mathbf{x})$  if  $\ell \in L_c$  and  $(\ell, \phi, \ell')$  is the only transition in  $\mapsto$  such that  $\mathbf{x} \models \phi$  (conditional transitions);
- $\text{pre}_\eta(\ell, \mathbf{x}) := \mathbb{E}_R(g, \mathbf{x})$  if  $\ell \in L_a$ , where  $g$  is the function such that  $g(\mathbf{x}, \mathbf{r}) = \eta(\ell', f(\mathbf{x}, \mathbf{r}))$  and  $(\ell, f, \ell')$  is the only transition in  $\mapsto$  (assignment transitions); and
- $\text{pre}_\eta(\ell, \mathbf{x}) := \eta(\ell, \mathbf{x})$  if  $\ell = \perp$  (terminal location).

The following lemma establishes the relationship between pre-expectation and conditional expectation whose proof is in Appendix E.

**Lemma 1.** *Let  $\eta : L_\perp \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  be a function such that each  $\eta(\ell, \cdot)$  (for all  $\ell \in L_\perp$ ) is a polynomial function over  $X$ , and  $\sigma$  be any scheduler. Let the stochastic process  $\{X_n\}_{n \in \mathbb{N}_0}$  be defined by:  $X_n := \eta(\theta_n, \bar{\mathbf{x}}_n)$ . Then for all  $n \in \mathbb{N}_0$ , we have  $\mathbb{E}^\sigma(X_{n+1} \mid \mathcal{H}_n) \leq \text{pre}_\eta(\theta_n, \bar{\mathbf{x}}_n)$ .*

*Example 3.* Consider the running example in Example 1 with CFG in Fig. 2. Let  $\eta$  be the function specified in the second and fifth column of Table 1, where  $g(x) := (x - 1)(10 - x)$ . Then  $\text{pre}_\eta$  is given in the third and sixth column of Table 1. Note that the case for  $i = 2$  is obtained from  $\text{pre}_\eta(2, x) = \max\{g(x) + 9.6, g(x) + 9.6\}$ , and the case for  $i = 3$  is from  $\text{pre}_\eta(3, x) = \mathbb{E}_R(h, x)$ , where  $h$  is the function  $h(y, r) = g(y) - (2y - 11)r - r^2 + 10$ .

$i$	$\eta(i, x)$	$\text{pre}_\eta(i, x)$	$i$	$\eta(i, x)$	$\text{pre}_\eta(i, x)$
1	$g(x) + 10$	$\mathbf{1}_{1 \leq x \leq 10} \cdot (g(x) + 9.8) + \mathbf{1}_{x < 1 \vee x > 10} \cdot (-0.2)$	5	$g(x) + 2x - 1.8$	$g(x) + 2x - 2$
2	$g(x) + 9.8$	$g(x) + 9.6$	6	$g(x) - 2x + 20.2$	$g(x) - 2x + 20$
3	$g(x) + 9.6$	$g(x) + 9$	7	$-0.2$	$-0.2$
4	$g(x) + 9.6$	$g(x) + 0.04x + 8.98$			

**Table 1.**  $\eta$  and  $\text{pre}_\eta$  for Example 1 and Fig. 2

We now define the notion of polynomial ranking-supermartingale. The intuition is that we encode the RSM-difference condition as a logical formula, treat zero as the threshold between terminal and non-terminal labels, and use the invariant  $I$  to over-approximate the set of reachable configurations at each label. Below for each  $\ell \in L_c$ , we define  $\text{PP}(\ell)$  to be the propositional polynomial predicate  $\bigvee_{(\ell, \phi, \ell') \in \mapsto, \ell' \neq \perp} \phi$ ; and for  $\ell \in L \setminus L_c$ , we let  $\text{PP}(\ell) := \text{true}$ .

**Definition 8 (Polynomial Ranking-Supermartingale).** *A  $d$ -degree polynomial ranking-supermartingale map (in short,  $d$ -pRSM) w.r.t  $(P, I)$  is a function  $\eta : L_{\perp} \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  satisfying that there exist  $\epsilon > 0$  and  $K \leq -\epsilon$  such that for all  $\ell \in L_{\perp}$  and all  $\mathbf{x} \in \mathbb{R}^{|X|}$ , the conditions (C1-C4) hold:*

- C1: the function  $\eta(\ell, \cdot) : \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  is a polynomial over  $X$  of order at most  $d$ ;
- C2: if  $\ell \neq \perp$  and  $\mathbf{x} \models I(\ell)$ , then  $\eta(\ell, \mathbf{x}) \geq 0$ ;
- C3: if  $\ell = \perp$ , then  $\eta(\ell, \mathbf{x}) = K$ ;
- C4: if  $\ell \neq \perp$  and  $\mathbf{x} \models I(\ell) \wedge \text{PP}(\ell)$ , then  $\text{pre}_{\eta}(\ell, \mathbf{x}) \leq \eta(\ell, \mathbf{x}) - \epsilon$ .

Note that C2 and C3 together separate non-termination and termination by the threshold 0, and C4 is the *RSM difference* condition which is intuitively related to the  $\epsilon$  difference in the RSM definition (cf. Definition 6). By generalizing our previous proofs in [13] (from LRSM to pRSM), we establish the soundness of pRSMs w.r.t both almost-sure and finite termination (proof in Appendix E).

**Theorem 1.** *If there exists a  $d$ -pRSM  $\eta$  w.r.t  $(P, I)$  with constants  $\epsilon, K$  (cf. Definition 8), then  $P$  is a.s. terminating and  $\text{ET}(P) \leq \text{UB}(P) := \frac{\eta(\ell_0, \mathbf{x}_0) - K}{\epsilon}$ .*

*Example 4.* Consider the running example (cf. Example 1 and Example 2) and the function  $\eta$  given in Example 3. Assuming that the initial valuation satisfies  $1 \leq x \wedge x \leq 10$ , we assign the trivial invariant  $I$  such that  $I(1) = 0 \leq x \wedge x \leq 11$ ,  $I(j) = 1 \leq x \wedge x \leq 10$  for  $2 \leq j \leq 6$  and  $I(7) = x < 1 \vee x > 10$ . It is straightforward to verify that  $\eta$  is a 2-pRSM with  $\epsilon = 0.2$  and  $K = -0.2$  (cf. Definition 8 for  $\epsilon, K$ ). Hence by Theorem 1, the program in Example 1 terminates almost-surely under any scheduler and its expected termination time is at most  $5 \cdot (x_0 - 1) \cdot (10 - x_0) + 51$ , given the initial value  $x_0$ .

*Remark 2.* The running example (cf. Example 1 and Example 2) does not admit a linear (i.e. 1-) pRSM since  $\mathbb{E}_R(r) = 0$  at label 3. This indicates that linear pRSMs may not exist even over simple affine programs like Example 1. Thus, this motivates the study of pRSMs even for simple affine programs.

*Remark 3.* The non-strict inequality symbol ' $\geq$ ' in C2 can be replaced by its strict counterpart ' $>$ ' since  $\eta + c$  ( $c > 0$ ) remains to be a pRSM if  $\eta$  is a pRSM and  $K$  (in C3) is sufficiently small. (By definition,  $\text{pre}_{\eta+c} = \text{pre}_{\eta} + c$ .) And the non-strict inequality symbol ' $\leq$ ' in C4 can be replaced by ' $<$ ' since a pRSM  $\eta$

and a constant  $K$  (for C3) can be scaled by a constant factor (e.g. 1.1) so that strict inequalities are ensured. Moreover, one can also assume that  $K = -1$  and  $\epsilon = 1$  in Definition 8. This is because one can first scale a pRSM with constants  $\epsilon, K$  by a positive scalar to ensure that  $\epsilon = 1$ , and then safely set  $K = -1$  due to C2.

Theorem 1 answers the questions of almost-sure and finite termination in a unified fashion. Generalizing our approach in [13], we show that by restricting a pRSM to have *bounded difference*, we also obtain concentration results.

**Definition 9 (Difference-Bounded pRSM).** *A  $d$ -pRSM  $\eta$  is difference-bounded w.r.t a non-empty interval  $[a, b] \subseteq \mathbb{R}$  if the following conditions hold:*

- for all  $\ell \in L_d \cup L_p$  and  $(\ell, \alpha, \ell') \in \mapsto$ , and for all  $\mathbf{x} \in \llbracket I(\ell) \rrbracket$ , it holds that  $a \leq \eta(\ell', \mathbf{x}) - \eta(\ell, \mathbf{x}) \leq b$ ;
- for all  $\ell \in L_c$  and  $(\ell, \phi, \ell') \in \mapsto$ , and for all  $\mathbf{x} \in \llbracket I(\ell) \wedge \phi \rrbracket$ , it holds that  $a \leq \eta(\ell', \mathbf{x}) - \eta(\ell, \mathbf{x}) \leq b$ ;
- for all  $\ell \in L_a$  and  $(\ell, f, \ell') \in \mapsto$ , for all  $\mathbf{x} \in \llbracket I(\ell) \rrbracket$  and for all  $\mathbf{r} \in \{\mathbf{r}' \mid \forall r \in R. \mathbf{r}'[r] \in \text{Supp}_r\}$ , it holds that  $a \leq \eta(\ell', f(\mathbf{x}, \mathbf{r})) - \eta(\ell, \mathbf{x}) \leq b$ .

Note that if a  $d$ -pRSM  $\eta$  with constants  $\epsilon, K$  (cf. Definition 8) is difference-bounded w.r.t  $[a, b]$ , then from definition  $a \leq -\epsilon$ ; one can further assume that  $-\epsilon \leq b$  since otherwise one can reset  $\epsilon := -b$ . By definition, the stochastic process  $X_n := \eta(\theta_n, \bar{\mathbf{x}}_n)$  defined through a difference-bounded pRSM w.r.t  $[a, b]$  satisfies that  $a \leq X_{n+1} - X_n \leq b$ ; then using Hoeffding’s Inequality [27,13], we establish a concentration bound.

**Theorem 2.** *Let  $\eta$  be a difference-bounded  $d$ -pRSM w.r.t  $[a, b]$  with constants  $\epsilon$  and  $K$ . For all  $n \in \mathbb{N}$ , if  $\epsilon(n-1) > \eta(\ell_0, \mathbf{x}_0)$ , then  $\mathbb{P}(T_P > n) \leq e^{-\frac{2(\epsilon(n-1) - \eta(\ell_0, \mathbf{x}_0))^2}{(n-1)(b-a)^2}}$ .*

From Theorem 2, a difference-bounded  $d$ -pRSM  $\eta$  implies a concentration bound of  $\frac{\eta(\ell_0, \mathbf{x}_0)}{\epsilon} + 2$  (detailed proof of the theorem is in Appendix F).

*Example 5.* Consider again our running example in Example 1 with invariant given in Example 4. Let  $\eta$  be the function illustrated in Table 1. One can verify that the interval  $[-10.2, 8.6]$  satisfies the conditions specified in Definition 9 for  $\eta$ , as the following hold:

- for all  $x \in [1, 10]$ ,  $\eta(2, x) - \eta(1, x) = -0.2$ ;
- for all  $x \in [0, 1) \cup (10, 11]$ ,  $-10.2 \leq \eta(7, x) - \eta(1, x) \leq -0.2$ ;
- for all  $x \in [1, 10]$  and  $i \in \{3, 4\}$ ,  $\eta(i, x) - \eta(2, x) = -0.2$ ;
- for all  $x \in [1, 10]$  and  $i \in \{5, 6\}$ ,  $-9.4 \leq \eta(i, x) - \eta(4, x) \leq 8.6$ ;
- for all  $x \in [1, 10]$ ,  $\eta(1, x-1) - \eta(5, x) = -0.2$ ;
- for all  $x \in [1, 10]$ ,  $\eta(1, x+1) - \eta(6, x) = -0.2$ ;

– for all  $x \in [1, 10]$  and  $r \in \{-1, 1\}$ ,

$$-9.6 \leq \eta(1, x+r) - \eta(3, x) (= -2rx - r^2 + 11r + 0.4) \leq 8.4 .$$

Then by Theorem 2, assuming that the program have initial value  $x_0 = 5$ , one can deduce that

$$\mathbb{P}(T_P > 50000) \leq e^{-\frac{2 \cdot (0.2 \cdot 49999 - 30)^2}{49999 \cdot 18.8^2}} \approx 1.3016 \cdot 10^{-5} .$$

We end this section with a result stating that whether a (difference-bounded)  $d$ -pRSM exists can be decided (proof in Appendix G). However, the complexity obtained for Theorem 3 is high since it involves quantifier elimination. In the next section, we will present efficient algorithms for synthesizing pRSMs.

**Theorem 3.** *For any fixed natural number  $d \in \mathbb{N}$ , the problem whether a (difference-bounded)  $d$ -pRSM w.r.t an input pair  $(P, I)$  exists is decidable.*

## 5 The Synthesis Algorithm

In this section, we present an efficient algorithmic approach for solving almost-sure/finite termination and concentration questions through synthesis of pRSMs. Instead of quantifier elimination (of Theorem 3) we use Positivstellensatz (German for “positive-locus-theorem” which is related to polynomials over semialgebraic sets), and the approach of this section is sound but not complete (in contrast to the computationally expensive but complete method of Theorem 3). Note that by Theorem 1, the existence of a pRSM implies both almost-sure and finite termination of a probabilistic program.

*The General Framework.* To synthesize a pRSM, the algorithm first sets up a polynomial template with unknown coefficients. Next, the algorithm finds values for the unknown coefficients,  $\epsilon, K$  (cf. Definition 8) and  $[a, b]$  (cf. Definition 9) so that C2-C4 in Definition 8 and concentration conditions in Definition 9 are satisfied. Note that from Definition 7, each  $\text{pre}_\eta(\ell, \cdot)$  is a polynomial over  $X$  whose coefficients are *linear combinations* of unknown coefficients from the polynomial template. Instead of using quantifier elimination (cf. e.g. [52] or Theorem 3), we utilize Positivstellensatz’s [45]. We observe that each universally-quantified formula described in C2, C4 and Definition 9 can be decomposed (through disjunctive normal form of propositional polynomial predicate or transformation of max in Definition 7 into two conjunctive clauses) into a conjunction of formulae of the following pattern (†)

$$\forall \mathbf{x} \in \mathbb{R}^{|X|}. [(g_1(\mathbf{x}) \geq 0 \wedge \dots \wedge g_m(\mathbf{x}) \geq 0) \rightarrow g(\mathbf{x}) > 0] \quad (\dagger)$$

where each  $g_i$  is a polynomial with constant coefficients and  $g$  is one with unknown coefficients from the polynomial template. In the pattern, we over-approximate any possible ‘ $g_j(\mathbf{x}) > 0$ ’ by ‘ $g_j(\mathbf{x}) \geq 0$ ’. By Remark 3, the difference between ‘ $g(\mathbf{x}) > 0$ ’ and ‘ $g(\mathbf{x}) \geq 0$ ’ does not matter.

*Example 6.* Consider again the program in Example 1 with CFG in Example 2. Consider the invariant specified in Example 4. The instances of the pattern for termination of this program are listed as follows, where each instance is represented by a pair  $(\Gamma, g)$  where  $\Gamma$  and  $g$  corresponds to  $\{g_1, \dots, g_m\}$  and resp.  $g$  described in  $(\dagger)$ .

- (C4, label 1)  $(\{x - 1, 10 - x, x, 11 - x\}, \eta(1, x) - \eta(2, x) - \epsilon)$ ;
- (C4, label 2)  $(\{x - 1, 10 - x\}, \eta(2, x) - \eta(3, x) - \epsilon)$  and  $(\{x - 1, 10 - x\}, \eta(2, x) - \eta(4, x) - \epsilon)$ ;
- (C4, label 3)  $(\{x - 1, 10 - x\}, \eta(3, x) - \mathbb{E}_R((y, r) \mapsto \eta(1, y + r), x) - \epsilon)$ ;
- (C4, label 4)  $(\{x - 1, 10 - x\}, \eta(4, x) - 0.51\eta(5, x) - 0.49\eta(6, x) - \epsilon)$ ;
- (C4, label 5)  $(\{x - 1, 10 - x\}, \eta(5, x) - \eta(1, x - 1) - \epsilon)$ ;
- (C4, label 6)  $(\{x - 1, 10 - x\}, \eta(6, x) - \eta(1, x + 1) - \epsilon)$ ;
- (C2)  $(\{x, 11 - x\}, \eta(1, x))$  and  $(\{x - 1, 10 - x\}, \eta(j, x))$  for  $2 \leq j \leq 6$ .

In the next part, we show that such pattern can be solved by Positivstellensatz's.

## 5.1 Positivstellensatz's

We fix a linearly-ordered finite set  $X$  of variables and a finite set  $\Gamma = \{g_1, \dots, g_m\} \subseteq \mathfrak{R}[X]$  of polynomials. Let  $[\Gamma]$  be the set of all vectors  $\mathbf{x} \in \mathbb{R}^{|X|}$  satisfying the propositional polynomial predicate  $\bigwedge_{i=1}^m g_i \geq 0$ . We first define pre-orderings and sums of squares as follows.

**Definition 10 (Sums of Squares).** Define  $\Theta$  to be the set of sums-of-squares, i.e.,

$$\Theta := \left\{ \sum_{i=1}^k h_i^2 \mid k \in \mathbb{N} \text{ and } h_1, \dots, h_k \in \mathfrak{R}[X] \right\} .$$

**Definition 11 (Preordering).** The preordering generated by  $\Gamma$  is defined by:

$$PO(\Gamma) := \left\{ \sum_{w \in \{0,1\}^m} h_w \cdot \prod_{i=1}^m g_i^{w_i} \mid \forall w. h_w \in \Theta \right\} .$$

*Remark 4.* It is well-known that a real-coefficient polynomial  $g$  of degree  $2d$  is a sum of squares iff there exists a  $k$ -dimensional positive semi-definite real square matrix  $Q$  such that  $g = \mathbf{y}^T Q \mathbf{y}$ , where  $k$  is the number of monomials of degree no greater than  $d$  and  $\mathbf{y}$  is the column vector of all such monomials (cf. [28, Corollary 7.2.9]). This implies that the problem whether a given polynomial (with real coefficients) is a sum of squares can be solved by semi-definite programming [25].

Now we present the first Positivstellensatz, called Schmüdgen's Positivstellensatz.

**Theorem 4 (Schmüdgen’s Positivstellensatz [46]).** *Let  $g \in \mathfrak{R}[X]$ . If the set  $\llbracket \Gamma \rrbracket$  is compact and  $g(\mathbf{x}) > 0$  for all  $\mathbf{x} \in \llbracket \Gamma \rrbracket$ , then  $g \in PO(\Gamma)$ .*

From Schmüdgen’s Positivstellensatz, any polynomial  $g$  which is positive on  $\llbracket \Gamma \rrbracket$  can be represented by

$$(\ddagger) \quad g = \sum_{w \in \{0,1\}^m} h_w \cdot g_w ,$$

where  $g_w := \prod_{i=1}^m g_i^{w_i}$  and  $h_w \in \Theta$  for each  $w \in \{0,1\}^m$ . To apply Schmüdgen’s Positivstellensatz, the degrees of those  $h_w$ ’s are restricted to be no greater than a fixed natural number. Then from Remark 4 and by equating the coefficients of the same monomials between the two polynomials, Eq.  $(\ddagger)$  results in a system of linear equalities that involves variables for synthesis of a pRSM and variables (grouped as  $2^m$  square matrices) under semi-definite constraints.

*Example 7.* Assume that  $X = \{x\}$  and  $\Gamma = \{1 - x, 1 + x\}$ . Choose the maximal degree for sums of squares to be 2. Then from Remark 4, the form of Eq.  $(\ddagger)$  can be written as:

$$g = \sum_{i=1}^4 \left[ (1 \ x) \cdot \begin{pmatrix} a_{i,1,1} & a_{i,1,2} \\ a_{i,2,1} & a_{i,2,2} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \right] \cdot u_i$$

where  $u_1 = 1$ ,  $u_2 = 1 - x$ ,  $u_3 = 1 + x$ ,  $u_4 = 1 - x^2$  and each matrix  $(a_{i,j,k})_{2 \times 2}$  ( $1 \leq i \leq 4$ ) is a matrix of variables subject to be positive semi-definite.

Theorem 4 can be further refined by a weaker version of Putinar’s Positivstellensatz.

**Theorem 5 (Putinar’s Positivstellensatz [42]).** *Let  $g \in \mathfrak{R}[X]$ . If (i) there exists some  $g_i \in \Gamma$  such that the set  $\{\mathbf{x} \in \mathbb{R}^{|X|} \mid g_i(\mathbf{x}) \geq 0\}$  is compact and (ii)  $g(\mathbf{x}) > 0$  for all  $\mathbf{x} \in \llbracket \Gamma \rrbracket$ , then*

$$(\S) \quad g = h_0 + \sum_{i=1}^m h_i \cdot g_i$$

for some sums of squares  $h_0, \dots, h_m \in \Theta$ .

Likewise, Eq.  $(\S)$  results in a system of linear equalities that involves variables for synthesis of a pRSM and matrices of variables under semi-definite constraints, provided that an upper bound on the degrees of sums of squares is enforced.

*Example 8.* Assume that  $X = \{x\}$  and  $\Gamma = \{1 - x^2, 0.5 - x\}$ . Choose the maximal degree for sums of squares to be 2. Then the form of Eq.  $(\S)$  can be written as:

$$g = \sum_{i=1}^3 \left[ (1 \ x) \cdot \begin{pmatrix} a_{i,1,1} & a_{i,1,2} \\ a_{i,2,1} & a_{i,2,2} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \right] \cdot u_i$$

where  $u_1 = 1$ ,  $u_2 = 1 - x^2$ ,  $u_3 = 0.5 - x$  and each matrix  $(a_{i,j,k})_{2 \times 2}$  ( $1 \leq i \leq 4$ ) is a matrix of variables subject to be positive semi-definite.

In the following, we introduce a Positivstellensatz entitled Handelman's Theorem when  $\Gamma$  consists of only linear (degree one) polynomials. For Handelman's Theorem, we assume that  $\Gamma$  consists of only linear (degree 1) polynomials and  $\llbracket \Gamma \rrbracket$  is non-empty. (Note that whether a system of linear inequalities has a solution is decidable in PTIME [47].)

**Definition 12 (Monoid).** *The monoid of  $\Gamma$  is defined by:*

$$\text{Monoid}(\Gamma) := \left\{ \prod_{i=1}^k h_i \mid k \in \mathbb{N}_0 \text{ and } h_1, \dots, h_k \in \Gamma \right\} .$$

**Theorem 6 (Handelman's Theorem [26]).** *Let  $g \in \mathfrak{R}[X]$  be a polynomial such that  $g(\mathbf{x}) > 0$  for all  $\mathbf{x} \in \llbracket \Gamma \rrbracket$ . If  $\llbracket \Gamma \rrbracket$  is compact, then*

$$(\#) \quad g = \sum_{i=1}^d a_i \cdot u_i$$

for some  $d \in \mathbb{N}$ , real numbers  $a_1, \dots, a_d \geq 0$  and  $u_1, \dots, u_d \in \text{Monoid}(\Gamma)$ .

To apply Handelman's theorem, we consider a natural number which serves as a bound on the number of multiplicands allowed to form an element in  $\text{Monoid}(\Gamma)$ ; then Eq. (#) results in a system of linear equalities involving  $a_1, \dots, a_d$ . Unlike previous Positivstellensatz's, the form of Handelman's theorem allows us to construct a system of linear equalities free from semi-definite constraints.

*Example 9.* Assume that  $X = \{x\}$  and  $\Gamma = \{1 - x, 1 + x\}$ . Fix the maximal number of multiplicands in an element of  $\text{Monoid}(\Gamma)$  to be 2. Then the form of Eq. (#) can be rewritten as

$$g = \sum_{i=1}^6 a_i \cdot u_i$$

where  $u_1 = 1$ ,  $u_2 = 1 - x$ ,  $u_3 = 1 + x$ ,  $u_4 = 1 - x^2$ ,  $u_5 = 1 - 2x + x^2$ ,  $u_6 = 1 + 2x + x^2$  and each  $a_i$  ( $1 \leq i \leq 6$ ) is subject to be a non-negative real number.

## 5.2 The Algorithm for pRSM Synthesis

Based on the Positivstellensatz's introduced in the previous part, we present our algorithm for synthesis of pRSMs. Below, we fix an input probabilistic program  $P$ , an input polynomial invariant  $I$  and an input initial configuration  $(\ell_0, \mathbf{x}_0)$  for  $P$ . Let  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  be the associated CFG of  $P$ .

*Description of the Algorithm PRSMSYNTH.* We present a succinct description of the key ideas. The description of the key steps of the algorithm is as follows.

1. *Template  $\eta$  for a pRSM.* The algorithm fix a natural number  $d$  as the maximal degree for a pRSM, construct  $\mathcal{M}_d$  as the set of all monomials over  $X$  of degree no greater than  $d$ , and set up a template  $d$ -pRSM  $\eta$  such that  $\eta(\ell, \cdot)$  is the polynomial  $\sum_{h \in \mathcal{M}_d} a_{h,\ell} \cdot h$  where each  $a_{h,\ell}$  is a (distinct) scalar variable (cf. C1).
2. *Bound for Sums of Squares and Monoid Multiplicands.* The algorithm fix a natural number  $k$  as the maximal degree for a sum of squares (cf. Schmüdgen’s and Putinar’s Positivstellensatz) or as the maximal number of multiplicands in a monoid element (cf. Handelman’s Theorem).
3. *RSM-Difference and Terminating-Negativity.* From Remark 3, the algorithm fixes  $\epsilon$  to be 1 (cf. condition C3) and  $K$  to be  $-1$  (cf. condition C4).
4. *Computation of pre-expectation  $\text{pre}_\eta$ .* With  $\epsilon, K$  fixed to be resp. 1,  $-1$  in the previous step, the algorithm computes  $\text{pre}_\eta$  by Definition 7, whose all involved coefficients are linear combinations from  $a_{h,\ell}$ ’s.
5. *Pattern Extraction.* The algorithm extracts instances conforming to pattern  $(\dagger)$  from C2, C4 and formulae presented in Definition 9, and translates them into systems of linear equalities over variables among  $a_{h,\ell}$ ’s,  $\epsilon$ ,  $K$ , and extra matrices of variables assumed to be positive semi-definite (cf. Schmüdgen’s and Putinar’s Positivstellensatz) or scalar variables assumed to be non-negative (cf. Handelman’s Theorem) through Eq.  $(\ddagger)$ , Eq.  $(\S)$  and Eq.  $(\#)$ .
6. *Solution via Semidefinite or Linear Programming.* The algorithm calls semi-definite programming (for Schmüdgen’s and Putinar’s Positivstellensatz) or linear programming (for Handelman’s Theorem) in order to check the feasibility or to optimize  $\text{UB}(P)$  (cf. Theorem 1 for upper bound of  $\text{ET}(P)$ ) over all variables among  $a_{h,\ell}$ ’s and extra matrix/scalar variables from Eq.  $(\ddagger)$ , Eq.  $(\S)$  and Eq.  $(\#)$ . Note that the feasibility implies the existence of a (difference-bounded)  $d$ -pRSM; the existence of a  $d$ -pRSM in turn implies finite termination, and the existence of a difference-bounded  $d$ -pRSM in turn implies a concentration bound through Theorem 2.

The soundness of our algorithm is as follows, whose proof is in Appendix H.

**Theorem 7 (Soundness).** *Any function  $\eta$  synthesized through the algorithm PRSMSYNTH is a valid pRSM.*

*Remark 5 (Efficiency).* It is well-known that for semi-definite programs with a positive real number  $R$  to bound the Frobenius norm of any feasible solution, an approximate solution upto precision  $\epsilon$  can be computed in polynomial time in the size of the semi-definite program (with rational numbers encoded in binary),  $\log R$  and  $\log \epsilon^{-1}$  [25]. Thus, our sound approach presents an efficient method for analysis of many probabilistic programs. Moreover, when each propositional polynomial predicate in the probabilistic program involves only linear polynomials, then the sound form of Handelman’s theorem can be applied, resulting in feasibility checking of systems of linear inequalities rather than semi-definite constraints. By polynomial-time algorithms for solving systems of linear

inequalities [47], our approach is polynomial time (and thus efficient) over such programs.

*Remark 6 (Semi-Completeness).* Consider probabilistic programs of the following form: **while**  $\phi$  **do if**  $\star$  **then**  $P_1$  **else**  $P_2$  **od**, where  $P_1, P_2$  are single assignments,  $[[\phi]]$  is compact, and invariants which assign to each label a propositional polynomial predicate is in DNF form that involves no strict inequality (i.e. no ‘<’ or ‘>’). Upon such inputs, our approach is *semi-complete* in the sense that by raising the upper bounds for the degree of a sum of squares and the number of multiplicands in a monoid element, the algorithm PRSMSYNTH will eventually find a pRSM if it exists. This is because Theorem 4 to 6 are “semi-complete” when  $[[I]]$  is compact, as the terminal label can be separately handled by  $\text{PP}(\cdot)$  so that only compact  $I$ ’s for Positivstellensatz’s may be formed, and the difference between strict and non-strict inequalities does not matter (cf. Remark 3).

*Remark 7 (Comparison with our previous result [13]).* Our approach using Handelman’s theorem is a strict generalization of the LRSM (linear ranking supermartingale) approach of [13] that uses Farkas’ lemma. For example, our approach using Handelman’s Theorem applied to affine programs can handle Example 1, where no LRSM exists (Remark 2).

*Remark 8 (New techniques for nonprobabilistic programs).* To the best of our knowledge, Schmüdgen’s Positivstellensatz and Handelman’s Theorem have not been used for nonprobabilistic programs, and thus our approach presents new analysis methods even for nonprobabilistic programs (though our approach is for the more general class of nondeterministic probabilistic programs).

*Remark 9 (Key Insights).* The key insights of this paper are (i) the need for pRSMs (cf. Remark 2), (ii) the adaptation of conditional expectation with pRSMs, (iii) the connection between synthesis of pRSMs and Positivstellensatz’s and (iv) the adoption of semidefinite and linear programming to synthesize pRSMs.

## 6 Experimental Results

In this section, we present experimental results for our algorithm through the semi-definite programming tool SOSTOOLS [3] (that uses SeDuMi [1]) and the linear programming tool CPLEX [2]. Due to space constraints, the detailed description of the input probabilistic programs are in Appendix I.

*Experimental examples and setup.* We consider six classical examples of probabilistic programs that exhibit distinct types non-linear behaviours. Our examples are, namely, *Logistic Map* adopted in [15] (Example 10 in Appendix I) which was previously handled by Lagrangian relaxation and semi-definite programming whereas our approach is polynomial time using linear programming,

*Decay* that models a sequence of points converging stochastically to the origin (Example 11 in Appendix I), *Random Walk* that models a random walk within a bounded region defined through non-linear curves (Example 12 in Appendix I), *Gambler’s Ruin* which is our running example (Example 1), *Gambler’s Ruin Variant* (Example 14 in Appendix I) which is a variant of (Example 13), and *Nested Loop* (Example 15 in Appendix I) which is a nested loop with stochastic increments. Except for *Gambler’s Ruin Variant* and *Nested Loop*, our approach is semi-complete for all other examples (cf. Remark 6). In all the examples the invariants are straightforward and was manually integrated with the input. Since SOSTOOLS only produces numerical results, we modify C2 to “ $\eta(\ell, \mathbf{x}) \geq 0$ ” to “ $\eta(\ell, \mathbf{x}) \geq 1$ ” for Putinar’s or Schmüdgen’s Positivstellensatz and check whether the maximal numerical error of all equalities added to SOSTOOLS is sufficiently small over a bounded region. In our examples, the bounded region is  $\{(x, y) \mid x^2 + y^2 \leq 2\}$  (cf. Example 12 and Example 11) and the maximal numerical error should not exceed 1. Note that 1 is also our fixed  $\epsilon$  in C4, and by Remark 3, the modification on C2 is not restrictive. Instead, one may also pursue Sylvester’s Criterion (cf. [28, Theorem 7.2.5]) to check membership of sums of squares through checking whether a square matrix is positive semi-definite or not. More elegant approaches for numerical problems is a subject of future work.

*Experimental results.* In Table 2, we present the experimental results, where ‘Method’ means that whether we use either Handelman’s Theorem, Putinar’s Positivstellensatz or Schmüdgen’s Positivstellensatz to synthesize pRSMs, ‘SOSTOOLS/CPLEX’ means the running time for CPLEX/SOSTOOLS in seconds, ‘error’ is the maximal numerical error of equality constraints added into SOSTOOLS (when instantiated with the solutions), and  $\eta(\ell_0, \cdot)$  is the polynomial for the initial label in the synthesized pRSM. The synthesized pRSMs (in the last column) refer to the variables of the program. All numbers except errors are rounded to  $10^{-4}$ . For all the examples, our translation to the optimization problems are linear. We report the running times of the optimization tools and synthesized pRSMs. The experimental results were obtained on Intel Core i7-2600 machine with 3.4 GHz with 16GB RAM.

Example	Method	SOSTOOLS	error	$\eta(\ell_0, \cdot)$
Decay	Putinar	0.1248s	$\leq 10^{-9}$	$5282.3435x^2 + 5282.3435y^2 + 1$
Random Walk	Schmüdgen	0.7176s	$\leq 10^{-4}$	$-300x^2 - 300y^2 + 601$
Example	Method	CPLEX	-	$\eta(\ell_0, \cdot)$
Gambler’s Ruin	Handelman	$\leq 10^{-2}$ s	-	$33x - 3x^2$
Gambler’s Ruin V.	Handelman	$\leq 10^{-2}$ s	-	$-21 + 100x - 70y - 100x^2 + 100xy$
Logistic Map	Handelman	$\leq 10^{-2}$ s	-	$1000500.7496x$
Nested Loop	Handelman	$\leq 2 \cdot 10^{-2}$ s	-	$48 + 160n + (m - x)(800n + 240)$

**Table 2.** Experimental Results

For all the examples we consider except Logistic Map, their almost-sure termination cannot be answered by previous approaches. For the Logistic-Map example, our reduction is to linear programming whereas existing approaches [15,48] reduce to semidefinite programming.

## 7 Conclusion and Future Work

In this paper, we extended linear ranking supermartingale (LRSM) for probabilistic programs proposed in [11,13] to polynomial ranking supermartingales (pRSM) for nondeterministic probabilistic programs. We developed the notion of (difference bounded) pRSM and proved that it is sound for almost-sure and finite termination, as well as for concentration bound (Theorem 1 and Theorem 2). Then we developed an efficient (sound but not complete) algorithm for synthesizing pRSMs through Positivstellensatz's (cf. Sect. 5.1), proved its soundness (Theorem 7) and argued its semi-completeness (Remark 6) over an important class of programs. Finally, our experiments demonstrate the effectiveness of our synthesis approach over various classical probabilistic programs, where LRSMs do not exist (cf. Example 1 and Remark 2). Directions of future work are to explore (a) more elegant methods for numerical problems related to semi-definite programming, and (b) other forms of RSMs for more general class of probabilistic programs.

**Acknowledgement** We thank anonymous referees for valuable comments. We also thank Hui Kong for his help on SOSTOOLS. The research was partly supported by Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE), ERC Start grant (279307: Graph Games), ERC Advanced Grant (267989: QUAREM), and Natural Science Foundation of China (NSFC) under Grant No. 61532019.

## References

1. SeDuMi 1.3. <http://sedumi.ie.lehigh.edu/> (2008)
2. IBM ILOG CPLEX Optimizer Interactive Optimizer Community Edition 12.6.3.0. <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/> (2010)
3. SOSTOOLS v3.00. <http://www.cds.caltech.edu/sostools/> (2013)
4. Babic, D., Cook, B., Hu, A.J., Rakamaric, Z.: Proving termination of nonlinear command sequences. *Formal Asp. Comput.* 25(3), 389–403 (2013)
5. Baier, C., Katoen, J.P.: *Principles of model checking*. MIT Press (2008)
6. Basu, S., Pollack, R., Roy, M.: *Algorithms in Real Algebraic Geometry*. Springer, 2nd edn. (2006)
7. Billingsley, P.: *Probability and Measure*. Wiley, 3rd edn. (1995)

8. Bournez, O., Garnier, F.: Proving positive almost-sure termination. In: Giesl, J. (ed.) Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3467, pp. 323–337. Springer (2005), [http://dx.doi.org/10.1007/978-3-540-32033-3\\_24](http://dx.doi.org/10.1007/978-3-540-32033-3_24)
9. Bradley, A.R., Manna, Z., Sipma, H.B.: Linear ranking with reachability. In: Etesami, K., Rajamani, S.K. (eds.) Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3576, pp. 491–504. Springer (2005)
10. Bradley, A.R., Manna, Z., Sipma, H.B.: Termination of polynomial programs. In: Cousot [17], pp. 113–129
11. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 511–526. Springer (2013)
12. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through positivstellensatz's. In: CAV (2016)
13. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In: Bodík, R., Majumdar, R. (eds.) Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016. pp. 327–342. ACM (2016), <http://doi.acm.org/10.1145/2837614.2837639>
14. Colón, M., Sipma, H.: Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2031, pp. 67–81. Springer (2001)
15. Cousot, P.: Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In: Cousot [17], pp. 1–24
16. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977. pp. 238–252. ACM (1977)
17. Cousot, R. (ed.): Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3385. Springer (2005)
18. Dubhashi, D., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press, New York, NY, USA, 1st edn. (2009)
19. Durrett, R.: Probability: Theory and Examples (Second Edition). Duxbury Press (1996)
20. Esparza, J., Gaiser, A., Kiefer, S.: Proving termination of probabilistic programs using patterns. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. Lecture Notes in Computer Science, vol. 7358, pp. 123–138. Springer (2012)

21. Farkas, J.: A fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikai és Természettudományi Értesítő* 12, 457–472 (1894)
22. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: Rajamani, S.K., Walker, D. (eds.) *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. pp. 489–501. ACM (2015)
23. Floyd, R.W.: Assigning meanings to programs. *Mathematical Aspects of Computer Science* 19, 19–33 (1967)
24. Foster, F.G.: On the stochastic matrices associated with certain queuing processes. *The Annals of Mathematical Statistics* 24(3), pp. 355–360 (1953)
25. Grötschel, M., Lovasz, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag Berlin Heidelberg (1993)
26. Handelman, D.: Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.* 132, 35–62 (1988)
27. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301), 13–30 (1963)
28. Horn, R.A., Johnson, C.R.: *Matrix Analysis*. Cambridge University Press, 2nd edn. (2013)
29. Howard, H.: *Dynamic Programming and Markov Processes*. MIT Press (1960)
30. Hungerford, T.W.: *Algebra*. Springer (1974)
31. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1), 99–134 (1998)
32. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4, 237–285 (1996)
33. Kemeny, J., Snell, J., Knapp, A.: *Denumerable Markov Chains*. D. Van Nostrand Company (1966)
34. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics* 25(6), 1370–1381 (2009)
35. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011*. *Proceedings. Lecture Notes in Computer Science*, vol. 6806, pp. 585–591. Springer (2011)
36. McIver, A., Morgan, C.: Developing and reasoning about probabilistic programs in *pGCL*. In: Cavalcanti, A., Sampaio, A., Woodcock, J. (eds.) *Refinement Techniques in Software Engineering, First Pernambuco Summer School on Software Engineering, PSSE 2004, Recife, Brazil, November 23-December 5, 2004, Revised Lectures*. *Lecture Notes in Computer Science*, vol. 3167, pp. 123–155. Springer (2004), [http://dx.doi.org/10.1007/11889229\\_4](http://dx.doi.org/10.1007/11889229_4)
37. McIver, A., Morgan, C.: *Abstraction, Refinement and Proof for Probabilistic Systems*. *Monographs in Computer Science*, Springer (2005)
38. Monniaux, D.: An abstract analysis of the probabilistic termination of programs. In: Cousot, P. (ed.) *Static Analysis, 8th International Symposium, SAS 2001, Paris, France, July 16-18, 2001, Proceedings*. *Lecture Notes in Computer Science*, vol. 2126, pp. 111–126. Springer (2001), [http://dx.doi.org/10.1007/3-540-47764-0\\_7](http://dx.doi.org/10.1007/3-540-47764-0_7)
39. Motwani, R., Raghavan, P.: *Randomized Algorithms*. Cambridge University Press, New York, NY, USA (1995)
40. Paz, A.: *Introduction to probabilistic automata (Computer science and applied mathematics)*. Academic Press (1971)

41. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2937, pp. 239–251. Springer (2004)
42. Putinar, M.: Positive polynomials on compact semi-algebraic sets. *Indiana Univ. Math. J.* 42, 969–984 (1993)
43. Rabin, M.: Probabilistic automata. *Information and Control* 6, 230–245 (1963)
44. Sankaranarayanan, S., Chakarav, A., Gulwani, S.: Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In: PLDI. pp. 447–458 (2013)
45. Scheiderer, C.: Positivity and sums of squares: A guide to recent results. *The IMA Volumes in Mathematics and its Applications* 149, 271–324 (2008)
46. Schmüdgen, K.: The  $K$ -moment problem for compact semi-algebraic sets. *Math. Ann.* 289, 203–206 (1991)
47. Schrijver, A.: *Theory of Linear and Integer Programming*. Wiley-Interscience series in discrete mathematics and optimization, Wiley (1999)
48. Shen, L., Wu, M., Yang, Z., Zeng, Z.: Generating exact nonlinear ranking functions by symbolic-numeric hybrid method. *J. Systems Science & Complexity* 26(2), 291–301 (2013)
49. Sohn, K., Gelder, A.V.: Termination detection in logic programs using argument sizes. In: Rosenkrantz, D.J. (ed.) Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado, USA. pp. 216–226. ACM Press (1991)
50. Tarski, A.: A decision method for elementary algebra and geometry. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pp. 24–84. Texts and Monographs in Symbolic Computation, Springer Vienna (1951)
51. Williams, D.: *Probability with Martingales*. Cambridge University Press (1991)
52. Yang, L., Zhou, C., Zhan, N., Xia, B.: Recent advances in program verification through computer algebra. *Frontiers of Computer Science in China* 4(1), 1–16 (2010), <http://dx.doi.org/10.1007/s11704-009-0074-7>

## A Propositional Polynomial Predicates

Formally, the set of propositional polynomial predicates over  $X$  is defined as the smallest set satisfying the following conditions:

1. each polynomial constraint over  $X$  is a propositional polynomial predicate;
2. both *true* and *false* are propositional polynomial predicates;
3. if  $\phi$  is a propositional polynomial predicate, then so is  $\neg\phi$ ;
4. if  $\phi, \psi$  are propositional polynomial predicates, then so are  $\phi \wedge \psi$  and  $\phi \vee \psi$ .

The satisfaction relation  $\models$  between real vectors  $\mathbf{x}$  and propositional polynomial predicates  $\phi$  is defined by:

- $\mathbf{x} \models \textit{true}$  and  $\mathbf{x} \not\models \textit{false}$  for all vectors  $\mathbf{x}$ ;
- $\mathbf{x} \models g_1 \bowtie g_2$  iff  $g_1(\mathbf{x}) \bowtie g_2(\mathbf{x})$  ;
- $\mathbf{x} \models \neg\phi$  iff  $\mathbf{x} \not\models \phi$  ;
- $\mathbf{x} \models \phi \wedge \psi$  iff  $\mathbf{x} \models \phi$  and  $\mathbf{x} \models \psi$  ;
- $\mathbf{x} \models \phi \vee \psi$  iff  $\mathbf{x} \models \phi$  or  $\mathbf{x} \models \psi$  .

## B Probabilistic Programs: Detailed Syntax

Let  $\mathcal{X}$  and  $\mathcal{R}$  be the disjoint countable collections of *program* and *sampling* variables, respectively. We assume that each sampling variable  $r$  be associated with a one-dimensional cumulative distribution function  $\Upsilon_r$  and a non-empty bounded interval  $\text{supp}_r$  in  $\mathbb{R}$  such that  $\Upsilon_r(\text{supp}_r) = 1$  and  $\Upsilon_r(\text{inf supp}_r) = 0$ , and the sampled values for  $r$  fall in  $\text{supp}_r$  with probability 1 (this is the rigorous condition of the boundedness of the sampling variables).

*The Syntax.* The syntax of probabilistic programs is given by the grammar in Figure 3. The expressions  $\langle pvar \rangle$ ,  $\langle rvar \rangle$  and  $\langle pvarlist \rangle$  range over  $\mathcal{X}$ ,  $\mathcal{R}$  and finite sequences of program variables, respectively. The expressions  $\langle expr \rangle$ ,  $\langle rexr \rangle$  and  $\langle rexrlist \rangle$  may be evaluated to any polynomial with variables in  $\mathcal{X}$ , any polynomial with variables in  $\mathcal{X} \cup \mathcal{R}$  and any finite list of polynomials with variables in  $\mathcal{X} \cup \mathcal{R}$ , respectively. The assignment statement  $\langle pvarlist \rangle := \langle rexrlist \rangle$  specifies simultaneous update of program variables in  $\langle pvarlist \rangle$  by polynomial  $\langle rexrlist \rangle$  in sequel; we thus assume that each instance of  $\langle pvarlist \rangle$  will contain distinct program variables and the length of each instance of  $\langle pvarlist \rangle$  will always be equal to the corresponding instance of  $\langle rexrlist \rangle$ . From the assignment statement one observes that sampling variables can only be used in the RHS of an assignment. Sequential composition, if-branch and while-loop are indicated by semicolon, the keyword **if** and the keyword **while**, respectively. Moreover,  $\langle bexr \rangle$  may be evaluated to any propositional polynomial predicate.

*The “if-then-else” Statement.* The guard  $\langle ndbexpr \rangle$  of each if-then-else statement is either a keyword  $\star$  representing demonic resolution of non-determinism, or a keyword **prob**( $p$ ) ( $p \in (0, 1)$ ) being a number given in decimal representation) representing the probabilistic choice that the if-branch is executed with probability  $p$  and the then-branch with probability  $1 - p$ , or a propositional polynomial predicate, in which case the statement represents a standard deterministic conditional branching.

$$\begin{aligned}
\langle stmt \rangle &::= \langle pvarlist \rangle \text{' := ' } \langle rexpriist \rangle \\
&| \text{' if ' } \langle ndbexpr \rangle \text{' then ' } \langle stmt \rangle \text{' else ' } \langle stmt \rangle \text{' fi ' } \\
&| \text{' while ' } \langle bexpr \rangle \text{' do ' } \langle stmt \rangle \text{' od ' } \\
&| \langle stmt \rangle \text{' ; ' } \langle stmt \rangle | \text{' skip ' } \\
\\
\langle expr \rangle &::= \langle constant \rangle | \langle pvar \rangle \\
&| \langle expr \rangle \text{' * ' } \langle expr \rangle \\
&| \langle expr \rangle \text{' + ' } \langle expr \rangle | \langle expr \rangle \text{' - ' } \langle expr \rangle \\
\\
\langle rexpri \rangle &::= \langle constant \rangle | \langle pvar \rangle | \langle rvar \rangle \\
&| \langle rexpri \rangle \text{' * ' } \langle rexpri \rangle \\
&| \langle rexpri \rangle \text{' + ' } \langle rexpri \rangle | \langle rexpri \rangle \text{' - ' } \langle rexpri \rangle \\
\\
\langle pvarlist \rangle &::= \langle pvar \rangle \text{' , ' } \langle pvarlist \rangle | \langle pvar \rangle \\
\langle rexpriist \rangle &::= \langle rexpri \rangle \text{' , ' } \langle rexpriist \rangle | \langle rexpri \rangle \\
\\
\langle literal \rangle &::= \langle expr \rangle \text{' \le ' } \langle expr \rangle | \langle expr \rangle \text{' \ge ' } \langle expr \rangle \\
\langle polyexpr \rangle &::= \langle literal \rangle | \langle literal \rangle \text{' and ' } \langle polyexpr \rangle \\
\langle bexpr \rangle &::= \langle polyexpr \rangle | \langle polyexpr \rangle \text{' or ' } \langle bexpr \rangle \\
\langle ndbexpr \rangle &::= \text{' \star ' } | \text{' prob ' } (p) \text{' ' } \langle bexpr \rangle
\end{aligned}$$

**Fig. 3.** Syntax of Probabilistic Programs

## C Transformation from Programs to CFGs

Below we fix a set  $X$  of program variables and a set  $R$  of sampling variables. We also fix two linear orders on  $X$  and  $R$  under which  $X = \{x_1, \dots, x_{|X|}\}$  and  $R = \{r_1, \dots, r_{|R|}\}$ .

We recall that a *valuation* of program variables is a vector  $\mathbf{x} \in \mathbb{R}^{|X|}$  interpreted in the way that the actual value held by a program variable  $x_i$  ( $1 \leq i \leq |X|$ ) is  $\mathbf{x}[i]$ ; similarly, a *valuation* of sampling variables is a vector  $\mathbf{r} \in \mathbb{R}^{|R|}$  such that the sampled value held by  $r_i$  is  $\mathbf{r}[i]$ . Every update function  $f$  in a CFG can then be viewed as a tuple  $(f_1, \dots, f_{|X|})$ , where each  $f_i$  is of type  $\mathbb{R}^{|X|} \times \mathbb{R}^{|R|} \rightarrow \mathbb{R}$ . We use the following succinct notation for special update functions: by *id* we denote the function which does not change the program variables at all, i.e. for every  $1 \leq i \leq |X|$  we have  $f_i(\mathbf{x}, \mathbf{r}) = \mathbf{x}[i]$ . For any  $k$  functions  $g_1, \dots, g_k : \mathbb{R}^{|X|} \times \mathbb{R}^{|R|} \rightarrow \mathbb{R}$  and any sequence  $n_1, \dots, n_k$  of  $k$  distinct numbers in  $\{1, \dots, |X|\}$ , we denote by  $[\{x_{n_j}\}_1^k / \{g_j\}_1^k]$  the update function  $f$  such that  $f_{n_j}(\mathbf{x}, \mathbf{r}) = g_j(\mathbf{x}, \mathbf{r})$  for  $1 \leq j \leq k$  and  $f_i(\mathbf{x}, \mathbf{r}) = \mathbf{x}[i]$  whenever  $i \notin \{n_1, \dots, n_k\}$ .

*From Probabilistic Programs to CFGs.* To every probabilistic program  $P$  with programs variables from  $X$  and sampling variables from  $R$ , we construct a CFG  $\mathcal{G}_P$  inductively on the structure of  $P$ . The CFG  $\mathcal{G}_P$  has  $X$  and resp.  $R$  as its set of program and resp. sampling variables. For each program  $P$ , the CFG  $\mathcal{G}_P$  involves two distinguished labels, namely  $\ell_P^{in}$  and  $\ell_P^{out}$ , that intuitively represent the label assigned to the first statement to be executed in  $P$  and the terminal label of  $P$ , respectively. The inductive construction is as follows.

1. *Assignments and Skips.* For  $P = x_{n_1}, \dots, x_{n_k} := E_1, \dots, E_k$  or resp.  $P = \mathbf{skip}$ , the CFG  $\mathcal{G}_P$  consists of a new assignment labels  $\ell_P^{in}$  and a new terminal label  $\ell_P^{out}$ , and a transition  $(\ell_P^{in}, [\{x_{n_j}\}_1^k / \{E_j\}_1^k], \ell_P^{out})$  or resp.  $(\ell_P^{in}, \text{id}, \ell_P^{out})$ , where we treat each  $E_j$  as a function through direct evaluation of variables.
2. *Sequential Statements.* For  $P = Q_1; Q_2$ , we take the disjoint union of the CFGs  $\mathcal{G}_{Q_1}, \mathcal{G}_{Q_2}$ , while redefining  $\ell_{Q_1}^{out}$  to be  $\ell_{Q_2}^{in}$  and putting  $\ell_P^{in} = \ell_{Q_1}^{in}$  and  $\ell_P^{out} = \ell_{Q_2}^{out}$ .
3. *While Statements.* For  $P = \mathbf{while} \phi \mathbf{do} Q \mathbf{od}$ , we add a new terminal label  $\ell_P^{out}$ , change  $\ell_Q^{out}$  to a branching label, add transitions  $(\ell_Q^{out}, \phi, \ell_Q^{in})$  and  $(\ell_Q^{out}, \neg\phi, \ell_P^{out})$ , and define  $\ell_P^{in} := \ell_Q^{in}$ .
4. *If Statements.* For  $P = \mathbf{if} B \mathbf{then} Q_1 \mathbf{else} Q_2 \mathbf{fi}$ , we consider different cases on  $B$ : if  $B$  is some  $\mathbf{prob}(p)$ , then we add a new probabilistic label  $\ell_P^{in}$  together with two transitions  $(\ell_P^{in}, p, \ell_{Q_1}^{in})$  and  $(\ell_P^{in}, 1 - p, \ell_{Q_2}^{in})$ ; if  $B$  is some propositional polynomial predicate  $\phi$  then we add a new branching label  $\ell_P^{in}$  together with transitions  $(\ell_P^{in}, \phi, \ell_{Q_1}^{in})$  and  $(\ell_P^{in}, \neg\phi, \ell_{Q_2}^{in})$ ; otherwise,  $B = \star$  and we add a new demonic label  $\ell_P^{in}$  together with transitions  $(\ell_P^{in}, \star, \ell_{Q_1}^{in})$  and  $(\ell_P^{in}, \star, \ell_{Q_2}^{in})$ . In any of the cases above, we also add a new terminal label  $\ell_P^{out}$  and identify both  $\ell_{Q_1}^{out}$  and  $\ell_{Q_2}^{out}$  with  $\ell_P^{out}$ .

## D The Semantics: Detailed Description

The behaviour of a probabilistic program  $P$  accompanied with its CFG  $\mathcal{G} = (L, \perp, (X, R), \mapsto)$  under a scheduler  $\sigma$  is described as follows. The program starts in the initial configuration  $(\ell_0, \mathbf{x}_0)$ . Then in each *step*  $i$  ( $i \in \mathbb{N}_0$ ), given the current

configuration  $(\ell_i, \mathbf{x}_i)$ , the next configuration  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is determined by the following procedure:

1. a valuation  $\mathbf{r}_i$  of the sampling variables is sampled according to the joint distribution of the cumulative distributions  $\{\mathcal{T}_r\}_{r \in R}$  and independent of all previously-traversed configurations (including  $(\ell_i, \mathbf{x}_i)$ ), all previous samplings on  $R$  and previous executions of probabilistic branches;
2. if  $\ell_i \in L_d$  and  $c_0, \dots, c_i$  is the finite path traversed so far (i.e.,  $c_0 = (\ell_0, \mathbf{x}_0)$  and  $c_i = (\ell_i, \mathbf{x}_i)$ ) with  $\sigma(c_0, \dots, c_i) = (\ell_i, \star, \ell')$ , then  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is set to be  $(\ell', \mathbf{x}_i)$ ;
3. if  $\ell_i \in L_p$  and  $(\ell_i, p, \ell_1), (\ell_i, 1-p, \ell_2)$  are namely the two transitions in  $\mapsto$  with source label  $\ell_i$ , then with a Bernoulli experiment independent of all previous samplings, probabilistic branches and traversed configurations,  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is set to be (i)  $(\ell_1, \mathbf{x}_i)$  with probability  $p$  and (ii)  $(\ell_2, \mathbf{x}_i)$  with probability  $1-p$ ;
4. if  $\ell_i \in L_c$  and  $(\ell_i, \phi, \ell_1), (\ell_i, \neg\phi, \ell_2)$  are namely the two transitions in  $\mapsto$  with source label  $\ell_i$ , then  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is set to be (i)  $(\ell_1, \mathbf{x}_i)$  when  $\mathbf{x}_i \models \phi$  and (ii)  $(\ell_2, \mathbf{x}_i)$  when  $\mathbf{x}_i \not\models \phi$ ;
5. if  $\ell_i \in L_a$  and  $(\ell_i, f, \ell')$  is the only transition in  $\mapsto$  with source location  $\ell_i$ , then  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is set to be  $(\ell', f(\mathbf{x}_i, \mathbf{r}_i))$ ;
6. if  $\ell_i = \perp$  then  $(\ell_{i+1}, \mathbf{x}_{i+1})$  is set to be  $(\ell_i, \mathbf{x}_i)$ .

## E Proof of Lemma 1 and Theorem 1

**Lemma 1.** Let  $\eta : L_\perp \times \mathbb{R}^{|X|} \rightarrow \mathbb{R}$  be a function such that each  $\eta(\ell, \cdot)$  (for all  $\ell \in L_\perp$ ) is a polynomial function over  $X$ , and  $\sigma$  be any scheduler. Let the stochastic process  $\{X_n\}_{n \in \mathbb{N}_0}$  be defined by:  $X_n := \eta(\theta_n, \bar{\mathbf{x}}_n)$ . Then for all  $n \in \mathbb{N}_0$ , we have  $\mathbb{E}^\sigma(X_{n+1} \mid \mathcal{H}_n) \leq \text{pre}_\eta(\theta_n, \bar{\mathbf{x}}_n)$ .

*Proof.* For all  $n \in \mathbb{N}_0$ , from the syntax and semantics of probabilistic program we have

$$X_{n+1} = \mathbf{1}_{\theta_n = \perp} \cdot X_n + Y_p + Y_d + Y_c + Y_a$$

where the terms are described below.

$$Y_p := \sum_{\ell \in L_p} \left[ \mathbf{1}_{\theta_n = \ell} \cdot \sum_{i \in \{0,1\}} (\mathbf{1}_{Z_\ell = i} \cdot \eta(\ell_{Z_\ell = i}, \bar{\mathbf{x}}_n)) \right]$$

where each random variable  $Z_\ell$  is the Bernoulli random variable for the decision of the probabilistic branch and  $\ell_{Z_\ell = 0}, \ell_{Z_\ell = 1}$  are the corresponding target labels from  $\ell$  in  $\mapsto$ . (Note that all  $Z_\ell$ 's are independent of  $\mathcal{H}_n$ .) In other words,  $Y_p$  describes the semantics of statements with probabilistic labels.

$$Y_a := \sum_{\ell \in L_a} [\mathbf{1}_{\theta_n = \ell} \cdot \eta(\ell', f_\ell(\bar{\mathbf{x}}_n, \bar{\mathbf{r}}_n))]$$

where  $(\ell, f_\ell, \ell')$  is the only transition in  $\mapsto$  with source label  $\ell$ , describing the semantics of statements with assignment labels.

$$Y_c := \sum_{\ell \in L_c} \sum_{(\ell, \phi, \ell') \in \mapsto} [\mathbf{1}_{\theta_n = \ell \wedge \bar{\mathbf{x}}_n \models \phi} \cdot \eta(\ell', \bar{\mathbf{x}}_n)]$$

which describes the semantics of statements with branching labels.

$$Y_d := \sum_{\ell \in L_d} \mathbf{1}_{\theta_n = \ell} \cdot \eta \left( \text{tgt} \left[ \sigma \left( \{(\theta_k, \bar{\mathbf{x}}_k)\}_{0 \leq k \leq n}\right) \right], \bar{\mathbf{x}}_n \right)$$

where  $\text{tgt} \left[ \sigma \left( \{(\theta_k, \bar{\mathbf{x}}_k)\}_{0 \leq k \leq n}\right) \right]$  is the target label of the transition  $\sigma \left( \{(\theta_k, \bar{\mathbf{x}}_k)\}_{0 \leq k \leq n}\right)$ , describing the semantics of demonic labels. Then from properties of conditional expectation [51, Page 88], one obtains:

$$\mathbb{E}^\sigma(X_{n+1} \mid \mathcal{H}_n) = \mathbf{1}_{\theta_n = \perp} \cdot X_n + Y'_p + Y'_a + Y_c + Y_d$$

(see below for details). This can be seen as follows. From the fact that  $\mathbf{1}_{\theta_n = \perp} \cdot X_n$ ,  $Y_d, Y_c$  are measurable in  $\mathcal{H}_n$ , we have  $\mathbb{E}^\sigma(\mathbf{1}_{\theta_n = \perp} \cdot X_n \mid \mathcal{H}_n) = \mathbf{1}_{\theta_n = \perp} \cdot X_n$  and similarly for  $Y_d, Y_c$ . For  $Y_p$  and  $Y_a$  we need their conditional expectation as  $Y'_p$  and  $Y'_a$  defined below:

$$Y'_p := \sum_{\ell \in L_p} \left[ \mathbf{1}_{\theta_n = \ell} \cdot \sum_{i \in \{0,1\}} (\mathbb{P}^\sigma(Z_\ell = i) \cdot \eta(\ell_{Z_\ell = i}, \bar{\mathbf{x}}_n)) \right]$$

and

$$Y'_a := \sum_{\ell \in L_a} [\mathbf{1}_{\theta_n = \ell} \cdot \mathbb{E}_R(g_\ell, \bar{\mathbf{x}}_n)]$$

where  $g_\ell$  equals the function  $(\mathbf{x}, \mathbf{r}) \mapsto \eta(\ell', f_\ell(\mathbf{x}, \mathbf{r}))$ . Note that the fact that  $\mathbb{E}_R(g_\ell, \bar{\mathbf{x}}_n)$  is well-defined is because we consider polynomial functions (i.e., pRSMs).

Note that the case for  $Y'_a$  is derived from the fact that each  $\eta(\ell', \cdot)$  is a polynomial over  $X$  and  $\bar{\mathbf{r}}_n$  is independent of  $\mathcal{H}_n$ . Now by definition,

$$\mathbf{1}_{\theta_n \notin L_d} \cdot \text{pre}_\eta(\theta_n, \bar{\mathbf{x}}_n) = \mathbf{1}_{\theta_n = \perp} \cdot X_n + Y'_p + Y'_a + Y_c$$

and

$$Y_d \leq \mathbf{1}_{\theta_n \in L_d} \cdot \text{pre}_\eta(\theta_n, \bar{\mathbf{x}}_n) .$$

Then the result follows.  $\square$

*Remark 10.* In the proof of the above result, which generalizes the existing proof from LRSM to pRSM, the crucial property of pRSM we use is for assignments (locations in  $L_a$ ) where we used the well-definedness of  $\mathbb{E}_R(g_\ell, \bar{\mathbf{x}}_n)$  due to polynomials. For more general RSMs if the well-definedness of  $\mathbb{E}_R(g_\ell, \bar{\mathbf{x}}_n)$  can be ensured then our proof ensures that the above result holds as well.

To prove Theorem 1, one also needs an important property which states that an RSM falls below zero almost surely.

**Proposition 1.** [22,13] *Let  $\{X_n\}_{n \in \mathbb{N}_0}$  be an RSM w.r.t a filtration  $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$  and constants  $K, \epsilon$  (cf. Definition 6). Let  $Z$  be the random variable defined by  $Z := \min\{n \in \mathbb{N}_0 \mid X_n < 0\}$  with  $\min \emptyset := \infty$ , denoting the first time  $n$  that the RSM drops below 0. Then  $\mathbb{P}(Z < \infty) = 1$  and  $\mathbb{E}(Z) \leq \frac{\mathbb{E}(X_0) - K}{\epsilon}$ .*

Now the proof for Theorem 1 is as follows.

**Theorem 1.** If there exists a  $d$ -pRSM  $\eta$  w.r.t  $(P, I)$  with constants  $\epsilon, K$  (cf. Definition 8), then  $P$  is a.s. terminating and  $\text{ET}(P) \leq \text{UB}(P) := \frac{\eta(\ell_0, \mathbf{x}_0) - K}{\epsilon}$ .

*Proof.* Let  $\eta$  be a  $d$ -pRSM and  $\{X_n\}_{n \in \mathbb{N}_0}$  be the stochastic process defined in Lemma 1. By Lemma 1, C4 and the fact that  $K \leq -\epsilon$ ,  $\{X_n\}_{n \in \mathbb{N}_0}$  is a ranking-supermartingale (w.r.t  $\{\mathcal{H}_n\}_{n \in \mathbb{N}}$ ). Then by C2, C3 and Proposition 1,

$$\text{ET}(P) = \sup_{\sigma} \mathbb{E}^{\sigma}(T_P) \leq \frac{\eta(\ell_0, \mathbf{x}_0) - K}{\epsilon} .$$

□

## F Proof of Theorem 2

To prove Theorem 2, we need the following concentration inequality.

**Theorem 8 (Hoeffding's Inequality [27,13]).** *Let  $\{X_n\}_{n \in \mathbb{N}}$  be a supermartingale w.r.t some filtration  $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$  and  $\{[a_n, b_n]\}_{n \in \mathbb{N}}$  be a sequence of intervals of positive length in  $\mathbb{R}$ . If  $X_1$  is a constant random variable and  $X_{n+1} - X_n \in [a_n, b_n]$  a.s. for all  $n \in \mathbb{N}$ , then*

$$\mathbb{P}(X_n - X_1 \geq \lambda) \leq e^{-\frac{2\lambda^2}{\sum_{k=2}^n (b_k - a_k)^2}}$$

for all  $n \in \mathbb{N}$  and  $\lambda > 0$ .

Now we fix a difference-bounded  $d$ -pRSM  $\eta$  w.r.t  $[a, b]$ . Recall that  $X_n := \eta(\theta_n, \bar{\mathbf{x}}_n)$ . Define the stochastic process  $\{Y_n\}_{n \in \mathbb{N}}$  by:

$$Y_n = X_n + \epsilon \cdot (\min\{T_P, n\} - 1) .$$

The following proposition shows that  $\{Y_n\}_{n \in \mathbb{N}}$  is a supermartingale and satisfies the requirements of Hoeffding's Inequality.

**Proposition 2.**  *$\{Y_n\}_{n \in \mathbb{N}}$  is a supermartingale and  $Y_{n+1} - Y_n \in [a + \epsilon, b + \epsilon]$  almost surely for all  $n \in \mathbb{N}$ .*

*Proof.* Consider the following random variable:

$$U_n = \min\{T_P, n + 1\} - \min\{T_P, n\} ,$$

and observe that this is equal to  $\mathbf{1}_{T_P > n}$ . From the properties of conditional expectation [51, Page 88] and the facts that (i) the event  $T_P > n$  is measurable in  $\mathcal{F}_n$  (which implies that  $\mathbb{E}(\mathbf{1}_{T_P > n} | \mathcal{F}_n) = \mathbf{1}_{T_P > n}$ ); and (ii)  $X_n \geq 0$  iff  $T_P > n$  (cf. conditions C2 and C3), we have

$$\begin{aligned} \mathbb{E}(Y_{n+1} | \mathcal{F}_n) - Y_n &= \mathbb{E}(X_{n+1} | \mathcal{F}_n) - X_n + \epsilon \cdot \mathbb{E}(U_n | \mathcal{F}_n) \\ &= \mathbb{E}(X_{n+1} | \mathcal{F}_n) - X_n + \epsilon \cdot \mathbb{E}(\mathbf{1}_{T_P > n} | \mathcal{F}_n) \\ &= \mathbb{E}(X_{n+1} | \mathcal{F}_n) - X_n + \epsilon \cdot \mathbf{1}_{T_P > n} \\ &\leq -\epsilon \cdot \mathbf{1}_{X_n \geq 0} + \epsilon \cdot \mathbf{1}_{T_P > n} \\ &= 0 . \end{aligned}$$

Note that the inequality above is due to the fact that  $X_n$  is a ranking supermartingale. Moreover, since  $T_P \leq n$  implies  $\theta_n = \ell_P^{out}$  and  $X_{n+1} = X_n$  we have that  $(X_{n+1} - X_n) = \mathbf{1}_{T_P > n} \cdot (X_{n+1} - X_n)$ . Hence we have

$$\begin{aligned} Y_{n+1} - Y_n &= X_{n+1} - X_n + \epsilon \cdot U_n \\ &= (X_{n+1} - X_n) + \epsilon \cdot \mathbf{1}_{T_P > n} \\ &= \mathbf{1}_{T_P > n} \cdot (X_{n+1} - X_n + \epsilon) . \end{aligned}$$

Hence  $Y_{n+1} - Y_n \in [a + \epsilon, b + \epsilon]$ . □

*Proof (of Theorem 2).* Let  $W_0 := Y_1 = \eta(\ell_0, \mathbf{x}_0)$ . Fix any demonic strategy  $\sigma$ . By Hoeffding's Inequality, for all  $\lambda > 0$ , we have  $\mathbb{P}(Y_n - W_0 \geq \lambda) \leq e^{-\frac{2\lambda^2}{(n-1)(b-a)^2}}$ . Note that  $T_P > n$  iff  $X_n \geq 0$  by conditions C2 and C3 of pRSM. Let  $\alpha = \epsilon(n-1) - W_0$  and  $\hat{\alpha} = \epsilon(\min\{n, T_P\} - 1) - W_0$ . Note that with the conjunct  $T_P > n$  we have that  $\alpha$  and  $\hat{\alpha}$  coincide. Thus, for  $\mathbb{P}(T_P > n) = \mathbb{P}(X_n \geq 0 \wedge T_P > n)$  we have

$$\begin{aligned} \mathbb{P}(X_n \geq 0 \wedge T_P > n) &= \mathbb{P}((X_n + \alpha \geq \alpha) \wedge (T_P > n)) \\ &= \mathbb{P}((X_n + \hat{\alpha} \geq \alpha) \wedge (T_P > n)) \\ &\leq \mathbb{P}((X_n + \hat{\alpha} \geq \alpha)) \\ &= \mathbb{P}(Y_n - Y_1 \geq \epsilon(n-1) - W_0) \\ &\leq e^{-\frac{2(\epsilon(n-1) - W_0)^2}{(n-1)(b-a)^2}} \end{aligned}$$

for all  $n > \frac{W_0}{\epsilon} + 1$ . The first equality is obtained by simply adding  $\alpha$  on both sides, and the second equality uses that because of the conjunct  $T_P > n$  we have  $\min\{n, T_P\} = n$  which ensures  $\alpha = \hat{\alpha}$ . The first inequality is obtained by simply dropping the conjunct  $T_P > n$ . The following equality is by definition, and the final inequality is an application of Hoeffding's Inequality. □

*Remark 11.* The above result holds for general difference-bounded RSMs and does not rely on the fact that it is a pRSM.

## G Proof for Theorem 3

**Theorem 3.** The problem whether a (difference-bounded)  $d$ -pRSM w.r.t  $(P, I)$  exists is decidable.

*Proof.* Let  $M$  be the set of all monomials of degree no greater than  $d$ . Let a template for a  $d$ -pRSM be  $\sum_{h \in M} a_h \cdot h$ , where  $a_h$  are scalar variables to be resolved. Then it is straightforward that conditions C1-C4 can be directly encoded as formulae in the first-order theory of reals which is first existentially quantified over the variables  $a_h, K, \epsilon$  and then universally quantified over the vector variable  $\mathbf{x}$ . The conditions for difference-bounded pRSMs can also be encoded as formulae which are firstly existentially quantified over the scalar variables  $a, b$  and then universally quantified over vector variable  $\mathbf{x}$ . Thus, the existence a (difference-bounded)  $d$ -pRSM is reduced to the validity of a formula in the first-order theory of reals, which is decidable [50,6].  $\square$

## H Proof of Theorem 7

**Theorem 7.** Any function  $\eta$  synthesized through the algorithm PRSMSYNTH is a valid pRSM.

*Proof.* To prove the soundness we observe that Steps 1-3 of the algorithm are basically instantiation of the template and obtaining the coefficients. Step 4 is the pre-expectation computation based on the definition. The crucial step is Step 5 and Step 6. The soundness of Step 5 and Step 6 follows from the soundness of Positivstellensatz’s (cf. Theorem 4 to 6) regardless of the compactness of  $\llbracket I \rrbracket$ : either Eq. (‡), Eq. (§) or Eq. (#) guarantees that formula (†) holds with ‘ $g(\mathbf{x}) > 0$ ’ replaced by ‘ $g(\mathbf{x}) \geq 0$ ’. It ensures that the synthesized pRSM is indeed a pRSM.  $\square$

## I Experimental Details

In the following description of the programs, we use “ $a \leq f \leq b$ ” for an abbreviation of “ $a \leq f \wedge f \leq b$ ”, and “ $(x_{n_1}, \dots, x_{n_k})^T := (E_{n_1}, \dots, E_{n_k})^T$ ” as a compact form for assignment “ $x_{n_1}, \dots, x_{n_k} := E_{n_1}, \dots, E_{n_k}$ ”. We also use UNIF( $a, b$ ) to denote the uniform distribution on  $[a, b]$ . Besides, the invariants are written in a bracketed fashion [...] and are put directly after the labels they are attached to. In all our examples the invariants are straightforward to obtain directly from the program.

*Example 10 (Logistic Map).* Consider the logistic-map example adopted in [15]. The program is depicted in Fig. 4.

```

[0 ≤ a ≤ 1 ∧ 0 ≤ x ≤ 1]
while 0 ≤ a ≤ 0.999 ∧ 0.001 ≤ x ≤ 1 do
  [0 ≤ a ≤ 0.999 ∧ 0.001 ≤ x ≤ 1]
  x := a * x * (1 - x)
od

```

**Fig. 4.** Logistic Map

*Example 11 (Decay).* Consider a decay example in Fig. 5 which is a discretized randomized version of the system of differential equations  $x' = -x + y, y' = -x - y$ ; the ODE describes the exponential decay of any initial value to the origin.

```

[x2 + y2 ≤ 2]
while 0.1 ≤ x2 + y2 ≤ 1 do
  [0.1 ≤ x2 + y2 ≤ 1]
   $\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} \text{UNIF}(0.98, 1) * x + 0.01 * y \\ \text{UNIF}(0.98, 1) * y - 0.01 * x \end{pmatrix}$ 
od

```

**Fig. 5.** Decay

*Example 12 (Random Walk).* Consider a demonic random-walk example in Fig. 6 which mimics a random walk within a bounded region; the region is defined through two non-linear parabola curves instead of linear constraints.

```

[x2 + y2 ≤ 2]
while x2 + y ≤ 1 ∧ x2 - y ≤ 1 do
  [x2 + y ≤ 1 ∧ x2 - y ≤ 1]
   $\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} x + \text{UNIF}(-0.1, 0.1) \\ y + \text{UNIF}(-0.1, 0.1) \end{pmatrix}$ 
od

```

**Fig. 6.** Random Walk

*Example 13 (Gambler's Ruin).* Finally, we consider the gambler's ruin in Example 1 with invariants given in Example 4.

*Example 14 (Gambler's Ruin Variant).* Consider a variant of Example 13 depicted in Fig. 7. Note that this example is another affine program that also does not admit a linear ranking supermartingale.

*Example 15 (Nested Loop).* Consider the example in Fig. 8. The example is a nested loop with two independent loop-control variables.

```

[0.7 ≤ x ≤ y + 0.3]
while 1 ≤ x ≤ y do
  [1 ≤ x ≤ y]
  if ★ do
    [1 ≤ x ≤ y]
    x := x + UNIF(-0.3, 0.3)
  else
    [1 ≤ x ≤ y]
    if prob(0.5) do
      [1 ≤ x ≤ y]
      x := x + 0.1
    else
      [1 ≤ x ≤ y]
      x := x - 0.1
    fi
  fi
od

```

**Fig. 7.** Gambler's Ruin Variant

```

[ $x \leq m + 0.2 \wedge n \geq 0$ ]
while  $x \leq m$  do
  [ $x \leq m \wedge n \geq 0$ ]
   $y := 0$ ;
  [ $x \leq m \wedge y \leq n + 0.2 \wedge n \geq 0$ ]
  while  $y \leq n$  do
    [ $x \leq m \wedge y \leq n \wedge n \geq 0$ ]
     $y := y + \text{UNIF}(-0.1, 0.2)$ 
  od;
  [ $x \leq m \wedge y \geq n \wedge n \geq 0$ ]
   $x := x + \text{UNIF}(-0.1, 0.2)$ 
od

```

**Fig. 8.** Nested Loop