Compositional Analysis for Almost-Sure Termination of Probabilistic Programs

Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, Amir K Goharshady

Compositional Analysis for Almost-Sure Termination of Probabilistic Programs

Mingzhang Huang, Hongfei Fu
Shanghai Jiao Tong University
mingzhanghuang@gmail.com, fuhf@cs.sjtu.edu.cn

Krishnendu Chatterjee, Amir Kafshdar Goharshady
IST Austria (Institute of Science and Technology Austria)
kchatterjee@ist.ac.at, goharshady@ist.ac.at

***Abstract*—In this work, we consider the almost-sure termination problem for probabilistic programs that asks whether a given probabilistic program terminates with probability 1. Scalable approaches for program analysis often rely on compositional analysis as their theoretical basis. In non-probabilistic programs, the classical variant rule (V-rule) of Floyd-Hoare logic is the foundation for compositional analysis. Extension of this rule to almost-sure termination of probabilistic programs is quite tricky, and a probabilistic variant was proposed in [15]. While the proposed probabilistic variant cautiously addresses the key issue of integrability, we show that the proposed compositional rule is still not sound for almost-sure termination of probabilistic programs.**

Besides establishing unsoundness of the previous rule, our contributions are as follows: First, we present a sound compositional rule for almost-sure termination of probabilistic programs. Our approach is based on a novel notion of descent supermartingales. Second, for algorithmic approaches, we consider descent supermartingales that are linear and show that they can be synthesized in polynomial time. Finally, we present experimental results on several natural examples that model various types of nested while loops in probabilistic programs and demonstrate that our approach is able to efficiently prove their almost-sure termination property.

I. Introduction

PROBABILISTIC PROGRAMS. Extending classical imperative programs with randomness, i.e. generation of random values according to probability distributions, gives rise to probabilistic programs [21]. Such programs provide a flexible framework for many different applications, ranging from the analysis of network protocols [17], [41], [26], to machine learning applications [37], [20], [40], [11], and robot planning [42], [43]. The recent interest in probabilistic programs has led to many probabilistic programming languages (such as Church [18], Anglican [44] and WebPPL [19]) and their analysis is an active research area in formal methods and programming languages (see [5], [45], [35], [1], [9], [7], [13], [28], [27]).

TERMINATION PROBLEMS. In program analysis the most basic liveness problem is that of *termination*, that given a program asks whether the program always terminates. In presence of probabilistic behavior, there are two natural extensions of the termination problem: first, the almost-sure termination problem that asks whether the program terminates with probability 1; and second, the finite-termination problem that asks whether the expected termination time is finite. While finite-termination implies almost-sure termination, the converse is not true. Both problems have been widely studied for probabilistic programs, e.g. [27], [7], [28], [9].

COMPOSITIONAL APPROACHES. Scalable approaches for program analysis are often based on compositional analysis as their theoretical foundation. For non-probabilistic programs, the classical variant rule (V-rule) of Floyd-Hoare logic [16], [29] provides the necessary foundations for compositional analysis. Such compositional methods allow decomposition of the programs into smaller parts, reasoning about the parts, and then combining the results on the parts to deduce the desired result for the entire program. Thus, they are the key technique in many automated methods for large programs.

COMPOSITIONAL APPROACHES FOR PROBABILISTIC PROGRAMS. The compositional approach for almost-sure termination of probabilistic programs was considered in [15]. First, it was shown that a direct extension of the V-rule of non-probabilistic programs is not sound for almost-sure termination of probabilistic programs, as there is a crucial issue regarding integrability. Then, a compositional rule, which cautiously addresses the integrability issue, was proposed as a sound rule for almost-sure termination of probabilistic programs. We refer to this rule as the FHV-rule.

OUR CONTRIBUTIONS. Our main contributions are as follows:

1) First, we show that the FHV-rule of [15], which is the natural extension of the V-rule with integrability condition, is not sound for almost-sure termination of probabilistic programs.
2) Second, we show that besides the issue of integrability, there is another crucial issue, regarding the non-negativity requirement in ranking supermartingales, that is not addressed by [15]. We present a sound compositional rule for almost-sure termination of probabilistic programs that addresses both crucial issues. Our approach is based on a novel notion called "descent supermartingales" (DSMs), which is an important technical contribution of our work.
3) Third, while we present our compositional approach for general DSMs, for algorithmic approaches we focus on DSMs that are linear. We present an efficient polynomial-time algorithm for the synthesis of linear DSMs.
4) Finally, we present an implementation of our synthesis algorithm for linear DSMs and demonstrate that our approach is applicable to probabilistic programs containing various types of nested while-loops and can efficiently prove that they terminate almost-surely.

II. Preliminaries

Throughout the paper, we denote by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of positive integers, nonnegative integers, integers, and real

numbers, respectively. We first review several useful concepts in probability theory and then present the syntax and semantics of our probabilistic programs.

A. Stochastic Processes and Martingales

We provide a short review of some necessary concepts in probability theory. For a more detailed treatment, see [46].

DISCRETE PROBABILITY DISTRIBUTIONS. A *discrete probability distribution* over a countable set U is a function $q : U \to [0,1]$ such that $\sum_{z \in U} q(z) = 1$. The *support* of q is defined as $\mathrm{supp}(q) := \{z \in U \mid q(z) > 0\}$.

PROBABILITY SPACES. A *probability space* is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a non-empty set (called *sample space*), $\mathcal{F}$ is a *σ-algebra* over Ω (i.e. a collection of subsets of Ω that contains the empty set $\emptyset$ and is closed under complementation and countable union) and $\mathbb{P}$ is a *probability measure* on $\mathcal{F}$, i.e. a function $\mathbb{P}\colon \mathcal{F} \to [0,1]$ such that (i) $\mathbb{P}(\Omega) = 1$ and (ii) for all set-sequences $A_1, A_2, \cdots \in \mathcal{F}$ that are pairwise-disjoint (i.e. $A_i \cap A_j = \emptyset$ whenever $i \neq j$) it holds that $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right)$. Elements of $\mathcal{F}$ are called *events*. An event $A \in \mathcal{F}$ holds *almost-surely* (a.s.) if $\mathbb{P}(A) = 1$.

RANDOM VARIABLES. A *random variable* X from a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $X\colon \Omega \to \mathbb{R} \cup \{-\infty, +\infty\}$, i.e. a function such that for all $d \in \mathbb{R} \cup \{-\infty, +\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) < d\}$ belongs to $\mathcal{F}$.

EXPECTATION. The *expected value* of a random variable X from a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, denoted by $\mathbb{E}(X)$, is defined as the Lebesgue integral of X w.r.t $\mathbb{P}$, i.e. $\mathbb{E}(X) := \int X \,\mathrm{d}\mathbb{P}$. The precise definition of Lebesgue integral is somewhat technical and is omitted here (cf. [46, Chapter 5] for a formal definition). If *range* $X = \{d_0, d_1, \ldots\}$ is countable, then we have $\mathbb{E}(X) = \sum_{k=0}^{\infty} d_k \cdot \mathbb{P}(X = d_k)$.

FILTRATIONS. A *filtration* of a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$ of σ-algebras over Ω such that $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$ for all $n \in \mathbb{N}_0$. Intuitively, a filtration models the information available at any given point of time.

CONDITIONAL EXPECTATION. Let X be any random variable from a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ such that $\mathbb{E}(|X|) < \infty$. Then, given any σ-algebra $\mathcal{G} \subseteq \mathcal{F}$, there exists a random variable (from $(\Omega, \mathcal{F}, \mathbb{P})$), denoted by $\mathbb{E}(X|\mathcal{G})$, such that
(E1) $\mathbb{E}(X|\mathcal{G})$ is $\mathcal{G}$-measurable, and
(E2) $\mathbb{E}\left(|\mathbb{E}(X|\mathcal{G})|\right) < \infty$, and
(E3) for all $A \in \mathcal{G}$, we have $\int_A \mathbb{E}(X|\mathcal{G}) \,\mathrm{d}\mathbb{P} = \int_A X \,\mathrm{d}\mathbb{P}$.
The random variable $\mathbb{E}(X|\mathcal{G})$ is called the *conditional expectation* of X given $\mathcal{G}$. The random variable $\mathbb{E}(X|\mathcal{G})$ is a.s. unique in the sense that if Y is another random variable satisfying (E1)–(E3), then $\mathbb{P}(Y = \mathbb{E}(X|\mathcal{G})) = 1$. We refer to [46, Chapter 9] for details. Intuitively, $\mathbb{E}(X|\mathcal{G})$ is the expectation of X, when assuming the information in G.

DISCRETE-TIME STOCHASTIC PROCESSES. A *discrete-time stochastic process* is a sequence $\Gamma = \{X_n\}_{n \in \mathbb{N}_0}$ of random variables where X_n's are all from some probability space $(\Omega, \mathcal{F}, \mathbb{P})$. The process Γ is *adapted to* a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$ if for all $n \in \mathbb{N}_0$, X_n is $\mathcal{F}_n$-measurable. Intuitively, the random variable X_i models some value at the i-th step of the process.

DIFFERENCE-BOUNDEDNESS. A discrete-time stochastic process $\Gamma = \{X_n\}_{n \in \mathbb{N}_0}$ adapted to a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$ is *difference-bounded* if there exists a $c \in (0, \infty)$ such that for all $n \in \mathbb{N}_0$, $|X_{n+1} - X_n| \leq c$ almost-surely.

SUPERMARTINGALES. A discrete-time stochastic process $\Gamma = \{X_n\}_{n \in \mathbb{N}_0}$ adapted to a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}_0}$ is a *supermartingale* if for every $n \in \mathbb{N}_0$, $\mathbb{E}(|X_n|) < \infty$ and it holds a.s. that $\mathbb{E}(X_{n+1}|\mathcal{F}_n) \leq X_n$. We refer to [46, Chapter 10] for a deeper treatment. Intuitively, a supermartingale is a discrete-time stochastic process in which for an observer who has seen the values of $X_0, \ldots, X_n$, the expected value at the next step, i.e. $\mathbb{E}(X_{n+1}|\mathcal{F}_n)$, is no more than the last observed value X_n.

B. Syntax

In the sequel, we fix two disjoint countable sets: the set of *program variables* and the set of *sampling variables*. Informally, program variables are directly related to the control flow of a program, while sampling variables represent random inputs sampled from distributions. We assume that every program variable is integer-valued, and every sampling variable is bound to a discrete probability distribution over integers. We first define several basic notions and then present the syntax.

VALUATIONS. A *valuation* over a finite set V of variables is a function $\nu : V \to \mathbb{Z}$ that assigns a value to each variable. The set of all valuations over V is denoted by Val_V.

ARITHMETIC EXPRESSIONS. An *arithmetic expression* $\mathfrak{e}$ over a finite set V of variables is an expression built from the variables in V, integer constants, and arithmetic operations such as addition, subtraction, multiplication, exponentiation, etc. For our theoretical results we consider a general setting for arithmetic expressions in which the set of allowed arithmetic operations can be chosen arbitrarily.

PROPOSITIONAL ARITHMETIC PREDICATES. A *propositional arithmetic predicate* over a finite set V of variables is a propositional formula ϕ built from (i) atomic formulae of the form $\mathfrak{e} \bowtie \mathfrak{e}'$ where $\mathfrak{e}, \mathfrak{e}'$ are arithmetic expressions and $\bowtie \in \{<, \leq, >, \geq\}$, and (ii) propositional connectives such as $\vee, \wedge, \neg$. The satisfaction relation $\models$ between a valuation ν and a propositional arithmetic predicate ϕ is defined through evaluation and standard semantics of propositional connectives, e.g. (i) $\nu \models \mathfrak{e} \bowtie \mathfrak{e}'$ iff $\mathfrak{e} \bowtie \mathfrak{e}'$ holds when the variables in $\mathfrak{e}, \mathfrak{e}'$ are substituted by their values in ν, (ii) $\nu \models \neg\phi$ iff $\nu \not\models \phi$ and (iii) $\nu \models \phi_1 \wedge \phi_2$ (resp. $\nu \models \phi_1 \vee \phi_2$) iff $\nu \models \phi_1$ and (resp. or) $\nu \models \phi_2$.

THE SYNTAX. Our syntax is illustrated by the grammar in Figure 1. Below, we explain the grammar.

- *Variables.* Expressions $\langle pvar \rangle$ (resp. $\langle rvar \rangle$) range over program (resp. sampling) variables.
- *Arithmetic Expressions.* Expressions $\langle expr \rangle$ (resp. $\langle pexpr \rangle$) range over arithmetic expressions over all program and sampling variables (resp. all program variables).
- *Boolean Expressions.* Expressions $\langle bexpr \rangle$ range over propositional arithmetic predicates over program variables.
- *Programs.* A program can either be a single assignment statement (indicated by ':='), or '**skip**' which is the

$$
\begin{array}{rcl}
\langle prog \rangle & ::= & \text{`}\textbf{skip}\text{'} \\
& & \mid \langle pvar \rangle \text{ `:='} \langle expr \rangle \\
& & \mid \langle prog \rangle \text{ `;'} \langle prog \rangle \\
& & \mid \text{`}\textbf{if}\text{'} \langle bexpr \rangle \text{ `}\textbf{then}\text{'} \langle prog \rangle \text{ `}\textbf{else}\text{'} \langle prog \rangle \text{ `}\textbf{fi}\text{'} \\
& & \mid \text{`}\textbf{if}\text{'} \text{ `}\star\text{'} \text{ `}\textbf{then}\text{'} \langle prog \rangle \text{ `}\textbf{else}\text{'} \langle prog \rangle \text{ `}\textbf{fi}\text{'} \\
& & \mid \text{`}\textbf{if}\text{'} \text{ `}\textbf{prob}(p)\text{'} \text{ `}\textbf{then}\text{'} \langle prog \rangle \text{ `}\textbf{else}\text{'} \langle prog \rangle \text{ `}\textbf{fi}\text{'} \\
& & \mid \text{`}\textbf{while}\text{'} \langle bexpr \rangle \text{ `}\textbf{do}\text{'} \langle prog \rangle \text{ `}\textbf{od}\text{'} \\
\\
\langle literal \rangle & ::= & \langle pexpr \rangle \text{ `}\leq\text{'} \langle pexpr \rangle \mid \langle pexpr \rangle \text{ `}\geq\text{'} \langle pexpr \rangle \\
\langle bexpr \rangle & ::= & \langle literal \rangle \mid \neg \langle bexpr \rangle \mid \langle bexpr \rangle \text{ `}\textbf{or}\text{'} \langle bexpr \rangle \\
& & \mid \langle bexpr \rangle \text{ `}\textbf{and}\text{'} \langle bexpr \rangle
\end{array}
$$

Fig. 1: The Syntax of Probabilistic Programs

```
1:  while x ≥ 1 do
2:      z := y;
3:      while z ≥ 0 do
4:          if x < 2 then
5:              x := x + r
            else
6:              skip
            fi;
7:          z := z − 1
        od;
8:      y := 4 × y;
9:      x := x − 1
    od
10:
```

Fig. 2: A Probabilistic Program and its Labels

special statement that does nothing, or a conditional branch (indicated by '**if** $\langle bexpr \rangle$'), or a non-deterministic branch (indicated by '**if** $\star$'), or a probabilistic branch (indicated by '**if prob**(p)', where $p \in [0,1]$ is the probability of executing the **then** branch and $1-p$ that of the **else** branch), or a while-loop (indicated by the keyword '**while**'), or a sequential composition of two sub-programs (indicated by semicolon).

PROGRAM COUNTERS. We assign a *program counter* to each assignment statement, skip, if branch and while-loop. Intuitively, the counter specifies the current point in the execution of a program. We also refer to program counters as *labels*.

C. Semantics

To specify the semantics of our probabilistic programs, we follow previous approaches, such as [5], [9], [7], and use Control Flow Graphs (CFGs) and Markov Decision Processes (MDPs) (see [2, Chapter 10]). Informally, a CFG describes how the program counter and valuations over program variables change along an execution of a program. Then, based on the CFG, one can construct an MDP as the semantical model of the probabilistic program.

Definition 1 (Control Flow Graphs)**.** A *Control Flow Graph* (CFG) is a tuple

$$\mathcal{G} = (L, (V_{\mathrm{p}}, V_{\mathrm{r}}), \to)$$

with the following components:

- L is a finite set of *labels*, which is partitioned into the set L_{b} of *conditional-branch* labels, the set L_{a} of *assignment* labels, the set L_{p} of *probabilistic* labels and the set L_{d} of *nondeterministic-branch* labels;
- V_{p} and V_{r} are disjoint finite sets of *program* and *sampling* variables, respectively;
- $\to$ is a *transition relation* in which every member (called a *transition*) is a tuple of the form (ℓ, α, ℓ') for which (i) ℓ (resp. ℓ') is the *source label* (resp. *target label*) in L and (ii) α is either a propositional arithmetic predicate if $\ell \in L_{\mathrm{b}}$, or an *update function* $u : \mathit{Val}_{V_{\mathrm{p}}} \times \mathit{Val}_{V_{\mathrm{r}}} \to \mathit{Val}_{V_{\mathrm{p}}}$ if $\ell \in L_{\mathrm{a}}$, or $p \in [0,1]$ if $\ell \in L_{\mathrm{p}}$ or $\star$ if $\ell \in L_{\mathrm{d}}$.

We always specify an *initial* label $\ell_{\mathrm{in}} \in L$ representing the starting point of the program, and a *terminal* label $\ell_{\mathrm{out}} \in L$ that represents termination and has no outgoing transitions.

INTUITION FOR CFGS. Informally, a control flow graph specifies how the program counter and values for program variables change in a program. We have three types of labels, namely *branching*, *assignment* and *nondeterministic*. The initial label ℓ_{in} corresponds to the initial statement of the program. A conditional branch label corresponds to a conditional-branching statement indicated by '**if** ϕ' or '**while** ϕ' , and leads to the next label determined by ϕ without changing the valuation. An assignment label corresponds to an assignment statement indicated by ':=' or **skip**, and leads to the next label right after the statement and an update to the value of the variable in the left-hand-side of ':=' that is specified by its right-hand side. This update can be seen as a function that gives the next valuation over program variables based on the current valuation and the sampled values. The statement '**skip**' is treated as an assignment statement that does not change values. A probabilistic branch label corresponds to a probabilistic-branching statement indicated by '**if prob**(p)', and leads to the label of '**then**' (resp. '**else**') branch with probability p (resp. $1-p$). A nondeterministic branch labels corresponds to nondeterministic choice statement indicated by '**if** $\star$', and has transitions to the two labels corresponding to the '**then**' and '**else**' branches.

By standard constructions, one can transform any probabilistic program into an equivalent CFG. We refer to [5], [9], [7] for details.

Example 1. Consider the probabilistic program in Figure 2. Its CFG is given in Figure 3. In this program, x, y and z are program variables, and r is a sampling variable that observes the probability distribution $\mathbb{P}(r=1) = \mathbb{P}(r=-1) = 0.5$. The numbers 1–10 are the program counters (labels). In particular, 1 is the initial label and 10 is the terminal label. The arcs represent transitions in the CFG. For example, the arc from 5 to 7 specifies the transition from label 5 to label 7 with the update function $x \mapsto x + r$ that assigns to program variable x, the value of the expression $x + r$, obtained by adding the value of x to a sampled value for the sampling variable r.

THE SEMANTICS. Based on CFGs, we define the semantics of probabilistic programs through the standard notion of Markov decision processes. Below, we fix a probabilistic program P

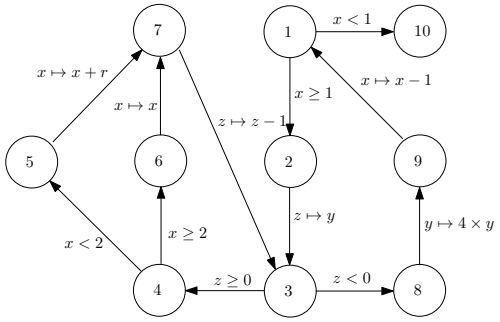

Fig. 3: The CFG of the Program in Figure 2

with its CFG in form (1). We define the notion of *configurations* such that a configuration is a pair (ℓ, ν), where ℓ is a label (that represents the current program counter) and $\nu \in \mathit{Val}_{V_\mathrm{p}}$ is a valuation (that represents the current valuation for program variables). We also fix a *sampling function* Υ which assigns to every sampling variable $r \in V_\mathrm{r}$, a discrete probability distribution over $\mathbb{Z}$. Then, the *joint* discrete probability distribution $\overline{\Upsilon}$ over $\mathit{Val}_{V_\mathrm{r}}$ is defined as $\overline{\Upsilon}(\mu) := \prod_{r \in V_\mathrm{r}} (\Upsilon(r))(\mu(r))$ for all valuations μ over sampling variables.

The semantics is described by a Markov decision process (MDP). Intuitively, the MDP models the stochastic transitions, i.e. how the current configuration jumps to the next configuration. The state space of the MDP is the set of all configurations. The actions are τ, **th** and **el** and correspond to the absence of nondeterminism, taking the **then**-branch of a nondeterministic-branch label, and taking the **else**-branch of a nondeterministic-branch label, respectively. The MDP transition probabilities are determined by the current configuration, the action chosen for the configuration and the statement at the current configuration.

To resolve nondeterminism in MDPs, we use schedulers. A *scheduler* σ is a function which maps every history, i.e. all information up to the current execution point, to a probability distribution over the actions available at the current state. Informally, it resolves nondeterminism at nondeterministic-branch labels by discrete probability distributions over actions that specify the probability of taking each action.

From the MDP semantics, the behaviour of a probabilistic program P with its CFG in the form (1) is described as follows: Consider an arbitrary scheduler σ. The program starts in an initial configuration (ℓ_0, ν_0) where $\ell_0 = \ell_\mathrm{in}$. Then in each step i ($i \geq 0$), given the current configuration (ℓ_i, ν_i), the next configuration (ℓ_{i+1}, ν_{i+1}) is determined as follows:

1) a valuation μ_i of the sampling variables is sampled according to the joint distribution $\overline{\Upsilon}$;
2) if $\ell_i \in L_\mathrm{a}$ and (ℓ_i, u, ℓ') is the transition in $\to$ with source label ℓ_i and update function u, then (ℓ_{i+1}, ν_{i+1}) is set to be $(\ell', u(\nu_i, \mu_i))$;
3) if $\ell_i \in L_\mathrm{b}$ and $(\ell_i, \phi, \ell'), (\ell_i, \neg\phi, \ell'')$ are the two transitions in $\to$ with source label ℓ_i, then (ℓ_{i+1}, ν_{i+1}) is set to be either (i) (ℓ', ν_i) if $\nu_i \models \phi$, or (ii) (ℓ'', ν_i) if $\nu_i \models \neg\phi$;
4) if $\ell_i \in L_\mathrm{d}$ and $(\ell_i, \star, \ell')$, $(\ell_i, \star, \ell'')$ are the two transitions in $\to$ with source label ℓ_i, then (ℓ_{i+1}, ν_{i+1}) is set to be (ℓ''', ν_i), where the label ℓ''' is chosen from ℓ', ℓ'' using the scheduler σ.
5) if $\ell_i \in L_\mathrm{p}$ and $(\ell_i, p, \ell'), (\ell_i, 1-p, \ell'')$ are the two transitions in $\to$ with source label ℓ_i, then (ℓ_{i+1}, ν_{i+1}) is set to be either (i) (ℓ', ν_i) with probability p, or (ii) (ℓ'', ν_i) with probability $1 - p$;
6) if there is no transition in $\to$ emitting from ℓ_i (i.e. if $\ell_i = \ell_\mathrm{out}$), then (ℓ_{i+1}, ν_{i+1}) is set to be (ℓ_i, ν_i).

For a detailed construction of the MDP, see Appendix A.

RUNS AND THE PROBABILITY SPACE. A *run* is an infinite sequence of configurations. Informally, a run $\{(\ell_n, \nu_n)\}_{n \in \mathbb{N}_0}$ specifies that the configuration at the n-th step of a program execution is (ℓ_n, ν_n), i.e. the program counter (resp. the valuation for program variables) at the n-th step is ℓ_n (resp. ν_n). By construction, with an initial configuration $\mathbf{c}$ (as the initial state of the MDP) and a scheduler σ, the Markov decision process for a probabilistic program induces a unique probability space over the runs (see [2, Chapter 10] for details). In the rest of the paper, we denote by $\mathbb{P}^\sigma_\mathbf{c}$ the probability measure under the initial configuration $\mathbf{c}$ and the scheduler σ, and by $\mathbb{E}^\sigma_\mathbf{c}(-)$ the corresponding expectation operator.

III. PROBLEM STATEMENT

In this section, we define the compositional verification problem of almost-sure termination over probabilistic programs. Below, we fix a probabilistic program P with its CFG in the form (1). We first define the notion of almost-sure termination. Informally, the property of almost-sure termination requires that a program terminates with probability 1. We follow the definitions in [5], [15], [9].

Definition 2 (Almost-sure Termination)**.** A run $\omega = \{(\ell_n, \nu_n)\}_{n \in \mathbb{N}_0}$ of a program P is *terminating* if $\ell_n = \ell_\mathrm{out}$ for some $n \in \mathbb{N}_0$. We define the *termination time* as a random variable T such that for a run $\omega = \{(\ell_n, \nu_n)\}_{n \in \mathbb{N}_0}$, $T(\omega)$ is the smallest n such that $\ell_n = \ell_\mathrm{out}$ if such an n exists (this case corresponds to program termination), and ∞ otherwise (this corresponds to non-termination). The program P is said to be *almost-surely (a.s.) terminating* under initial configurations $\mathbf{c}$ if $\mathbb{P}^\sigma_\mathbf{c}(T < \infty) = 1$ for all schedulers σ.

Lemma 1. Let the program P be the sequential (resp. conditional) composition of two other programs P_1 and P_2, i.e. $P := P_1; P_2$ (resp. $P :=$ **if** $-$ **then** P_1 **else** P_2 **fi**), and assume that both P_1 and P_2 are a.s. terminating for any initial value. Then, P is also a.s. terminating for any initial value. See Appendix B for a detailed proof.

Remark 1. *The lemma above shows that a.s. termination is closed under branching and sequential composition. Hence, in this paper, we only consider the major problem of compositional verification for a.s. termination of nested while loops.*

We now define the problem of compositional verification of a.s. termination.

Definition 3 (Compositional Properties)**.** We first describe the notion of compositionality in general. Consider a com-

positional operator op (e.g. sequential composition or loop nesting) over general objects. We say that a property ϕ is op-*compositional* under a *side condition* ψ if we have

$$(\psi(O, O') \wedge \phi(O) \wedge \phi(O')) \Rightarrow \phi(\mathsf{op}(O, O'))$$

holds for all objects O, O'. In other words, the property ϕ is op-*compositional* if the following assertion holds: for all objects O, O', if the side condition ψ and the property ϕ hold for O, O', then the property also holds on the (bigger) composed object $\mathsf{op}(O, O')$.

COMPOSITIONAL VERIFICATION. A compositional property ϕ can be proven by a natural divide-and-conquer approach: to prove ϕ for $\mathsf{op}(O, O')$, we first prove the same property on the (smaller) objects O and O' and then prove a side condition $\psi(O, O')$. Using compositional properties is an effective method for mitigating the state-space explosion problem usually arising in real-world verification problems.

THE ALMOST-SURE TERMINATION PROPERTY. In this paper, we are concerned with a.s. termination of while-loops. Our aim is to prove this based on the assumption that the loop body is a.s. terminating for *every* initial value. We consider $\textsc{Tm}(P)$ to be the target property expressing that the probabilistic program P is a.s. terminating for *every* initial value, and we consider the compositional operator to be the while-loop operator **while**, i.e. given a probabilistic program P and a propositional arithmetic predicate G (as the loop guard), the probabilistic program $\textbf{while}(G, P)$ is defined as **while** G **do** P **od**. Since P might itself be another while-loop, our setting encompasses probabilistic nested loops of any depth.

We focus on the compositional verification of $\textsc{Tm}(-)$ under the while-loop operator and solve the problem in the following steps: First, we establish a sufficient side condition ψ so that the assertion

$$(\psi(G, P) \wedge \textsc{Tm}(P)) \Rightarrow \textsc{Tm}(\textbf{while}(G, P)) \tag{1}$$

holds for all probabilistic programs P and propositional arithmetic predicates G[1]. Second, based on the proposed side conditions, we explore possible algorithmic approaches.

IV. PREVIOUS APPROACHES

In this section, we describe previous approaches for compositional verification of the (a.s.) termination property for (probabilistic) while-loops. We first present the variant rule from the Floyd-Hoare logic [16], [29] that is sound for non-probabilistic programs. Then we describe the probabilistic extension proposed in [15].

A. Classical Approach for Non-probabilistic Programs

Consider a non-probabilistic while-loop

$$P = \textbf{while } G \textbf{ do } P_1; \ldots; P_n \textbf{ od}$$

where the programs $P_1, \ldots, P_n$ may contain nested while-loops and are assumed to be terminating. The fundamental approach for compositional analysis is the following classical variant rule (V-rule) from the Floyd-Hoare logic [16], [29]:

$$\text{V-RULE}\ \frac{\begin{array}{c}\forall k : P_k \text{ terminates and } \{R = z\}P_k\{R \preceq z\} \\ \exists k\, \{R = z\}P_k\{R \prec z\}\end{array}}{\textbf{while } G \textbf{ do } P_1; \ldots; P_n \textbf{ od} \ \text{ terminates}}$$

In the V-rule above, R is an arithmetic expression over program variables that acts as a *ranking function*. The relation $\prec$ represents a well-founded relation when restricted to the loop guard G, while the relation $\preceq$ is the "non-strict" version of $\prec$ such that (i) $a \prec b \wedge b \preceq c \Rightarrow a \prec c$ and (ii) $a \preceq b \wedge b \prec c \Rightarrow a \prec c$. Then, the premise of the rule says that (i) for all P_k, the value of R after the execution of P_k does not increase in comparison with its initial value z before the execution, and (ii) there is some k such that the execution of P_k leads to a decrease in the value of R. If $\{R = z\}P_k\{R \preceq z\}$ holds, then P_k is said to be *unaffecting* for R. Similarly, if $\{R = z\}P_k\{R \prec z\}$ holds, then P_k is *ranking* for R. Informally, the variant rule says that if all P_k's are unaffecting and there is at least one P_k that is ranking, then P terminates.

The variant rule is sound for proving termination of non-probabilistic programs, because the value of R cannot be decremented infinitely many times, given that the relation $\prec$ is well-founded when restricted to the loop guard G.

B. A Previous Approach for Probabilistic Programs

Fioriti and Hermanns's approach in [15] can be viewed as an extension of the abstract V-rule, which is a proof system for a.s. terminating property. We call this abstract rule the FHV-rule:

$$\text{FHV-RULE}\ \frac{\begin{array}{c}\forall k : P_k \text{ terminates and } \{R = z\}P_k\{R \preceq z\} \\ \exists k\, \{R = z\}P_k\{R \prec z\}\end{array}}{\textbf{while } G \textbf{ do } P_1; \ldots; P_n \textbf{ od} \ \text{ terminates}}$$

Note that while the FHV-rule looks identical to the V-rule, semantics of the Hoare triple in the FHV-rule are different from that of the V-rule.

The FHV-rule is a direct probabilistic extension of the V-rule through the notion of *ranking supermartingales* (RSMs, see [5], [9], [7]). RSMs are discrete-time stochastic processes that satisfy the following conditions: (i) their values are always non-negative; and (ii) at each step of the process, the conditional expectation of the value is decreased by at least a positive constant ϵ. The decreasing and non-negative nature of RSMs ensures that with probability 1 and in finite expected number of steps, the value of any RSM hits zero. When embedded into programs through the notion of RSM-maps (see e.g. [5], [9]), RSMs serve as a sound approach for proving termination of probabilistic programs within finite expected time, which implies a.s. termination as well.

In [15], the R in the FHV-rule is a propositionally linear expression that represents an RSM, while $\prec$ is the well-founded relation on non-negative real numbers such that $x \prec y$ iff $x \leq y - \epsilon$ for some fixed positive constant ϵ and $\preceq$ is interpreted simply as $\leq$. Unaffecting and ranking conditions are extended to the probabilistic setting through conditional expectation (see $Dec_{\leq}(-,-), Dec_{<}(-,-)$ on [15, Page 9]).

[1]Note that we do not define or consider any assertion of the form $\textsc{Tm}(G)$, because checking the condition G always takes finite time.

Concretely, we say that (i) P_k is *unaffecting* if the expected value of R after the execution of P_k is no greater than its initial value before the execution; and (ii) P_k is *ranking* if the expected value of R after the execution of P_k is decreased by at least ϵ compared with its initial value before the execution. Note that in [15], R is also called a *compositional RSM*.

Crucial Issue 1 (Difference-boundedness and Integrability)**.** The authors of [15] accurately observed that simply extending the variant rule with expectation is not enough. They provided a counterexample in [15, Section 7.2] that is not a.s. terminating but has a compositional RSM. The problem is that random variables may not be integrable after the execution of a probabilistic while-loop. In order to resolve this integrability issue, they introduced the difference-boundedness condition (see Section II) for conditional expectation. Then, using the Optional Sampling/Stopping Theorem, they proved that, under the difference-boundedness condition, the random variables are integrable after the execution of while-loops. To ensure the difference-bounded condition, they established sound inference rules (see [15, Table 2 and Theorem 7.6]). With the integrability issue resolved, [15] finally claims that compositional ranking supermartingales provide a sound approach for proving a.s. termination of probabilistic while-loops (see [15, Theorem 7.7]).

V. A Counterexample to the FHV-rule

Although [15] takes care of the integrability issue, we show that, unfortunately, the FHV-rule is still not sound. We present an explicit counterexample on which the FHV-rule proves a.s. termination, while the program is actually not a.s. terminating.

Example 2 (The Counterexample)**.** Consider the probabilistic program in Figure 2 (Page 3). We will show that this program is not a.s. terminating. Recall that x, y, z are program variables and r is a sampling variable that observes $\mathbb{P}(r = 1) = \mathbb{P}(r = -1) = 0.5$. Intuitively, the program variable x models a random walk with one absorbing barrier at $x = 2$ in the inner loop, as indicated by the loop guard $x < 2$. By the structure of the outer loop, the program does not terminate only if, after a finite number of executions of the inner loop, the random walk stops at the absorbing barrier for every next iteration of the inner loop. Note that after each execution of the inner loop, the value of x does not increase in expectation. Furthermore, the value of x is decreased by one at the end of the loop body of the outer loop. Thus, the expected value of x decreases by 1 after each outer-loop iteration. As the outer loop guard is $x \geq 1$, this suggests that the program should be a.s. terminating. In contrast, we show that this program is not a.s. terminating (see Proposition 1 below).

Proposition 1. The probabilistic program in Example 2 (Figure 2, Page 3) is not a.s. terminating. Specifically, it does not terminate with probability 1 when the initial value for the program variable x is 1 and the initial value for the program variable y is sufficiently large.

Proof. The program does not terminate only if the value of x in label 9 is 2 after every execution of the inner loop. The key point is to prove that in the inner loop, the value of the program variable x will be 2 with higher and higher probability when the value of y increases. Consider the random walk in the inner loop. We abstract the values of x as three states '≤ 0','1' and '2'. From the structure of the program, we have that if we start with the state '1', then after the inner loop, the successor state may transit to either '≤ 0', '1' or '2'. If the successor state is either '≤ 0' or '1', then the outer-loop will terminate immediately. However, there is a positive probability that the successor state is '2' and the outer-loop does not terminate in this loop iteration (as the value of x will be set back to 1). This probability depends on the steps of the random walk in the inner loop (determined by the value of y), and we show that it is higher and higher when the value of y increases. Thus, after more and more executions of the outer loop, the value of y continues to increase exponentially, and with higher and higher probability the program would be not terminating in the current execution of the loop body.

The detailed demonstration is as follows: W.l.o.g, we assume that $x = 1$ at every beginning of the inner loop. The values of x at label 9 are the results of the execution of inner loop with the same initial value, hence they are independent mutually. We now temporarily fix the value $\hat{y}$ for y at the beginning of the outer-loop body and consider the probability that the value of x in label 9 is not 2. We use the random variable $\bar{X}_{\hat{y}}$ to describe the value of x at label 9 and analyze the situation $\bar{X}_{\hat{y}} \neq 2$ after the ($\hat{y}$ loop iterations of) the inner loop. Suppose that the $\hat{y}$ sampled values for r during the execution of the inner loop consist of m instances of -1 and $(\hat{y} - m)$ instances of 1. Since $\bar{X}_{\hat{y}} \neq 2$, we have $m \geq \frac{\hat{y}}{2}$. Then, there are $\binom{\hat{y}}{m} - \binom{\hat{y}}{m+1}$ different possible paths that avoid being absorbed by the barrier. The reason is that the only way to avoid absorption is to always have more -1's than 1's in any prefix of the path. Hence, the number of possible paths is the Catalan number. so we have $\mathbb{P}(\bar{X}_{\hat{y}} = 2) = 1 - \frac{1}{2^{\hat{y}}} \sum_{\frac{\hat{y}}{2} \leq m \leq \hat{y}} \left(\binom{\hat{y}}{m} - \binom{\hat{y}}{m+1}\right) = 1 - \frac{1}{2^{\hat{y}}} \binom{\hat{y}}{\lceil \frac{\hat{y}}{2} \rceil}$. Since $\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \leq n! \leq e n^{n+\frac{1}{2}} e^{-n}$ for $n \geq 1$ (applying Stirling's approximation), we have $1 - \frac{1}{2^{\hat{y}}} \binom{\hat{y}}{\lceil \frac{\hat{y}}{2} \rceil} = 1 - \frac{\hat{y}!}{2^{\hat{y}} (\frac{\hat{y}}{2}!)^2} \geq 1 - \frac{e \hat{y}^{\hat{y}+\frac{1}{2}} e^{-\hat{y}}}{2^{\hat{y}} (\sqrt{2\pi} \frac{\hat{y}}{2}^{\frac{\hat{y}}{2}+\frac{1}{2}} e^{-\frac{\hat{y}}{2}})^2} = 1 - \frac{e}{\pi \sqrt{\hat{y}}}$ for every even $\hat{y}$. Note that $\mathbb{P}(T = \infty) = \prod_{i \in \mathbb{N}_0} \mathbb{P}(\bar{X}_{\hat{y}_i} = 2)$, where $\hat{y}_i$ is the value of y at the i-th arrival to the label 9 and recall that $\hat{y}_0$ is sufficiently large. Furthermore, from the program we have $\hat{y}_i = 4^i \cdot \hat{y}_0$. Letting $d := \frac{e}{\pi \sqrt{\hat{y}_0}}$, we obtain that $\mathbb{P}(T = \infty) = \prod_{i \in \mathbb{N}_0} \mathbb{P}(\bar{X}_{\hat{y}_i} = 2) = \prod_{i \in \mathbb{N}_0} (1 - \frac{1}{2^{\hat{y}_i}} \binom{\hat{y}_i}{\lceil \frac{\hat{y}_i}{2} \rceil}) \geq \prod_{i \in \mathbb{N}_0} (1 - \frac{d}{\sqrt{4^i}})$. A well-known convergence criterion for infinite products is that $\prod_{i \in \mathbb{N}_0} (1 - q_n)$ converges to a non-zero number if and only if $\sum_{i \in \mathbb{N}_0} q_n$ converges for $0 \leq q_n < 1$. Since $\sum_{i \in \mathbb{N}_0} \frac{d}{2^i}$ converges, we have the infinite product $\prod_{i \in \mathbb{N}_0} (1 - \frac{d}{2^i})$ converges to a non-zero number. Thus, $\mathbb{P}(T = \infty) > 0$. □

We now show that, using the FHV-rule proposed in [15], one can deduce that the probabilistic program in Example 2 is a.s. terminating.

Proposition 2. The FHV-rule in [15] derives that the probabilistic program in Example 2 is a.s. terminating.

Proof. To see that the FHV-rule derives a.s. termination on this example, we show that the expression x is a compositional RSM that satisfies the integrability and difference-boundedness conditions. First, we can show that the program variable x is integrable and difference-bounded at every label. For example, for assignment statements at labels 2, 5, 7, 8 and 9 in Figure 2, the expression x is integrable and difference-bounded after these statements simply because either they do not involve x at the left-hand-side or the assignment changes the value of x by 1 unit. Similarly, within the nested loop, the loop body (from label 4 to label 7) causes bounded change to the value of x, so the expression x is integrable after the inner loop (using the while-rule in [15, Table 2]). Second, it is easy to see that the expression x is a compositional RSM as from [15, Definition 7.1] we have the following:

- The value of x does not increase after the assignment statements $z := y$ and $y := 4 \times y$;
- In the loop body of the nested loop, the expected value of x does not increase, given that it does not increase in any of the conditional branches;
- By definition of $Dec_{\leq}(-,-)$, the expected value of x does not increase after the inner loop;
- The value of x is decreased by 1 after the last assignment statement $x := x - 1$.

Thus, by applying [15]'s main theorem for compositionality ([15, Theorem 7.7]), we can conclude that the program should be a.s. terminating. □

From Proposition 1 and Proposition 2, we establish the main theorem of this section, i.e. that the FHV-rule is not sound. For a detailed explanation of the unsoundness of the FHV-rule, see Appendix C.

Theorem 1. The FHV-rule, i.e. the probabilistic extension of the V-rule as proposed in [15], is not sound for a.s. termination of probabilistic programs, even if we require the compositional RSM R to be difference-bounded and integrable.

Note that integrability is a very natural requirement in probability theory. Hence, Theorem 1 states that a natural probabilistic extension of the variant rule is not sufficient for proving a.s. termination of probabilistic programs.

VI. Our Compositional Approach

In the previous section, we showed that the FHV-rule is not sound for proving a.s. termination of probabilistic programs. In this section, we show how the FHV-rule can be strengthened to a sound approach.

Crucial Issue 2 (Non-negativity of RSMs)**.** The reason why the approach of [15] is not sound lies in the fact that ranking supermartingales (RSMs) are required to be non-negative stochastic processes (see e.g. [24, Example 3] for a counterexample, showing that the non-negativity condition is necessary). In the classical V-rule for non-probabilistic programs, non-negativity is not required, given that negative values in a non-probabilistic setting simply mean that R is negative. However, in the presence of probability, negative values only mean that the expected value of the expression R is negative. Thus, it is possible that the expected value of R decreases and becomes arbitrarily negative, tending to $-\infty$, while simultaneously the value of R increases with higher and higher probability. In our counterexample (Example 2), the expected value of x decreases after each outer-loop iteration, however the probability that the value of x remains the same increases with the value of y. More specifically, the expected-value decrease results from the fact that after the inner loop, the value of x may get arbitrarily negative towards $-\infty$.

The general idea of our approach is to require the expected value of the expression R in the variant rule to always decrease by at least a positive amount ϵ. We call this the *strict decrease* condition. This condition is in contrast with the FHV-rule that allows the value of R at certain statements to remain the same (in expectation). We show that after this strengthening, the resulting rule is sound for compositional verification of a.s. termination over probabilistic programs. Our main mathematical tools are the *concentration inequalities* (e.g. [31]) that give tight upper bounds on the probability that a stochastic process deviates from its mean value.

Instead of following an inference-rule-based method, we present our approach using martingales. This is because martingale-based approaches often lead to completely automated methods (e.g. [5], [9], [7]), while rule-based approaches mostly result in semi-automatic methods that require the use of interactive theorem provers (e.g. [28], [36], [34]). To clarify that our approach is indeed a strengthening of the FHV-rule in [15], we first write the rule-based approach of [15] in an equivalent martingale-based format.

Below, we fix a probabilistic program P' and a loop guard G and let $P := \textbf{while}(G, P')$. For the purpose of compositional verification, we assume that P' is a.s. terminating. We recall that T is the termination-time random variable (see Definition 2) and $\overline{\Upsilon}$ is the joint discrete probability distribution for sampling variables. We also use the standard notion of invariants, which are over-approximations of the set of reachable configurations at every label.

Invariants. An *invariant* is a function $I : L \to 2^{\mathit{Val}_{V_P}}$, such that for each label $\ell \in L$, the set $I(\ell)$ at least contains all valuations ν of program variables for which the configuration (ℓ, ν) can be visited in some run of the program. An invariant I is *linear* if every $I(\ell)$ is a finite union of polyhedra.

We can now describe the FHV-rule approach in [15] using *V-rule supermartingale maps*. A *V-rule supermartingale map* w.r.t an invariant I is a function $R : \mathit{Val}_{V_{\mathrm{p}}} \to \mathbb{R}$ satisfying the following conditions:

- *Non-increasing property.* The value of R does not increase in expectation after the execution of any of the statements in the outer-loop body. For example, the non-increasing condition for an assignment statement $\ell \in L_{\mathrm{a}}$ with $(\ell, u, \ell') \in \to$ (recall that u is the update function) is equivalent to $\sum_{\mu \in \mathit{Val}_{V_{\mathrm{r}}}} \overline{\Upsilon}(\mu) \cdot R(u(\nu, \mu)) \leq R(\nu)$ for all $\nu \in I(\ell)$. This condition can be similarly derived for other types of labels.
- *Decrease property.* There exists a statement that will definitely be executed in every loop iteration and will cause R to decrease (in expectation). For example, the condition for strict decrease at an assignment statement

$\ell \in L_{\rm a}$ with $(\ell, u, \ell') \in \to$ says that for all $\nu \in I(\ell)$ we have $\sum_{\mu \in \mathit{Val}_{V_{\rm r}}} \overline{\Upsilon}(\mu) \cdot R(u(\nu,\mu)) \leq R(\nu) - \epsilon$, where ϵ is a fixed positive constant.

- *Well-foundedness.* The values of R should be bounded from below when restricted to the loop guard. Formally, this condition requires that for a fixed constant c and all $\nu \in I(\ell)$ such that $\nu \models G$, we have $R(\nu) \geq c$.
- *Conditional difference-boundedness.* The conditional expected change in the value of R after the execution of each statement is bounded. For example, at an assignment statement $\ell \in L_{\rm a}$ with $(\ell, u, \ell') \in \to$, this condition says that there exists a fixed positive bound d, such that $\sum_{\mu \in \mathit{Val}_{V_{\rm r}}} \overline{\Upsilon}(\mu) \cdot |R(u(\nu,\mu)) - R(\nu)| \leq d$ for all $\nu \in I(\ell)$. The purpose of this condition is to ensure the integrability of R (see [15, Lemma 7.4]).

We strengthen the FHV-rule of [15] in two ways. First, as the major strengthening, we require that the expression R should strictly decrease in expectation at every statement, as opposed to [15] where the value of R is only required to decrease at some statement. Second, we slightly extend the conditional difference-boundedness condition and require that the difference caused in the value of R after the execution of each statement should always be bounded, i.e. we require difference-boundedness not only in expectation, but in every run of the program.

The core notion in our strengthened approach is that of *descent supermartingale maps (DSM-maps)*. A DSM-map is a function representing a decreasing amount (in expectation) at each step of the execution of the program.

Definition 4 (Descent Supermartingale Maps)**.** A *descent supermartingale map* (DSM-map) w.r.t real numbers $\epsilon > 0$, $c \in \mathbb{R}$, a non-empty interval $[a, b] \subseteq \mathbb{R}$ and an invariant I is a function $\eta : L \times \mathit{Val}_{V_{\rm p}} \to \mathbb{R}$ satisfying the following conditions:

(D1) For each $\ell \in L_{\rm a}$ with $(\ell, u, \ell') \in \to$, it holds that
- $a \leq \eta(\ell', u(\nu,\mu)) - \eta(\ell,\nu) \leq b$ for all $\nu \in I(\ell)$ and $\mu \in \mathit{Val}_{V_{\rm r}}$;
- $\sum_{\mu \in \mathit{Val}_{V_{\rm r}}} \overline{\Upsilon}(\mu) \cdot \eta(\ell', u(\nu,\mu)) \leq \eta(\ell,\nu) - \epsilon$ for all $\nu \in I(\ell)$;

(D2) For each $\ell \in L_{\rm b}$ and $(\ell, \phi, \ell') \in \to$, it holds that $a \leq \eta(\ell',\nu) - \eta(\ell,\nu) \leq \min\{-\epsilon, b\}$ for all $\nu \in I(\ell)$ such that $\nu \models \phi$;

(D3) For each $\ell \in L_{\rm d}$ and $(\ell, \star, \ell') \in \to$, it holds that $a \leq \eta(\ell',\nu) - \eta(\ell,\nu) \leq \min\{-\epsilon, b\}$ for all $\nu \in I(\ell)$;

(D4) For each $\ell \in L_{\rm p}$ with $(\ell, p, \ell'), (\ell, 1-p, \ell'') \in \to$, it holds that
- $a \leq \eta(\ell',\nu) - \eta(\ell,\nu) \leq b$ for all $\nu \in I(\ell)$,
- $a \leq \eta(\ell'',\nu) - \eta(\ell,\nu) \leq b$ for all $\nu \in I(\ell)$,
- $p \cdot \eta(\ell',\nu) + (1-p) \cdot \eta(\ell'',\nu) \leq \eta(\ell,\nu) - \epsilon$ for all $\nu \in I(\ell)$;

(D5) For all $\nu \in I(\ell_{\rm in})$ such that $\nu \models G$ (recall that G is the loop guard), it holds that $\eta(\ell_{\rm in}, \nu) \geq c$.

Informally, R is a DSM-map if:

(D1)–(D4) Its value decreases in expectation by at least ϵ after the execution of each statement (the strict decrease condition), and its change of value before and after each statement falls in $[a, b]$ (the strengthened difference-boundedness condition);

(D5) Its value is bounded from below by c at every entry into the loop body (the well-foundedness condition).

By the decreasing nature of DSM-maps, it is intuitively true that the existence of a DSM-map implies a.s. termination. However, this point is non-trivial as counterexamples will arise if we drop the difference-boundedness condition and only require the strict decrease condition (see e.g. [24, Example 3]). In the following, we use the difference-boundedness condition to derive a concentration property on the termination time (see [9]). Under this concentration property, we prove that DSM-maps are sound for proving a.s. termination.

We first present a well-known concentration inequality called *Hoeffding's Inequality*.

Theorem (Hoeffding's Inequality [23], [9])**.** Let $\{X_n\}_{n \in \mathbb{N}_0}$ 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 with positive length in $\mathbb{R}$. If X_0 is a constant random variable and $X_{n+1} - X_n \in [a_n, b_n]$ a.s. for all $n \in \mathbb{N}_0$, then

$$\mathbb{P}(X_n - X_0 \geq \lambda) \leq \exp\left(-\frac{2\lambda^2}{\sum_{k=1}^{n}(b_k - a_k)^2}\right)$$

for all $n \in \mathbb{N}_0$ and $\lambda > 0$.

Hoeffding's Inequality states that for any difference-bounded supermartingale, it is unlikely that its value X_n at the n-th step exceeds its initial value X_0 by much (measured by λ).

Using Hoeffding's Inequality, we prove the following lemma.

Lemma 2. Let $\{X_n\}_{n \in \mathbb{N}_0}$ be a supermartingale w.r.t some filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ and $[a, b]$ be an interval with positive length in $\mathbb{R}$. If X_0 is a constant random variable, it holds that $\mathbb{E}(X_{n+1} | \mathcal{F}_n) \leq X_n - \epsilon$ for some $\epsilon > 0$ and $X_{n+1} - X_n \in [a, b]$ a.s. for all $n \in \mathbb{N}_0$, then for any $\lambda \in \mathbb{R}$,

$$\mathbb{P}(X_n - X_0 \geq \lambda) \leq \exp\left(-\frac{2(\lambda + n \cdot \epsilon)^2}{n(b-a)^2}\right)$$

for all sufficiently large n.

Proof. Let $Y_n = X_n + n \cdot \epsilon$, then $a + \epsilon \leq Y_{n+1} - Y_n = X_{n+1} - X_n + \epsilon \leq b + \epsilon$. Given that

$$\begin{aligned} \mathbb{E}(Y_{n+1} | \mathcal{F}_n) &= \mathbb{E}(X_{n+1} | \mathcal{F}_n) + (n+1) \cdot \epsilon \\ &\leq X_n + n \cdot \epsilon \\ &= Y_n, \end{aligned}$$

we conclude that $\{Y_n\}_{n \in \mathbb{N}_0}$ is a supermartingale. Now we apply Hoeffding's Inequality for all n such that $\lambda + n \cdot \epsilon > 0$, and we get

$$\begin{aligned} \mathbb{P}(X_n - X_0 \geq \lambda) &= \mathbb{P}(Y_n - Y_0 \geq \lambda + n \cdot \epsilon) \\ &\leq \exp\left(-\frac{2(\lambda + n \cdot \epsilon)^2}{n(b-a)^2}\right) \end{aligned}$$

□

Thus, we have the following corollary by calculation.

Corollary 1. Let $\{X_n\}_{n\in\mathbb{N}_0}$ be a supermartingale satisfying the conditions of Lemma 2. Then, $\lim_{n\to+\infty}\sum_{k=n}^{+\infty}\mathbb{P}(X_k - X_0 \geq \lambda) = 0$.

We are now ready to prove the soundness of DSM-maps.

Theorem 2 (Soundness of DSM-maps)**.** If there exists a DSM-map η for P, then for any initial valuation $\nu^* \in \mathit{Val}_{V_\mathrm{p}}$ and for all schedulers σ, we have $\mathbb{P}^\sigma_{\nu^*}(T < \infty) = 1$.

Proof Sketch. Let ϵ, c, a, b be as defined in Definition 4. For a given program P with its DSM-map η, we define the stochastic process $\{X_n = \eta(\ell_n, \nu_n)\}_{n\in\mathbb{N}_0}$ where (ℓ_n, ν_n) is the pair of random variables that represents the configuration at the n-th step of a run. We also define the stochastic process $\{B_n\}_{n\in\mathbb{N}_0}$ in which each B_n represents the number of steps in the execution of P until the n-th arrival at the initial label ℓ_in. Then, X_{B_n} is the random variable representing the value of η at the n-th arrival at ℓ_in. Recall that, by condition (D5) in the definition of DSM-maps, the program stops if $X_{B_n} < c$. We now prove the crucial property that $\mathbb{P}(T' < \infty) \geq 1 - \lim_{n\to\infty}\mathbb{P}(X_{B_n} \geq c) = 1$, where T' is the random variable that measures the number of outer-loop iterations in a run. We want to estimate the probability of $\mathbb{P}(X_{B_n} \geq c)$ which is bounded by $\sum_{k=n}^{+\infty}\mathbb{P}(X_k \geq c)$. Note that X_n satisfies the conditions of Lemma 2. We use Corollary 1 to bound the probability. Since $\mathbb{P}(T < \infty) = 1$ iff $\mathbb{P}(T' < \infty) = 1$ (as P' is a.s. terminating), we obtain that $\mathbb{P}(T < \infty) = 1$. For a more detailed proof, see Appendix D. □

We illustrate an example application of Theorem 2.

Example 3. Consider the following probabilistic while-loop.

```
1 : while x ≥ 1 do
2 :   y := r;
3 :   while y ≥ 1 do
4 :     if ⋆ then
5 :       if prob (6/13) then
6 :         x := x + 1
          else
7 :         x := x − 1
          fi
        else
8 :       if prob (4/13) then
9 :         x := x + 2
          else
10 :        x := x − 1
          fi
        fi ;
11 :    y := y − 1
      od
    od
12 :
```

where the probability distribution for the sampling variable r is given by $\mathbb{P}(r = k) = 1/9$ for $k = 1, 2, \ldots, 9$.

The while-loop models a variant of gambler's ruin based on the mini-roulette game with 13 slots [8]. Initially, the gambler has x units of money and he continues betting until he has no money. At the start of each outer-loop iteration, the number of gambling rounds is chosen uniformly at random from $1, 2, \ldots, 9$ (i.e. the program variable y is the number of gambling rounds in this iteration). Then, at each round, the gambler takes one unit of money, and either chooses an *even-money bet* that bets the ball to stop at even numbers between 1 and 13, which has a probability of $\frac{6}{13}$ to win one unit of money (see the nondeterministic branch from label 5 to label 7), or a 2-*to*-1 *bet* that bets the ball to stop at 4 selected slots and wins two units of money with probability $\frac{4}{13}$ (see the branch from label 8 to label 10). During each outer-loop iteration, it is possible that the gambler runs out of money temporarily, but the gambler is allowed to continue gambling in the current loop iteration, and the program terminates only if he depletes his money when the program is back to the start of the outer-loop.

An invariant I for the program is as follows:

$$I(\ell) := \begin{cases} \textbf{true} & \text{if } \ell = 1 \\ x \geq 1 & \text{if } \ell = 2 \\ x \geq -8 \wedge 0 \leq y \leq 9 & \text{if } \ell = 3 \\ x \geq -7 \wedge 1 \leq y \leq 9 & \text{if } 4 \leq \ell \leq 11 \end{cases}.$$

For this program, we can define a DSM-map η as follows:

$$\eta(\ell, (x, y)) := \begin{cases} x & \text{if } \ell = 1 \\ x - 4/299 & \text{if } \ell = 2, 12 \\ x - 3/299 \cdot y + 7/299 & \text{if } \ell = 3 \\ x - 3/299 \cdot y + 3/299 & \text{if } \ell = 4 \\ x - 3/299 \cdot y - 1/299 & \text{if } \ell = 5, 8 \\ x - 3/299 \cdot y + 317/299 & \text{if } \ell = 6 \\ x - 3/299 \cdot y - 281/299 & \text{if } \ell = 7, 10 \\ x - 3/299 \cdot y + 616/299 & \text{if } \ell = 9 \\ x - 3/299 \cdot y + 14/299 & \text{if } \ell = 11 \end{cases}$$

where the 'x' (resp. 'y') represents the value for the program variable x (resp. y). One can verify that η is a DSM-map by choosing $\epsilon = 4/299, a = -280/299, b = 617/299$ and $c = 1$. The minimal and maximal one-step differences of η are met in the transitions from labels 9 and 10 to label 11. Thus, the differences are in the interval $[-1 + 19/299, 2 + 19/299] = [a, b]$, and the expected value of η decreases by at least $4/299 = \epsilon$ in each step. Also, if the outer loop is not stopped, then $x \geq 1 = c$ at the initial label. The other conditions can be similarly checked. Thus, η is a DSM-map for P. By applying Theorem 2, we conclude that the program terminates a.s. under any initial valuation.

We now compare the notion of DSM-maps with RSMs/RSM-maps [5], [9], [7] that have been successfully applied to prove finite expected termination time of probabilistic programs.

Remark 2 (Comparison with RSMs)**.** *Our notion of DSM-maps is slightly (but crucially) different from RSMs [5], [9], [7]. The difference is that a DSM-map does not require a global lower bound on its values, but instead requires the difference-boundedness condition, while an RSM requires its values to be non-negative, but has no difference-boundedness condition. As demonstrated in Theorem 2 and the previous section, this crucial difference leads to the fact that DSM-maps*

rather than RSMs serve as a sound approach for compositional verification of a.s. termination over probabilistic programs.

Remark 3 (Comparison with [15])**.** *We remark the reason why the approach in [15] is not sound while ours is. This has to do with Crucial Issue 2 (Page 7). The approach in [15] neglects the fact that RSMs have to be non-negative and is therefore not sound. In contrast, our approach uses DSM-maps which are not restricted to be non-negative and are sound for proving a.s. termination of probabilistic programs. As we have described previously, our approach of DSM-maps mainly strengthens the approach in [15] with the strict decrease condition at every statement. In our approach of DSM-maps, this situation is avoided through concentration inequalities, which guarantee that the probability of the value of R tending unboundedly to $-\infty$ is exponentially decreasing (see Lemma 2).*

Remark 4 (Real-valued Variables)**.** *Although we illustrate our approach on integer-valued variables, we show that it also works for real-valued variables. First, we directly extend the notion of DSM-maps to real-valued variables, where we only replace the discrete summation $\sum_{\mu\in \mathit{Val}_{V_r}} \overline{\Upsilon}(\mu)\cdot\eta(\ell', u(\nu,\mu))$ to an integral. Then we can prove the soundness of DSM-maps and construct the synthesis algorithm in the same way as for the integer case.*

PROOF SYSTEM FOR DSM. We construct a proof system $\mathcal{D}$, in the style of Hoare logic, for the DSM approach. Let R and R' be arithmetic expressions over program variables. The Hoare triple $\{R\}P\{R'\}$ indicates that for the program P, there exists a DSM η such that $\eta(\ell^P_{\text{in}}, \_) = R$ and $\eta(\ell^P_{\text{out}}, \_) = R'$. We have axioms for empty statement and assignments, rule of sequential composition, rule of conditional composition and the rule of while-loop composition in the proof system. We show the inference rule for while-loop composition here:

$$\text{DSM WHILE RULE}\ \frac{\begin{array}{c}\{R\}P\{R'\}, G \to R' \geq c,\\ G \to a \leq R - R' \leq -\epsilon,\\ \text{and } \neg G \to a \leq R'' - R' \leq -\epsilon\end{array}}{\{R'\}\ \textbf{while}\ G\ \textbf{do}\ P\ \textbf{od}\ \{R''\}}$$

Theorem 3. The proof system $\mathcal{D}$ is sound for a.s. termination of probabilistic programs.

Proof. We only consider the DSM while rule. Let $Q =$ **while** G **do** P **od** . Suppose that $\{R\}P\{R'\}$ and the DSM of P is η, then we construct a DSM η' by defining $\eta'(\ell^Q_{\text{in}}, \_) := R'$, $\eta'(\ell^Q_{\text{out}}, \_) := R''$ and $\eta'(\ell, \_) := \eta(\ell, \_)$ for all labels ℓ in the loop body. It is easy to check that η' is a valid DSM, and we have $\{R'\}Q\{R''\}$. By Theorem 2, we have the soundness of proof system $\mathcal{D}$. For the complete proof system and a more detailed proof, see Appendix E □

We argue that the approach of DSM-maps is a compositional approach for proving a.s. termination.

Remark 5 (Compositionality of the DSM approach)**.** *As shown in the proof above, in the DSM while rule, the existence of the expressions R, R' and R'' implies the existence of the DSM-map η'. Hence, DSM-maps can be used as side conditions of a compositional approach for proving a.s. termination, in the sense that the existence of the DSM-map η' serves as the ψ in (1). Notice that the proof system $\mathcal{D}$ alone cannot be used as a tool to prove the a.s. termination property, since the goal is the existence of the DSM-map rather than a concrete DSM-map.*

VII. ALGORITHMIC METHODS

In this section, we provide an algorithm for synthesizing linear DSM-maps. Recall that the existence of DSM-maps leads to the compositionality of a.s. termination over probabilistic while-loops. Since DSM-maps are similar to RSM-maps [5], [9], [7], we can directly extend previous algorithms for synthesizing linear/polynomial RSM-maps [5], [9], [7] to linear DSM-maps. The key mathematical tool used in our algorithm is the well-known Farkas' Lemma.

Theorem (Farkas' Lemma [14], [39])**.** Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ and $d \in \mathbb{R}$. Assume that $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$. Then

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

iff there exists $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{y} \geq \mathbf{0}$, $\mathbf{A}^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ and $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq d$.

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

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

where ξ is a variable representing a column vector of dimension m.

Below, we fix an input probabilistic while-loop P with a linear invariant I. We assume that P is *affine*, i.e. (i) every assignment statement in P has an affine expression at its right-hand side; and (ii) the loop guards of the conditional branches of P are in disjunctive normal form and each atomic proposition is a comparison between affine expressions.

The Synthesis Algorithm for DSM-maps. Our algorithm for synthesizing DSM-maps is consists of the following four steps:

1) *Template.* The algorithm establishes a template η for a DSM-map by setting $\eta(\ell, \mathbf{x}) := (\alpha_\ell)^{\mathrm{T}}\mathbf{x} + \beta_\ell$ for each $\ell \in L$ and $\mathbf{x} \in \mathbb{Z}^{|V_{\mathrm{P}}|}$, where α^ℓ is a vector of scalar variables and β^ℓ is a scalar variable, both representing unknown coefficients.
2) *Variables for Parameters in a DSM-map.* The algorithm sets up a scalar variable ϵ, two scalar variables a, b and a scalar variable c. These variables directly correspond to the parameters for a DSM-map (see Definition 4).
3) *Farkas' Linear Assertions.* From the template, we establish Farkas' linear assertions from the conditions (D1)–(D4). For example, the condition (D1) at a label ℓ requires that for the template η, it holds that $\sum_{\mu\in \mathit{Val}_{V_r}} \overline{\Upsilon}(\mu) \cdot \eta(\ell', u(\nu,\mu)) \leq \eta(\ell,\nu) - \epsilon$ for all $\nu \in I(\ell)$. Since the template η is linear and we have affine assignments, the

inequality $\sum_{\mu \in \mathit{Val}_{V_r}} \overline{\Upsilon}(\mu) \cdot \eta(\ell', u(\nu, \mu)) \leq \eta(\ell, \nu) - \epsilon$ would also be linear. Then (D1) is essentially an inclusion of the set $I(\ell)$ in the halfspace represented by $\sum_{\mu \in \mathit{Val}_{V_r}} \overline{\Upsilon}(\mu) \cdot \eta(\ell', u(\nu, \mu)) \leq \eta(\ell, \nu) - \epsilon$, and can be equivalently transformed into a group of Farkas' linear assertions, given that $I(\ell)$ is a finite union of polyhedra.

4) *Solution through Linear Programming.* We group the constructed Farkas' linear assertions together in a conjunctive manner so that we have a system of linear inequalities over scalar variables (including template variables, parameter variables and fresh variables from Farkas' linear assertions). Then, we solve for the variables through linear programming. If we can get a solution for the scalar variables, then we get a DSM-map that witnesses the a.s. termination of the input program; otherwise, the algorithm cannot prove the a.s. termination property and outputs "*fail*".

Theorem 4. Linear DSM-maps can be computed in polynomial time.

Proof. It is straightforward to check that Steps (1)–(3) of the Synthesis algorithm have polynomial runtime. Hence, the resulting LP, which should be solved in Step (4), has polynomial size. It is well-known that LPs can be solved in polynomial time. □

Example 4. We now illustrate an application of our synthesis algorithm on the program in Example 3.

- First, we set the template function $\eta(\ell, \mathbf{x}) = (\alpha_\ell)^{\mathrm{T}}\mathbf{x} + \beta_\ell$ for every label ℓ, where $\mathbf{x} = (x, y)^{\mathrm{T}}$ is the vector of program variables and the scalar variable β_ℓ together with the coordinate variables in the vector α_ℓ are unknown coefficients at a label ℓ.
- Second, we set up the parameters $\epsilon, a, b, c \in \mathbb{R}$ as in the definition of DSM-maps. The unknown coefficients in α^ℓ, β^ℓ and the parameters are what we want to solve for, in order to obtain a concrete DSM-map.
- In the third step, we establish Farkas' linear assertions. Below we illustrate an example on the construction of Farkas' linear assertions. Consider the condition (D4) at the label 5. The linear invariant at the label 5 is $x \geq -7 \wedge 1 \leq y \leq 9$ that represents the polyhedron $H = \left\{\mathbf{x} \mid \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}\mathbf{x} \leq \begin{pmatrix} 7 \\ 9 \\ -1 \end{pmatrix}\right\}$. To satisfy (D4), we have to ensure that the following conditions hold for every $\mathbf{x}$: $a \leq \eta(6, \mathbf{x}) - \eta(5, \mathbf{x}) \leq b$, $a \leq \eta(7, \mathbf{x}) - \eta(5, \mathbf{x}) \leq b$ and $\frac{6}{13}\eta(6, \mathbf{x}) + \frac{7}{13}\eta(7, \mathbf{x}) \leq \eta(5, \mathbf{x}) - \epsilon$. We first rewrite them into $\big(\alpha_5 - \alpha_6\big)^{\mathrm{T}}\mathbf{x} \leq -a + \beta_6 - \beta_5$, $(-\alpha_5 + \alpha_6)^{\mathrm{T}}\mathbf{x} \leq b - \beta_6 + \beta_5$ and $(\frac{6}{13}\alpha_6 + \frac{7}{13}\alpha_7 - \alpha_5)^{\mathrm{T}}\mathbf{x} \leq -\epsilon$. Let $d := -a + \beta_6 - \beta_5$, $d' := b - \beta_6 + \beta_5$. Then we construct the Farkas' linear assertions $\Phi[H, \alpha_5 - \alpha_6, d](\xi)$, $\Phi[H, -\alpha_5 + \alpha_6, d'](\xi')$ and $\Phi[H, \frac{6}{13}\alpha_6 + \frac{7}{13}\alpha_7 - \alpha_5, \epsilon](\xi'')$.
- Finally, in the fourth step we group all generated Farkas' linear assertions together in a conjunctive manner and solve for the unknown coefficients, together with the parameters and the fresh variables from Farkas' linear assertions, using an LP-solver. If we can get a solution for the unknown coefficients, then the algorithm confirms that the input program is a.s. terminating (Theorem 2). Otherwise, the algorithm outputs "*fail*". In this case, our algorithm is able to synthesize a linear DSM-map (see Section VIII).

VIII. Experimental Results

In this section, we present experimental results obtained by an implementation of our algorithm for synthesizing linear DSM-maps. Note that our algorithm has very few dependencies, all of which are standard operations (e.g. linear invariant generation and linear programming).

Experimental Benchmarks. We consider the program of Figure 2 (i.e. the counterexample to the FHV-rule), the Mini-roulette program of Example 3, and three other classical examples of probabilistic programs that exhibit various types of nested while-loops (Figure 4). *Program 1* is a simple nested while-loop, in which the outer loop control variable is updated in the inner loop. *Program 2* is a nested while-loop with two sequentially-composed inner loops, in which the outer loop control variables are each updated in one of these inner loops. *Program 3* is a three-level nested while-loop.

Invariants. Our approach is able to synthesize DSMs using very simple invariants obtained from the loop guards. See Appendix F for a complete list of invariants used in the experiments. Note that in all cases, the invariants we use are strictly weaker than, and can be replaced by, invariants generated by standard tools such as [12] and [38]. However, we use weaker invariants to demonstrate the power of our algorithm.

Distributions. In the experiments, we assume that each sampling variable r in Programs 1, 2 and 3 is sampled according to the distribution $\mathbb{P}(r = 1) = 0.25, \mathbb{P}(r = -1) = 0.75$. This choice is arbitrary and our approach can synthesize linear DSMs for any distribution, as long as such a DSM exists.

Experimental Results. Table I summarizes our experimental results over the five benchmark programs. Note that the counterexample program does not terminate almost surely. Therefore, any sound approach is expected to fail in obtaining a linear DSM map. In all other cases, our approach is extremely efficient and synthesizes a linear DSM map in less than half a second. In all cases, the DSM parameters ϵ and c are synthesized as 1 and 0, respectively. See Appendix G for details of the synthesized DSM maps.

Implementation and Experiment Machine. We implemented our approach in Java. The implementation is publicly available at https://ist.ac.at/~akafshda/DSM. We used lp-solve [3] and JavaILP [30] for solving the linear programming instances. The results were obtained on a Windows 10 machine with a 2.5 GHz Intel Core i5-2520M processor and 8 GB of RAM.

IX. Related Works

We compare our results with the most related works on termination verification of probabilistic programs. We discuss two main classes of approaches: supermartingale-based and proof-rule-based.

Program 1	Program 2	Program 3
1 : **while** $x \geq 1$ **do** 2 : $z := y$; 3 : **while** $z \geq 0$ **do** 4 : $z := z - 1$; 5 : $x := x + r$ **od**; 6 : $y := 2 \times y$; 7 : $x := x - 1$ **od** 8 :	1 : **while** $x + y \geq 1$ **do** 2 : $a := z$; 3 : $b := z$; 4 : **while** $a \geq 0$ **do** 5 : $a := a - 1$; 6 : $x := x + r_1$ **od**; 7 : **while** $b \geq 0$ **do** 8 : $b := b - 1$; 9 : $y := y + r_2$ **od**; 10 : $z := 2 \times z$; 11 : $x := x - 1$ **od** 12 :	1 : **while** $x \geq 1$ **do** 2 : $a := z$; 3 : **while** $a \geq 0$ **do** 4 : $a := a - 1$; 5 : $b := z$; 6 : **while** $b \geq 0$ **do** 7 : $b := b - 1$; 8 : $x := x + r$ **od**; 9 : $x := x + r$ **od**; 10 : $z := 2 \times z$; 11 : $x := x - 1$ **od** 12 :

Fig. 4: Our benchmark programs. These programs exhibit different types of nested while-loops.

Example	Result	Runtime (s)	$\eta(\ell_{\text{in}})$	$[a, b]$
Counterexample	Fail	0.305	–	–
Example 3	Success	0.338	$75.4 \cdot x$	$[-70.6, 155.6]$
Program 1	Success	0.234	$6 \cdot x + 5$	$[-4, 8]$
Program 2	Success	0.430	$7 \cdot x + 7 \cdot y + 6$	$[-5, 9.5]$
Program 3	Success	0.376	$8 \cdot x + 7$	$[-5, 11]$

TABLE I: Experimental Results

Supermartingale-based approaches. The most relevant works related to supermartingale-based approach include [6], [32], [33], [5], [4], [7], [9], [10], [34]. Compared to these results, the most significant difference is that our result considers compositional verification of the termination property, while most previous approaches tackle the termination problem directly on the whole program. First, we have compared our approach with the most relevant result [15], and shown that their approach is not sound. We have also presented the required strengthening and proven that our new approach is sound for compositionally proving a.s. termination of probabilistic programs. Second, another related result in [1] considers lexicographic RSMs that are sound for a.s. termination of probabilistic programs. While lexicographic RSMs have some flavor of compositionality (such as decomposing based on lexicographic order), they do not have the compositional property as in Definition 3.

Proof-rule-based approaches. In this work, we considered the supermartingale-based approach for probabilistic programs. An alternative approach for termination analysis is based on the notion of proof rules [28], [22], [25], [36], [34]. For example, [28] presents a proof-rule-based approach for proving finite expected termination time of probabilistic while loops, and [36] presents sound proof rules for probabilistic programs with recursion. Most results on proof rules focus on specifying local logical properties at every program counter in order to ensure a global logical property, and do not consider compositional proof rules. In contrast, our supermartingale-based approach acts as an automated proof rule that proves the almost-sure termination property. The most relevant result is given in [35] that presents a compositional approach for deriving resource bounds of probabilistic programs. Compared with our result, their result focuses on resource bounds and can only handle programs with finite expected resource consumption, whereas our result focuses on termination properties and can handle programs with infinite expected termination time.

X. Conclusion

In this paper, we first proved that a natural probabilistic extension of the variant rule in the Floyd-Hoare logic is not sound for compositional verification of almost-sure termination of probabilistic programs and identified the flaw in the previous related work [15]. Then, we proposed a sound strengthening of the approach in [15], and demonstrated an algorithmic implementation of our strengthened approach. An important future direction is to investigate different rules and sound approaches for compositional verification of probabilistic termination.

References

[1] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL*, 2(POPL):34:1–34:32, 2018.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] Michel Berkelaar, Kjell Eikland, Peter Notebaert, et al. lpsolve: Open source (mixed-integer) linear programming system. *Eindhoven U. of Technology*, 2004.

[4] Olivier Bournez and Florent Garnier. Proving positive almost-sure termination. In *RTA*, pages 323–337, 2005.

[5] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In *CAV*, pages 511–526, 2013.

[6] Krishnendu Chatterjee and Hongfei Fu. Termination of nondeterministic recursive probabilistic programs. In *VMCAI*, 2019.

[7] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through positivstellensatz's. In *CAV*, pages 3–22, 2016.

[8] Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Nastaran Okati. Computational approaches for stochastic shortest path on succinct mdps. In *IJCAI 2018*, pages 4700–4707, 2018.

[9] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *POPL*, pages 327–342, 2016.

[10] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *POPL*, pages 145–160, 2017.

[11] Guillaume Claret, Sriram K Rajamani, Aditya V Nori, Andrew D Gordon, and Johannes Borgström. Bayesian inference using data flow analysis. In *Joint Meeting on Foundations of Software Engineering*, pages 92–102. ACM, 2013.

[12] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, pages 420–432, 2003.

[13] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving termination of probabilistic programs using patterns. In *CAV*, pages 123–138, 2012.

[14] Julius Farkas. A fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikaiés Természettudományi Értesitö*, 12:457–472, 1894.

[15] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In *POPL*, pages 489–501, 2015.

[16] Robert W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, 19:19–33, 1967.

[17] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In *ESOP*, pages 282–309. Springer, 2016.

[18] Noah D Goodman, Vikash K Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. In *UAI*, pages 220–229. AUAI Press, 2008.

[19] Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014.

[20] Andrew D Gordon, Mihhail Aizatulin, Johannes Borgstrom, Guillaume Claret, Thore Graepel, Aditya V Nori, Sriram K Rajamani, and Claudio Russo. A model-learner pattern for bayesian reasoning. In *ACM SIGPLAN Notices*, volume 48, pages 403–416. ACM, 2013.

[21] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181. ACM, 2014.

[22] Wim H. Hesselink. Proof rules for recursive procedures. *Formal Asp. Comput.*, 5(6):554–570, 1993.

[23] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[24] Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. New approaches for almost-sure termination of probabilistic programs. In *APLAS*, pages 181–201, 2018.

[25] Claire Jones. *Probabilistic Non-Determinism*. PhD thesis, The University of Edinburgh, 1989.

[26] David M. Kahn. Undecidable problems for probabilistic network programming. In *MFCS*, pages 68:1–68:17, 2017.

[27] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Informatica*, pages 1–31, 2018.

[28] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of probabilistic programs. In *ESOP*, pages 364–389, 2016.

[29] Shmuel Katz and Zohar Manna. A closer look at termination. *Acta Inf.*, 5:333–352, 1975.

[30] Martin Lukasiewycz. JavaILP - java interface to ILP solvers, http://javailp.sourceforge.net/, 2008.

[31] Colin McDiarmid. Concentration. In *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–248. 1998.

[32] Annabelle McIver and Carroll Morgan. Developing and reasoning about probabilistic programs in *pGCL*. In *PSSE*, pages 123–155, 2004.

[33] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.

[34] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *PACMPL*, 2(POPL):33:1–33:28, 2018.

[35] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: resource analysis for probabilistic programs. In *PLDI*, pages 496–512, 2018.

[36] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Reasoning about recursive probabilistic programs. In *LICS*, pages 672–681, 2016.

[37] DM Roy, VK Mansinghka, ND Goodman, and JB Tenenbaum. A stochastic programming perspective on nonparametric bayes. In *Nonparametric Bayesian Workshop, Int. Conf. on Machine Learning*, volume 22, page 26, 2008.

[38] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In *SAS 2004*, pages 53–68. Springer, 2004.

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

[40] Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. Practical probabilistic programming with monads. In *ACM SIGPLAN Notices*, volume 50, pages 165–176. ACM, 2015.

[41] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets Scott: semantic foundations for probabilistic networks. In *POPL*, pages 557–571, 2017.

[42] Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93, 2000.

[43] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[44] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and implementation of probabilistic programming language anglican. In *IFL*, pages 6:1–6:12. ACM, 2016.

[45] Di Wang, Jan Hoffmann, and Thomas W. Reps. PMAF: an algebraic framework for static analysis of probabilistic programs. In *PLDI*, pages 513–528, 2018.

[46] David Williams. *Probability with Martingales*. Cambridge University Press, 1991.

APPENDIX

A. The Detailed Semantics

The behavior of a probabilistic program P accompanied with its CFG $\mathcal{G} = (L, (V_\mathrm{p}, V_\mathrm{r}), \to)$ under a scheduler σ is described as follows. The program starts in the initial configuration (ℓ_0, ν_0). Then in each *step* i ($i \in \mathbb{N}_0$), given the current configuration (ℓ_i, ν_i), the next configuration (ℓ_{i+1}, ν_{i+1}) is determined by the following procedure:

1) a valuation μ_i of the sampling variables is sampled according to the joint distribution of the cumulative distributions $\{\Upsilon_r\}_{r \in V_\mathrm{r}}$ and independent of all previously-traversed configurations (including (ℓ_i, ν_i)), all previous samplings on V_r and previous executions of probabilistic branches;
2) if $\ell_i \in L_\mathrm{a}$ and (ℓ_i, u, ℓ') is the only transition in $\to$ with source label ℓ_i, then (ℓ_{i+1}, ν_{i+1}) is set to be $(\ell', u(\nu_i, \mu r_i))$.
3) if $\ell_i \in L_\mathrm{b}$ and $(\ell_i, \phi, \ell_1), (\ell_i, \neg\phi, \ell_2)$ are namely the two transitions in $\to$ with source label ℓ_i, then (ℓ_{i+1}, ν_{i+1}) is set to be (i) (ℓ_1, ν_i) when $\nu_i \models \phi$ and (ii) (ℓ_2, ν_i) when $\nu_i \models \neg\phi$;
4) if $\ell_i \in L_\mathrm{p}$ and $(\ell_i, p, \ell_1), (\ell_i, 1-p, \ell_2)$ are namely the two transitions in $\to$ with source label ℓ_i, then with a Bernoulli experiment independent of all previous samplings, probabilistic branches and traversed configurations, (ℓ_{i+1}, ν_{i+1}) is set to be (i) (ℓ_1, ν_i) with probability p and (ii) (ℓ_2, ν_i) with probability $1-p$;
5) if $\ell_i \in L_\mathrm{d}$ and $c_0, \ldots, c_i$ is the finite path traversed so far (i.e., $c_0 = (\ell_0, \nu_0)$ and $c_i = (\ell_i, \nu_i)$) with $\sigma(c_0, \ldots, c_i) = (\ell_i, \star, \ell')$, then (ℓ_{i+1}, ν_{i+1}) is set to be (ℓ', ν_i);
6) if there is no transition in $\to$ emitting from ℓ_i (i.e., $\ell_i = \ell_\mathrm{out}$), then (ℓ_{i+1}, ν_{i+1}) is set to be (ℓ_i, ν_i).

We define the semantics of probabilistic programs using Markov decision processes.

Definition 5 (The Semantics)**.** The Markov decision process $\mathcal{M}_W = (S_W, \mathit{Act}, \mathbf{P}_W)$ (for the probabilistic program W) is defined as follows.

- The *state space* S_W is the configuration set $(L \times \mathit{Val}_{V_\mathrm{p}})$.
- The *action set* Act is $\{\tau, \mathbf{th}, \mathbf{el}\}$. Intuitively, τ refers to absence of nondeterminism and **th** (resp. **el**) refers to the **then**- (resp. **else**-) branch of a nondeterministic label.
- The *probability transition function* $\mathbf{P}_W : S_W \times S_W \to [0, 1]$ is given as follows.
  For all configurations (ℓ, ν), we have:
  - *Assignment:* if $\ell \in L_\mathrm{a}$ is an assignment label and (ℓ, u, ℓ') is the only triple in $\to$ with source label ℓ and update function u, then

    $$\mathbf{P}_W\left((\ell, \nu), \tau, (\ell', \nu')\right) := \textstyle\sum_{\mu \in U} \overline{\Upsilon}(\mu)$$

    where $U = \{\mu \mid \nu' = u(\nu, \mu)\}$;
  - *Branching:* if $\ell \in L_\mathrm{b}$ and $(\ell, \phi, \ell_1), (\ell, \neg\phi, \ell_2)$ are the two triples in $\to$ with source label ℓ and propositional arithmetic predicate ϕ, then

    $$\mathbf{P}_W\left((\ell, \nu), \tau, (\ell', \nu)\right) := \begin{cases} 1 & \text{if } \nu \models \phi, \ell' = \ell_1 \\ 1 & \text{if } \nu \not\models \phi, \ell' = \ell_2 \\ 0 & \text{otherwise} \end{cases};$$

  - *Probabilistic:* If $\ell \in L_\mathrm{p}$ and $(\ell, p, \ell_1), (\ell, 1-p, \ell_2)$ are namely two triples in $\to$ with source label ℓ, then

    $$\mathbf{P}_W\left((\ell, \nu), \tau, (\ell', \nu)\right) := \begin{cases} p & \text{if } \ell' = \ell_1 \\ 1-p & \text{if } \ell' = \ell_2 \\ 0 & \text{otherwise} \end{cases}$$

  - *Nondeterminism:* If $\ell \in L_\mathrm{d}$ and $(\ell, \star, \ell_1), (\ell, \star, \ell_2)$ are namely two triples in $\to$ with source label ℓ such that ℓ_1 (resp. ℓ_2) refers to the **then**-(resp. **else**-) branch, then

    $$\mathbf{P}_W\left((\ell, \nu), \mathbf{th}, (\ell', \nu)\right) := \begin{cases} 1 & \text{if } \ell' = \ell_1 \\ 0 & \text{otherwise} \end{cases}$$

    and

    $$\mathbf{P}_W\left((\ell, \nu), \mathbf{el}, (\ell', \nu)\right) := \begin{cases} 1 & \text{if } \ell' = \ell_2 \\ 0 & \text{otherwise} \end{cases}$$

  - *Terminal label:* if there is no transition in $\to$ emitting from ℓ_i (i.e., $\ell_i = \ell_\mathrm{out}$), then $\mathbf{P}_W\left((\ell, \nu), \tau, (\ell, \nu)\right) := 1$;
  - for other cases, $\mathbf{P}_W\left((\ell, \nu), a, (\ell', \nu')\right) := 0$.

B. Proof of Lemma 1

Lemma 1. Let the program P be the sequential (resp. conditional) composition of two other programs P_1 and P_2, i.e. $P := P_1; P_2$ (resp. $P :=$ **if** $-$ **then** P_1 **else** P_2 **fi**), and assume that both P_1 and P_2 are a.s. terminating. Then, P is also a.s. terminating.

Proof. We first prove the sequential case. Let $V_p = \{x_1, x_2, \ldots, x_m\}$ be the set of program variables in P and T, T_1, T_2 be the termination time random variables of P, P_1 and P_2, respectively. Define the vector F_1 of random variables as follows: if $\omega = \{(\ell_j, \nu_j)\}_{j \in \mathbb{N}_0}$ is a terminating run of P_1 with $T_1(\omega) = n$, i.e. if ω terminates at (ℓ_n, ν_n), then $F_1(\omega) = \nu_n$. Intuitively, $(F_1(\omega))_i$ is the random variable that models the value of the i-th program variable at termination time of P_1. Then, we have:

$$\mathbb{P}^\sigma_{\mathbf{c}}(T < \infty) = \sum_{\nu \in \mathit{Val}_{V_p}} \mathbb{P}^\sigma_{\mathbf{c}}(T_1 < \infty \ \wedge \ F_1 = \nu) \cdot \mathbb{P}^\sigma_\nu(T_2 < \infty).$$

Informally, P terminates if and only if P_1 terminates and then P_2, run with the initial valuation obtained from the last step of P_1, terminates as well. However, P_2 is a.s. terminating, hence $\mathbb{P}^\sigma_\nu(T_2 < \infty) = 1$ for all ν. Therefore,

$$\begin{aligned} \mathbb{P}^\sigma_{\mathbf{c}}(T < \infty) &= \sum_{\nu \in \mathit{Val}_{V_p}} \mathbb{P}^\sigma_{\mathbf{c}}(T_1 < \infty \ \wedge \ F_1 = \nu) \\ &= \mathbb{P}^\sigma_{\mathbf{c}}(T_1 < \infty) \\ &= 1, \end{aligned}$$

so P is also a.s. terminating.

For the conditional case, note that if P does not terminate, then at least one of P_1 and P_2 does not terminate as well. Formally, $\mathbb{P}^\sigma_{\mathbf{c}}(T<\infty) \geq \mathbb{P}^\sigma_{\mathbf{c}}(T_1<\infty)\cdot\mathbb{P}^\sigma_{\mathbf{c}}(T_2<\infty) = 1$. $\square$

C. Flaw in the Proof of FHV-rule

Below we clarify the critical point on where the flaw in [15] lies. The flaw lies in the point that RSMs should be *non-negative*. In the following, we define an extra technical notion.

CHARACTERISTIC RANDOM VARIABLES. Given random variables $X_0,\ldots,X_n$ and a predicate Φ, we denote by $\mathbf{1}_{\phi(X_0,\ldots,X_n)}$ the random variable such that

$$\mathbf{1}_{\phi(X_0,\ldots,X_n)}(\omega) = \begin{cases} 1 & \text{if } \phi\left(X_0(\omega),\ldots,X_n(\omega)\right) \text{ holds} \\ 0 & \text{otherwise} \end{cases}$$

By definition, $\mathbb{E}\left(\mathbf{1}_{\phi(X_0,\ldots,X_n)}\right) = \mathbb{P}\left(\phi(X_0,\ldots,X_n)\right)$. Note that if ϕ does not involve any random variable, then $\mathbf{1}_\phi$ can be deemed as a constant whose value depends only on whether ϕ holds or not.

This point can also be observed from the counterexample (Figure 2) that the value of the program variable x may grow unboundedly below zero due to increasing values of y, breaking the non-negativity. In detail, the flaw lies in their proof of Theorem 7.7 at the claim that the stochastic process satisfying

$$\mathbb{E}\big(R_{T_{k+1}}|\mathcal{F}_{T_k}\big) \leq R_{T_k} - \epsilon\cdot\mathbf{1}_{G_0\cap\ldots\cap G_{T_k}}$$

is an RSM. However, due to the lack of guarantee on the non-negativity of $R_{T_{k+1}}$, we cannot say that this is an RSM, although its conditional expected value decreases in each step. The rest of their proof tries to remedy this issue by enforcing the stochastic process to be non-negative. In detail, their proof constructs the stochastic process $R_{T_k}\cdot\mathbf{1}_{R_{T_k}>0}$ which is non-negative, but then this process may not satisfy the decreasing condition of RSMs. Thus, no valid RSMs are constructed in their proof, implying that the proof is invalid.

D. Proof of Theorem 2

Theorem 2. If there exists a DSM-map η for P, then for any initial valuation $\nu^* \in \mathit{Val}_{V_\mathrm{p}}$ and for all schedulers σ, we have $\mathbb{P}^\sigma_{(\ell_\mathrm{in},\nu^*)}(T<\infty) = 1$.

Proof. Let η be any DSM-map for a program P, $\nu_0 \in \mathit{Val}_{V_\mathrm{p}}$ be any initial valuation and a,b,c,ϵ be the parameters in Definition 4.

We define the stochastic process $\{X_n = \eta(\ell_n,\nu_n)\}_{n\in\mathbb{N}_0}$ adapted to $\{\mathcal{F}_n\}_{n\in\mathbb{N}_0}$ representing the evaluation of P according to the semantics. If P evaluates to a label ℓ with no out transition, then $\eta(\ell,\nu)$ is a constant c by definition.

Informally, X_n is a ranking supermartingale. If $\{X_n\}_{n\in\mathbb{N}_0}$ decreases for sufficiently many times, it will be less than c at ℓ_in which implies termination. We have $X_{B_n} \geq c$ for every $n\in\mathbb{N}_0$, where B_n is the stochastic process representing the number of steps of P's n-th arrival to the label ℓ_in. We suppose that the program Q is terminating for any initial valuation, and thus we have B_n is well defined.

$$\begin{aligned}\mathbb{P}(X_{B_n}\geq c) &= \sum_{k=n}^{+\infty}\mathbb{P}(X_k\geq c\wedge B_n = k)\\ &\leq \sum_{k=n}^{+\infty}\mathbb{P}(X_k\geq c)\end{aligned}$$

Let $Y_n = X_n + n\cdot\epsilon$, then $a+\epsilon \leq Y_{n+1}-Y_n = X_{n+1}-X_n+\epsilon \leq b+\epsilon$.

$$\begin{aligned}\mathbb{E}(Y_{n+1}|\mathcal{F}_n) &= \mathbb{E}(X_{n+1}|\mathcal{F}_n) + (n+1)\cdot\epsilon\\ &= \mathbf{1}_{(\ell_n,u,\ell')\in\to}\cdot\sum_{\mu\in\mathit{Val}_{V_\mathrm{r}}}\bar{\Upsilon}(\mu)\cdot\eta(\ell',u(\nu,\mu))\\ &\quad +\mathbf{1}_{(\ell_n,\phi,\ell')\in\to\wedge\nu_n\models\phi}\cdot\eta(\ell',\nu_n)\\ &\quad +\mathbf{1}_{(\ell_n,\star,\ell')\in\to}\cdot\eta(\ell',\nu_n)\\ &\quad +\mathbf{1}_{(\ell_n,p,\ell'),(\ell_n,1-p,\ell'')\in\to}\cdot\\ &\qquad (p\eta(\ell',\nu_n)+(1-p)\eta(\ell'',\nu_n))\\ &\quad +(n+1)\cdot\epsilon\\ &\leq \eta(\ell_n,\nu_n)-\epsilon+(n+1)\cdot\epsilon\\ &= X_n + n\cdot\epsilon\\ &= Y_n\end{aligned}$$

Thus $\{Y_n\}_{n\in\mathbb{N}_0}$ is a supermartingale satisfying the condition of Hoeffding inequality and we have

$$\begin{aligned}\sum_{k=n}^{+\infty}\mathbb{P}(X_k\geq c) &= \sum_{k=n}^{+\infty}\mathbb{P}(Y_k - Y_0 \geq c - X_0 + k\cdot\epsilon)\\ &\leq \sum_{k=n}^{+\infty} e^{-\frac{2(c-X_0+k\cdot\epsilon)^2}{k(b-a)^2}}\\ &\leq \sum_{k=n}^{+\infty} e^{-\frac{2\epsilon^2}{(b-a)^2}k-\frac{4(c-X_0)\epsilon}{(b-a)^2}}\end{aligned}$$

The above term $\to 0$ when $n\to+\infty$, And we have

$$\mathbb{P}(T_P<\infty) \geq 1 - \lim_{n\to+\infty}\mathbb{P}(X_{B_n}\geq c) = 1$$

$\square$

E. Proof System for DSM

We construct a proof system $\mathcal{D}$ for the DSM approach in the style of Hoare logic. Let R and R' be arithmetic expressions over program variables. The Hoare triple $\{R\}P\{R'\}$ indicates that for the program P, there exists a DSM η such that $\eta(\ell^P_\mathrm{in},\_) = R$ and $\eta(\ell^P_\mathrm{out},\_) = R'$.

We have following axiom schema and rules.

1) Empty statement axiom schema:

$$\frac{a\leq R'-R\leq -\epsilon}{\{R\}\ \textbf{skip}\ \{R'\}}$$

2) Assignment axiom schema:

$$\frac{a\leq R'-R\leq -\epsilon}{\{R\}x := \langle expr\rangle\{R'\}}$$

3) Sequential composition rule:

$$\frac{\{R\}P_1\{R'\}, \{R'\}P_2\{R''\}}{\{R\}P; Q\{R''\}}$$

4) Conditional branch rule:

$$\frac{\begin{array}{c}\{R_1\}P_1\{R'\}, \{R_2\}P_2\{R'\},\\ G \to a \le R_1 - R \le -\epsilon,\\ \text{and } \neg G \to a \le R_2 - R \le -\epsilon\end{array}}{\{R\}\ \textbf{if}\ G\ \textbf{then}\ P_1\ \textbf{else}\ P_2\{R'\}}$$

5) Non-deterministic branch rule:

$$\frac{\begin{array}{c}\{R_1\}P_1\{R'\}, \{R_2\}P_2\{R'\},\\ a \le R_1 - R \le -\epsilon,\\ \text{and } a \le R_2 - R \le -\epsilon\end{array}}{\{R\}\ \textbf{if}\ \star\ \textbf{then}\ P_1\ \textbf{else}\ P_2\{R'\}}$$

6) Probabilistic branch rule:

$$\frac{\begin{array}{c}\{R_1\}P_1\{R'\}, \{R_2\}P_2\{R'\},\\ a \le R_1 - R \le b,\\ a \le R_2 - R \le b\\ \text{and } p \cdot R_1 + (1-p) \cdot R_2 \le R - \epsilon\end{array}}{\{R\}\ \textbf{if}\ \textbf{prob}(\mathrm{p})\ \textbf{then}\ P_1\ \textbf{else}\ P_2\{R'\}}$$

7) While rule:

$$\frac{\begin{array}{c}\{R\}P\{R'\}, G \to R' \ge c,\\ G \to a \le R - R' \le -\epsilon,\\ \text{and } \neg G \to a \le R'' - R' \le -\epsilon\end{array}}{\{R'\}\ \textbf{while}\ G\ \textbf{do}\ P\ \textbf{od}\ \{R''\}}$$

Theorem 3. The proof system $\mathcal{D}$ is sound for a.s. termination of probabilistic programs.

Proof. We prove every rule is sound. We can find the correspondence between the rules and Definition 4.

- Empty statement axiom schema and Assignment axiom schema:
  The premise is difference boundedness and decreasing.
- Sequential composition rule:
  We just combine the two DSM-maps.
- Conditional branch rule:
  The premise corresponds to (D2). Let η_1 be DSM-map for P_1 and η_2 be DSM-map for P_2. We define the DSM-map η' for the program $Q =$ **if** G **then** P_1 **else** P_2 by $\eta'(\ell^Q_{\text{in}}, \_) := R, \eta'(\ell^Q_{\text{out}}, \_) := R'$ and $\eta'(\ell, \_) := \eta(\ell, \_)$ for other ℓ in P_1 and P_2
- Non-deterministic branch rule:
  The premise corresponds to (D3)
- Probabilistic branch rule:
  The premise corresponds to (D4)
- While rule: Suppose that $\{R\}P\{R'\}$ and the DSM of P is η, then we construct a DSM η' by defining $\eta'(\ell^Q_{\text{in}}, \_) := R'$, $\eta'(\ell^Q_{\text{out}}, \_) := R''$ and $\eta'(\ell, \_) := \eta(\ell, \_)$ for all labels ℓ in the loop body.

It is straightforward to verify that every η' is a valid DSM. By Theorem 2, we have the soundness of the proof system $\mathcal{D}$. □

F. Invariants Used in the Experiments

The following invariants were used for obtaining experimental results over the benchmark programs:

Counterexample:
$I(1) :=$true
$I(2) := x \ge 1$
$I(3) := x \ge 0 \wedge z \le y$
$I(4) := z \ge 0 \wedge z \le y$
$I(5) := x \le 1 \wedge z \ge 0 \wedge z \le y$
$I(6) := x \ge 2 \wedge z \ge 0 \wedge z \le y$
$I(7) := z \ge 0 \wedge z \le y$
$I(8) := z \le -1 \wedge z \le y$
$I(9) := z \le -1 \wedge z \le 0.25 \cdot y$

Program 1:
$I(1) :=$true
$I(2) := x \ge 1$
$I(3) := z \le y$
$I(4) := z \ge 0 \wedge z \le y$
$I(5) := z \ge -1 \wedge z \le y - 1$
$I(6) := z \le -1 \wedge z \le y$
$I(7) := z \le -1$

Program 2:
$I(1) :=$true
$I(2) := x + y \ge 1$
$I(3) := x + y \ge 1 \wedge a = z$
$I(4) := a \le z \wedge b = z$
$I(5) := a \ge 0 \wedge a \le z \wedge b = z$
$I(6) := a \ge -1 \wedge a \le z - 1 \wedge b = z$
$I(7) := a \le -1 \wedge a \le z \wedge b \le z$
$I(8) := a \le -1 \wedge a \le z \wedge b \ge 0 \wedge b \le z$
$I(9) := a \le -1 \wedge a \le z \wedge b \ge -1 \wedge b \le z - 1$
$I(10) := a \le -1 \wedge a \le z \wedge b \le -1 \wedge b \le z$
$I(11) := a \le -1 \wedge a \le 0.5 \cdot z \wedge b \le -1 \wedge b \le 0.5 \cdot z$

Program 3:
$I(1) :=$true
$I(2) := x \ge 1$
$I(3) := a \le z$
$I(4) := a \ge 0 \wedge a \le z$
$I(5) := a \ge -1 \wedge a \le z - 1$
$I(6) := a \ge -1 \wedge a \le z - 1 \wedge b \le z$
$I(7) := a \ge -1 \wedge a \le z - 1 \wedge b \ge 0 \wedge b \le z$
$I(8) := a \ge -1 \wedge a \le z - 1 \wedge b \ge -1 \wedge b \le z - 1$
$I(9) := a \ge -1 \wedge a \le z - 1 \wedge b \le -1 \wedge b \le z$
$I(10) := a \le -1 \wedge a \le z$
$I(11) := a \le -1 \wedge a \le 0.5 \cdot z$

G. Details of the Synthesized DSM-maps

Our implementation produced the following DSM-maps for the benchmark programs:

$\epsilon = 1, c = 0, [a, b] = [-70.6, 155.6]$

ℓ	$\eta(\ell, \nu)$
1	$75.4 \cdot x$
2	$75.4 \cdot x - 1$
3	$75.4 \cdot x - 0.8 \cdot y + 2$
4	$75.4 \cdot x - 0.8 \cdot y + 1$
5	$75.4 \cdot x - 0.8 \cdot y$
6	$75.4 \cdot x - 0.8 \cdot y + 80.2$
7	$75.4 \cdot x - 0.8 \cdot y - 70.6$
8	$75.4 \cdot x - 0.8 \cdot y$
9	$75.4 \cdot x - 0.8 \cdot y + 155.6$
10	$75.4 \cdot x - 0.8 \cdot y - 70.6$
11	$75.4 \cdot x - 0.8 \cdot y + 3.8$

TABLE II: The Synthesized DSM-map for Example 3

$\epsilon = 1, c = 0, [a, b] = [-4, 8]$

ℓ	$\eta(\ell, \nu)$
1	$6 \cdot x + 5$
2	$6 \cdot x + 4$
3	$6 \cdot x + 2$
4	$6 \cdot x + 1$
5	$6 \cdot x$
6	$6 \cdot x + 1$
7	$6 \cdot x$

TABLE III: The Synthesized DSM-map for Program 1

$\epsilon = 1, c = 0, [a, b] = [-5, 9.5]$

ℓ	$\eta(\ell, \nu)$
1	$7 \cdot x + 7 \cdot y + 6$
2	$7 \cdot x + 7 \cdot y + 5$
3	$7 \cdot x + 7 \cdot y + 4$
4	$7 \cdot x + 7 \cdot y + 3$
5	$7 \cdot x + 7 \cdot y + 1.5$
6	$7 \cdot x + 7 \cdot y + 0.5$
7	$7 \cdot x + 7 \cdot y + 2$
8	$7 \cdot x + 7 \cdot y + 1$
9	$7 \cdot x + 7 \cdot y$
10	$7 \cdot x + 7 \cdot y + 1$
11	$7 \cdot x + 7 \cdot y$

TABLE IV: The Synthesized DSM-map for Program 2

$\epsilon = 1, c = 0, [a, b] = [-5, 11]$

ℓ	$\eta(\ell, \nu)$
1	$8 \cdot x + 7$
2	$8 \cdot x + 3$
3	$8 \cdot x + 2$
4	$8 \cdot x + 1$
5	$8 \cdot x$
6	$-b + 8 \cdot x + z - 1$
7	$-b + 8 \cdot x + z - 2$
8	$-b + 8 \cdot x + z - 4$
9	$8 \cdot x - 1$
10	$8 \cdot x + 1$
11	$8 \cdot x$

TABLE V: The Synthesized DSM-map for Program 3