Proving Almost-Sure Termination by Omega-Regular Decomposition

Jianhui Chen
School of Software, Tsinghua University
Key Laboratory for Information System Security, MoE
Beijing National Research Center for Information Science and Technology
Beijing, China
chenjian16@mails.tsinghua.edu.cn

Fei He*
School of Software, Tsinghua University
Key Laboratory for Information System Security, MoE
Beijing National Research Center for Information Science and Technology
Beijing, China
hefei@tsinghua.edu.cn

Abstract

Almost-sure termination is the most basic liveness property of probabilistic programs. We present a novel decomposition-based approach for proving almost-sure termination of probabilistic programs with complex control-flow structure and non-determinism. Our approach automatically decomposes the runs of the probabilistic program into a finite union of *ω-regular* subsets and then proves almost-sure termination of each subset based on the notion of *localized ranking supermartingales*. Compared to the lexicographic methods and the compositional methods, our approach does not require a lexicographic order over the ranking supermartingales as well as the so-called *unaffecting* condition. Thus it has high generality. We present the algorithm of our approach and prove its soundness, as well as its relative completeness. We show that our approach can be applied to some hard cases and the evaluation on the benchmarks of previous works shows the significant efficiency of our approach.

CCS Concepts: • **Theory of computation** → **Probabilistic computation**; **Logic and verification**.

Keywords: Probabilistic Programs, Almost-Sure Termination, Omega-Regular Languages, Ranking Supermartingales

ACM Reference Format:
Jianhui Chen and Fei He. 2020. Proving Almost-Sure Termination by Omega-Regular Decomposition. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '20), June 15–20, 2020, London, UK*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3385412.3386002

*Fei He is the corresponding author.

1 Introduction

Probabilistic programs are obtained by enriching classical imperative programs with probabilistic behaviors, e.g. the variables can be assigned to some random values sampled from a probability distribution, such as *uniform*, *exponential*, *normal*, etc. Besides the probabilistic behaviors, non-determinism also plays a crucial role in probabilistic programs. Probabilistic programs can be applied to randomized algorithms, machine learning, stochastic network protocols, security systems, etc. Thus, static analysis of probabilistic programs is also very important. Termination is one of the simplest and most important liveness properties in static analysis. In the field of static analysis of classical programs, the termination problem requires that the program always terminates for all of the inputs or the initial values. But for probabilistic programs, we care about the *almost-sure (a.s.) termination*, i.e., the probabilistic program terminates with probability 1 for all of the inputs or the initial values. We also care about the *positive termination*, i.e., the expected termination time of the probabilistic program is finite.

The widely studied method to analyze termination of classical programs is the notion of *ranking functions* [10, 11, 27], which is sound and complete. The counterpart of ranking functions in probabilistic setting is the so-called *Lyapunov ranking functions* [14]. Lyapunov ranking function-based method is a sound and complete method to prove positive termination of probabilistic programs with countable state spaces but without non-determinism. Besides, another notion of ranking supermartingales extends ranking functions for probabilistic programs with real-valued variables [5] and non-determinism [7, 13]. These supermartingale-based methods are limited only to probabilistic programs with rather restricted control-flow structure. Agrawal et al. [1] introduced the notion of lexicographic ranking supermartingales (*LexRSM*). Ferrer Fioriti and Hermanns [13], as well as Huang et al. [20], proposed the compositional methods based on ranking supermartingales. All of these methods can be applied to prove a.s. termination of probabilistic programs with complex control-flow structure. However, they all require a lexicographic order over the supermartingales and the *unaffecting* condition [1, 13], which are hard to be fulfilled in complicated situations. The simple probabilistic program presented in Example 1.1 has no *LexRSM*. Also, the *unaffecting* condition can not be met in this example. But actually, we can prove a.s termination of this program by a collection of linear ranking supermartingales and we do not require a lexicographic order or the *unaffecting* condition of them.

When we look at this program, two traces in the **while** loop can be found. The first trace is $\rho_1 = \boxed{x \neq 0}\ \boxed{x > 0}\ \boxed{prob(0.5)}\ \boxed{x := x - 1}$, and the second is $\rho_2 = \boxed{x \neq 0}\ \boxed{x \leq 0}\ \boxed{x := x + 1}$. If we leave out the semantic of the program, ρ_1 and ρ_2 can be arbitrarily interleaved during program executing. Suppose the program is not almost-surely (a.s.) terminating, then the set of non-terminating traces should be $\mathcal{T} = (\rho_1|\rho_2)^\omega$. We can easily prove that the subset $\mathcal{T}_1 = {\rho_1}^\omega$ is a.s. terminating by a ranking supermartingale x, since the integer variable x is decreased to 0 almost-surely. For the same reason, we can prove that another subset $\mathcal{T}_2 = {\rho_2}^\omega$ is a.s. terminating by a ranking supermartingale $-x$. Then, the remaining subsets of $\mathcal{T}$ are the cases of ρ_1 and ρ_2 interleaving, i.e., $\mathcal{T}_3 = (({\rho_1}^+\rho_2) \mid ({\rho_2}^+\rho_1))\ (\rho_1|\rho_2)^\omega$. It can be easily found that all of the trace in $\mathcal{T}_3$ are infeasible since the sequential compositon of either $\rho_1\rho_2$ or $\rho_2\rho_1$ is blocked. Thus $\mathcal{T}_3$ is also a.s terminating. Meanwhile, we have $\mathcal{T} \subseteq \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$. In this way, we have proved the a.s termination of this probabilistic program.

Such a decomposition-based approach has many advantages compared to the methods mentioned above. In general, synthesizing a.s. terminating arguments for an ω-regular subset should be more efficient than that for the entire program. The a.s. terminating arguments for the specialized subsets are also rather simple, i.e., we propose a notion of *localized ranking supermartingale*s for proving a.s termination of the ω-regular trace sets. The decomposition-based approach is also naturally parallelizable. Actually, analogous approaches are proved to be efficient in proving termination of large-scale classical programs [9, 18]. However, these approaches can not be applied to probabilistic programs. They are aimed at proving that every single trace is terminating. But in probabilistic setting, a.s. termination require us to prove a set of traces terminating with probability one. Nevertheless, some non-terminating traces with probability zero can still exist. The theoretic foundation of the decomposition-based approach in probabilistic setting is rather intricate. Moreover, the so-called *rank certificate*s of these approaches is also not applicable to probabilistic programs. We resolve all of the problems above by proposing a brand new decomposition-based approach.

We summarize our main contributions to proving a.s termination as below:

- We propose a new approach to a.s. termination which is based on ω-regular decomposition. Our approach is applicable to probabilistic programs with rather complex control-flow structure.
- We propose a notion of *localized ranking supermartingale* to prove a.s termination of the ω-regular probabilistic trace sets, which can be synthesized efficiently.
- We prove the soundness of our approach, as well its relative completeness.
- We show our approach is applicable to some hard cases. The evaluation on the benchmarks of previous works shows the significant efficiency of our approach.

Example 1.1. The variable x is integer-valued.

```
while x ≠ 0 do
  if x > 0 then
    if prob(0.5) then x := x - 1;
  else x := x + 1; fi
od
```

2 Preliminaries

2.1 Basic Notions

For a set A, we denote the cardinality of A as $|A|$, and denote the power set of A as 2^A. We denote by $\mathbb{N}$, $\mathbb{N}_0$, $\overline{\mathbb{N}}_0$, $\mathbb{Z}$ and $\mathbb{R}$ the sets of all positive integers, non-negative integers, non-negative integers with $+\infty$, integers and real numbers, respectively. We use boldface notation to denote vectors, e.g. $\mathbf{x}$, $\mathbf{y}$, etc. We denote the i-th component of a vector $\mathbf{x}$ by $\mathbf{x}[i]$.

2.2 Omega-Regular Language and Büchi Automaton

Omega-Regular Language. An ω-language $\mathcal{L}$ is *ω-regular* if it has the form

- A^ω where A is a nonempty regular language not containing the empty string.
- AB, the concatenation of a regular language A and an ω-regular language B.
- $\bigcup_{i=1}^{n} A_i$ where each A_i is ω-regular languages.

The *ω-regular language*s generalize the definition of *regular language*s to infinite words. They can be recognized by *Büchi automata.*

Büchi Automata. A *deterministic Büchi automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ that consists of the following components:

- Q is a finite set of the states of $\mathcal{A}$.
- Σ is a finite set called the alphabet of $\mathcal{A}$.
- $\delta : Q \times \Sigma \longrightarrow Q$ is the transition function of $\mathcal{A}$.
- $q_0 \in Q$ is the initial state of $\mathcal{A}$.
- F $\subseteq Q$ is a set of accepting states of $\mathcal{A}$.

The ω-regular language recognized by the Büchi automaton $\mathcal{A}$ is denoted as $\mathcal{L}(\mathcal{A})$. For any string ρ over Σ, we say ρ is accepted by $\mathcal{A}$ if and only if there exists $q \in F$ occurring infinitely often in the run of ρ, denoted as $\rho \in \mathcal{L}(\mathcal{A})$. The set of Büchi automata is closed under the operations of *union*, *intersection*, *concatenation* and *complementation*.

2.3 Probability Theory

Measure and Probability. Let Ω be a nonempty set. A σ-algebra $\mathcal{F}$ is a class of subsets of Ω, such that $\Omega \in \mathcal{F}$ and $\mathcal{F}$ is closed under *complement* and *countable union*. A set A is $\mathcal{F}$*-measurable* if $A \in \mathcal{F}$. A function $f : \Omega \to \mathbb{R}$ is $(\mathcal{F}, \mathcal{B}(\mathbb{R}))$*-measurable* or just $\mathcal{F}$-measurable if $f^{-1}(A) \in \mathcal{F}$ for all $A \in \mathcal{B}(\mathbb{R})$, where $\mathcal{B}(\mathbb{R})$ the smallest σ-algebra which contains all the subintervals in $\mathbb{R}$, called the *Borel σ-algebra*. The set of measurable functions is closed under *algebraic operations* and *point-wise sequential limits*, i.e., $\sup_{k\in\mathbb{N}}\{f_k\}$ or $\lim\inf_{k\in\mathbb{N}}\{f_k\}$. A *probability space* is a triple $(\Omega, \mathcal{F}, \mu)$, where Ω is a nonempty set (so-called *sample space*), $\mathcal{F}$ is a σ-algebra on Ω, and $\mu : \mathcal{F} \to [0, 1]$ is a probability measure on $(\Omega, \mathcal{F})$ such that $\mu(\Omega) = 1$ and $\sum_{i=1}^{\infty} \mu(A_i) = \mu\left(\bigcup_{i=1}^{\infty} A_i\right)$ for all pairwise-disjoint sets $A_1, A_2, \dots \in \mathcal{F}$.

Random Variables, Filtrations and Stopping Time. Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. A *random variable* X is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R}$. We denote by $\mathbb{E}(X)$ the *mathematical expectation* of X such that $\mathbb{E}(X) \triangleq \int_\Omega X d\mu$, where $\int$ is the Lebesgue integral [2]. A sequence of σ-algebra $\{\mathcal{F}_i\}_{i=0}^{\infty}$ is called a *filtration* if $\mathcal{F}_j \subseteq \mathcal{F}_k \subseteq \mathcal{F}$ for all $j \leq k$. The increasing sequence of σ-algebras can represent the growing amounts of the information. In addition, $(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mu)$ is called *filtered probability space*. Let $T : \Omega \to \overline{\mathbb{N}}_0$ be a random variable defined on that filtered probability space. Then T is a *stopping time* w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$ if $\{T \leq i\}$ is $\mathcal{F}_i$-measurable for all $i \in \overline{\mathbb{N}}_0$.

Conditional Expectation. Let $(\Omega, \mathcal{F}, \mu)$ be a probability space, $X : \Omega \to \mathbb{R}$ a random variable on that probability space with finite expectation, and $\mathcal{H} \subseteq \mathcal{F}$ a sub-sigma-algebra of $\mathcal{F}$. A *conditional expectation* of X, denoted as $\mathbb{E}(X|\mathcal{H})$, is an $\mathcal{H}$-measurable function which satisfies:

$$\int_{\mathcal{H}} \mathbb{E}(X|\mathcal{H}) \mathrm{d}\mu = \int_{\mathcal{H}} X \mathrm{d}\mu$$

The definition above does not guarantee that the conditional expectation is existing or uniquely defined. Sometimes the *uniform integrability* [2] of X is required. In this paper, we employ Proposition 3.1 from [1] to guarantee the existence of the conditional expectation, i.e., $\mathbb{E}(|X|) < \infty$ or X is real-valued and non-negative. In the rest of this paper, we suppose it is satisfied acquiescently.

Supermartingales and Ranking Supermartingales. Let $(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mu)$ be a filtered probability space. A stochastic process $\{X_i\}_{i=0}^{\infty}$ is a *supermartingale* w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$ if $\mathbb{E}(X_{n+1}|\mathcal{F}_n) \leq X_n$. Moreover, let $\epsilon \geq 0$ and T be a stopping time w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$. $\{X_i\}_{i=0}^{\infty}$ is an *ϵ-ranking supermartingale* if $X_n \geq 0$ and $\mathbb{E}(X_{n+1}|\mathcal{F}_n) \leq X_n - \epsilon \mathbf{1}_{\{T>n\}}$.

2.4 Probabilistic Programs

2.4.1 Syntax. The grammar of probabilistic programs is presented below.

```
⟨statement⟩   ::= ⟨assignment⟩ | 'skip'
               |  ⟨statement⟩ ';' ⟨statement⟩
               |  'if' ⟨ndpbexpr⟩ 'then' ⟨statement⟩
                  'else' ⟨statement⟩ 'fi'
               |  'while' ⟨bexpr⟩ 'do' ⟨statement⟩ 'od'

⟨assignment⟩  ::= ⟨pvar⟩ ':=' ⟨expr⟩
               |  ⟨pvar⟩ ':=' ⟨distribution⟩

⟨expr⟩        ::= ⟨constant⟩ | ⟨pvar⟩
               |  ⟨constant⟩ '·' ⟨pvar⟩
               |  ⟨expr⟩ '+' ⟨expr⟩ | ⟨expr⟩ '-' ⟨expr⟩

⟨bexpr⟩       ::= ⟨expr⟩ '>' ⟨expr⟩ | ⟨expr⟩ '<' ⟨expr⟩
               |  ⟨expr⟩ '=' ⟨expr⟩
               |  ⟨bexpr⟩ 'and' ⟨bexpr⟩
               |  ⟨bexpr⟩ 'or' ⟨bexpr⟩
               |  'not' ⟨bexpr⟩

⟨ndpbexpr⟩    ::= '*' | 'prob(p)' | ⟨bexpr⟩
```

Here $\langle pvar \rangle$ is the variables appearing in probabilistic programs. $\langle distribution \rangle$ stands for the values sampled from a probability distribution like *uniform*, *exponential*, *normal*, etc. The production $\langle ndpbexpr \rangle$ is the *guard* of **if-then-else** and **while** statements. It can be "*****", which represents a non-deterministic choice between the branches, or **prob(p)**, which represents the probabilistic choices $P_1 \oplus_p P_2$ in the style of McIver et al. [24]. This notation can be considered as syntax sugar for "$c := uniform(0, 1)$; **if** $c < p$ **then** P_1 **else** P_2 **fi**". The guard can also be a predicate $\langle bexpr \rangle$ representing the deterministic conditional branching. The operators of the logic and the arithmetic can be extended naturally, e.g. $\geq, \leq, \neq$, etc.

Example 2.1. An example of the probabilistic program below is first presented in [1]. It has a 3-dimensional lexicographic supermartingale. We employ it to demonstrate our decomposition-based approach.

```
0 assume x ≥ 0 and y ≥ 0;
1 while x ≥ 0 and y ≥ 0 do
2   if * then
3     if prob(0.5) then
4       x := x - 2;
5     else skip; fi
6   else
7     x := 2x; y := y - 1;
8   fi
9 od
```

2.4.2 Semantics. The semantics of a real-valued probabilistic program can be defined as an uncountable state-space

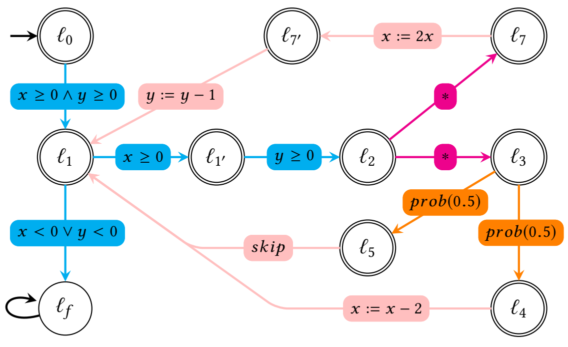

Figure 1. *pCFA* of Example 2.1.

Markov decision process. We define its semantics on the so-called *probabilistic control-flow automaton* (*pCFA*) and associate each program with a set of stochastic processes.

Probabilistic Control-flow Automata. The *probabilistic control-flow automaton* is a kind of representation of the probabilistic programs. It is a probabilistic generalization of the *program graph* [25] or the *control-flow automaton* [3] of classical programs. It can also be regarded as a variant of the *probabilistic control-flow graphs* [7], where the nodes are the program locations and each edge is labeled with a single program statement.

Definition 2.2 (Probabilistic Control-flow Automaton)**.** A *probabilistic control-flow automaton* is a tuple $\mathcal{P} = (L, \Delta, \Sigma$ $V, \ell_{init}, \ell_{term}, F)$ where

- L is a set of locations.
- Σ is a set of statements labeled on the control-flow edges.
- $\Delta \subseteq L \times \Sigma \times L$ is a set of transitions representing the control-flow edges.
- V is a set of variables occurring in statements of Σ.
- $\ell_{init} \in L$ is the initial location.
- $\ell_{term} \in L$ is the terminating location.
- $F \subseteq L$ is a set of accepting states.

For every *pCFA*, we require that each location has at least one outgoing transition. We assume there is a self-loop transition τ_{id} with an *id* statement st_{id} on the location ℓ_{term}. The *id* statement is like the **skip** statement which does not change the value of any program variables. We also assume that ℓ_{term} follows the location after the last outermost loop of the probabilistic programs. Every assignment operation is a tuple (i, u), where $1 \leq i \leq |V|$ is an index of the target variable and the update element u is a $\mathcal{B}(\mathbb{R})$-measurable function $u : \mathbb{R}^{|V|} \longrightarrow \mathbb{R}$ or a value sampled form a probability distribution d with expectation $\mathbb{E}[d]$. In a *pCFA*, there can be multiple control-flow edges from location ℓ_i to ℓ_j. For each transition $\tau \in \Delta$, we have $\tau \triangleq (\ell_i, st, \ell_j)$ such that ℓ_i is the source location and ℓ_j is the target location. We denote the statement st of τ as $St(\tau)$. If $St(\tau)$ is a deterministic branching statement, we denote the guard predicate of τ as $G(\tau)$. If $St(\tau)$ is a probabilistic branching statement, we denote the probability of τ as $Pr(\tau)$. In this case, we require there is another transition τ' outgoing form ℓ_i such that $Pr(\tau') = 1 - Pr(\tau)$. We call τ' the *complementary transition* of τ. This requirement is important and we call it *probability completeness.*

It is intuitively clear that any probabilistic program can be naturally transformed into a *pCFA*. The method is similar to the standard transformation method from a probabilistic programs to its *pCFG* [7]. When we transform a probabilistic program into a *pCFA*, **if-then-else** and **while** statements are replaced by **assume** statements on the control-flow edges. We omit the keyword "**assume**" and label only the $\langle$*ndpbexpr*$\rangle$ on the edges, i.e., a predicate, a probability or a star. Different to the $\langle$*statement*$\rangle$ defined in the grammar, we call the statements labeled on the control-flow edges the *unitary statements*. The unitary statements can be considered as the smallest units of the statements. In the rest of this paper, we always consider the unitary statements instead of the statements defined in the grammar.

Configurations, Runs and Traces. A configuration of a *pCFA* $\mathcal{P}$ is a tuple $C \triangleq (\ell, \mathbf{x})$, where ℓ is a location of $\mathcal{P}$ and $\mathbf{x}$ is an $|V|$-dimensional vector. We say that a transition τ is *enabled* on a configuration $(\ell, \mathbf{x})$ if ℓ is the source location of τ and (a) $St(\tau)$ is a deterministic branching statement and $\mathbf{x} \models G(\tau)$, or (b) $St(\tau)$ is not a deterministic branching statement. We say $(\ell', \mathbf{x}')$ is a *successor configuration* of $(\ell, \mathbf{x})$ via a transition $\tau = (\ell, st, \ell')$ if $\mathbf{x}'$ is calculated following:

- If $St(\tau)$ is not an assignment statement, then $\mathbf{x}' = \mathbf{x}$.
- If $St(\tau)$ is an assignment statement with update tuple (j, u), then $\mathbf{x}'$ equals to ($\mathbf{x}$ with $\mathbf{x}[j] \longleftarrow u(\mathbf{x})$) if u is a $\mathcal{B}(\mathbb{R})$-measurable function, or equals to a sample value from u if u is a distribution.

Associate $\mathcal{P}$ with a set of initial vectors Ξ_{init} and then we can define the paths on $\mathcal{P}$. A *computation* of length k in $\mathcal{P}$ is a finite sequence of configurations $\pi \triangleq (\ell_0, \mathbf{x}_0)$ $\cdots (\ell_k, \mathbf{x}_k)$ such that $\ell_0 = \ell_{init}, \mathbf{x}_0 \in \Xi_{init}$, and the configuration $(\ell_{i+1}, \mathbf{x}_{i+1})$ is a successor configuration of $(\ell_i, \mathbf{x}_i)$ for each $0 \leq i < k$. A *run* in $\mathcal{P}$ is an *infinite* sequence of configurations whose every finite prefix is a computation. Meanwhile, based on the unitary statements, the computations and runs can also be defined as a sequence of statements, i.e., $\pi_\Sigma \triangleq st_0\ st_1\ \cdots$, where st_0 is a virtual statement assigning the program variables to the initial vector $\mathbf{x}_0$ and each st_i is $St(\tau_i)$ such that τ_i is enabled on the configuration $(\ell_{i-1}, \mathbf{x}_{i-1})$. A *statement trace* is a sequence of unitary statements $\rho \triangleq st_1\ \cdots\ st_k\ \cdots$, which represents a set of computations or runs beginning with any initial vectors. We say a run π is *terminating* if it reaches a configuration whose first component is ℓ_{term}, or equivalently, π_Σ contains st_{id}. We say a trace ρ is *terminating* if it contains st_{id}.

Remark 1. *It is important that the pCFA $\mathcal{P}$ can be viewed as a Büchi automaton. The locations in L are the states of the Büchi automaton. The set of unitary statements Σ is the alphabet. The locations except the terminating location ℓ_{term} are accepting states. As a result, the ω-regular languages recognized by $\mathcal{P}$ are obviously the statement traces representing the non-terminating runs of $\mathcal{P}$, i.e., $\{\rho \mid \rho \text{ is not terminating}\} \subseteq \mathcal{L}(\mathcal{P})$.*

Example 2.3. Figure 1 is the *pCFA* transformed from the probabilistic program in Example 2.1. The set of unitary statements is $\Sigma \triangleq \{ \boxed{x \geq 0}, \boxed{y \geq 0}, \boxed{x < 0}, \boxed{y < 0}, \boxed{prob(0.5)}, \boxed{*}, \boxed{x := x - 2}, \boxed{x := 2x}, \boxed{y := y - 1} \}$. Except the terminating location ℓ_f, other locations are all accepting states.

Stochastic Process. The probabilistic behaviors of $\mathcal{P}$ can be quantitatively captured by constructing a suitable probability measure over the set of its runs. However, the non-deterministic behaviors are not allowed in the probability space. There are two kind of non-deterministic behaviors in a *pCFA*, i.e., (a) the non-deterministic branching statements, and (b) multiple transitions enabled on a given configuration $(\ell, \mathbf{x})$. We do not distinguish between these two kinds of non-determinism. We employ the *automaton scheduler* to resolve the non-determinism.

Definition 2.4 (Automaton Scheduler)**.** An *automaton scheduler* of a *pCFA* $\mathcal{P}$ is a function σ assigning to every computation π a probability distribution on its successor transition. Let Δ_e be the set of transitions enabled on the last configuration of π. We select a non-probabilistic transition τ from Δ_e with probability 1, or a couple of probabilistic transitions, i.e., τ and its complementary transition τ', with the probability p labeled on them. The unselected transitions in Δ_e have probability 0 on the distribution.

Remark 2. *The automaton scheduler does not distinguish between deterministic and non-deterministic transitions. So we have a probability distribution on every step. It can be regarded as a meticulous and special case of the* scheduler *defined in [1]. In the rest of this paper, we call it* scheduler *for short.*

A *pCFA* $\mathcal{P}$ associated with an initial vector $\mathbf{x}_{init} \in \Xi_{init}$ and a scheduler σ can uniquely define a stochastic process $\{\mathbf{C}_i\}_{i=0}^{\infty}$ which produces a random run $(\ell_0, \mathbf{x}_0)\,(\ell_1, \mathbf{x}_1)\cdots$. It begins with the initial configuration $(\ell_0, \mathbf{x}_0) = (\ell_{init}, \mathbf{x}_{init})$. Assume i steps have elapsed, i.e., a computation $\pi_i = (\ell_0, \mathbf{x}_0)\cdots(\ell_i, \mathbf{x}_i)$ has already been produced. Then a transition τ is sampled from the distribution $\sigma(\pi_i)$ and the successor configuration $(\ell_{i+1}, \mathbf{x}_{i+1})$ of $(\ell_i, \mathbf{x}_i)$ via the transition τ can be calculated by definitions. We call $\{\mathbf{C}_i\}_{i=0}^{\infty}$ the *canonical stochastic process.*

Remark 3. *We can also formalize the definition of $\{\mathbf{C}_i\}_{i=0}^{\infty}$, as well as its probability space $(\Omega, \mathcal{F}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$, which is uniquely determined by $\mathcal{P}$ together with $\mathbf{x}_{init}$ and σ. The sample space Ω is a set of all random runs in $\mathcal{P}$. $\mathbf{C}_i(\pi)$ denotes the i-the configuration on the random run π. Each $\mathbf{C}_i$ can be regarded as a vector-valued function that maps a run $\pi \in \Omega$ to the $|V| + 1$ dimensional vector space of the configurations. The function $\mathbf{L}_i(\pi)$ and $\mathbf{S}_i(\pi)$ denote the i-th location and the i-th statement of the random run π respectively. All of $\mathbf{C}_i$, $\mathbf{L}_i$ and $\mathbf{S}_i$ for $i \in \overline{\mathbb{N}}_0$ should be $\mathcal{F}$-measurable. Moreover, $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}$, as well as $\mathcal{F}$, are defined as the measure and the σ-algebra on the infinite product space respectively. They are derived from the production of the measurable spaces by the so-called* cylinder construction *in [2]. Besides, we can define the filtration $\{\mathcal{F}_i\}_{i=0}^{\infty}$ such that each $\mathcal{F}_i$ is the smallest σ-algebra that makes $\mathbf{C}_j$, $\mathbf{L}_j$ and $\mathbf{S}_j$ $\mathcal{F}_j$-measurable for $0 \leq j \leq i$. The filtration represents the information gained along with the execution of the probabilistic programs, with which we can distinguish more and more runs diverged on the branching statements in Ω.*

2.5 Almost-Sure Termination

Consider a *pCFA* $\mathcal{P}$ associated with an initial vector $\mathbf{x}_{init}$ and a scheduler σ, we have the filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$. Then we can define a random variable $Tm : \Omega \rightarrow \overline{\mathbb{N}}_0$ such that for each run $\pi \in \Omega$, the value $Tm(\pi)$ represents the first time that the terminating location ℓ_{term} is reached (or equivalently, the statement st_{id} appears in π_Σ). If π does not terminate, then $Tm(\pi) = \infty$. We call Tm the *termination time* of $\mathcal{P}$. Tm is apparently a stopping time w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$. We are interested in the termination time of all the runs, then we have the concept of *almost-sure termination.*

Definition 2.5 (Almost-Sure Termination [1])**.** A probabilistic program P is *almost-surely terminating* if for any initial vector $\mathbf{x}_{init} \in \Xi_{init}$ and any scheduler σ, it holds that $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) < \infty\}) = 1$.

There are many supermartingale-based methods to prove a.s. termination of probabilistic programs [1, 5, 7, 13]. Their mathematical foundations are similar to the following theorem.

Theorem 2.6 (Almost-Sure Termination)**.** *Given of a pCFA $\mathcal{P}$ with a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$, if there exists an ϵ-ranking supermartingale w.r.t. $\{\mathcal{F}_i\}_{i=0}^{\infty}$, then we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) < \infty\}) = 1$.*

3 Certified Stochastic Modules

As discussed in Section 1, we can partition the set of all non-terminating runs into a collection of ω-regular subsets, and then prove a.s. termination of the runs in each subset by a collection of rather simple arguments. In analogous approaches [9, 18], ranking functions and invariants, together with Floyd-Hoare logic, are used to prove termination of the traces of classical programs, and to extend a single trace to a general trace set. However, most of these notions are not applicable to probabilistic programs. For this reason, we propose a brand new approach with the help of the notions in the probabilistic setting. In this section, we provide the theoretical foundations of our decomposition-based approach. We first present the notion of *stochastic module*s, which can represent a set of ω-regular probabilistic runs effectively. We also propose the specialized a.s. terminating arguments for stochastic modules. Then we present the notion of *certified stochastic module*s, which can be used to generalize the ω-regular set to a larger set while maintain the identical a.s. terminating arguments.

3.1 Stochastic Modules

We use *stochastic module*s to represent the sets of probabilistic runs. A stochastic module is a *pCFA*. But different to the *pCFA* of the entire probabilistic program, the stochastic module is constructed from a lasso-shape ω-statement trace and it has *exactly one* accepting state. Thus, a stochastic module recognizes an ω-regular language in the form of $\mathcal{U} \cdot \mathcal{V}^\omega$, where $\mathcal{U}$ and $\mathcal{V}$ are regular languages over the alphabet of the unitary statements. We call $\mathcal{U}$ the *stem* of the trace and $\mathcal{V}$ the *loop* of the trace. We denote the accepting state as ℓ_a. It is clear that ℓ_a is visited *infinitely often* in every run of the trace accepted by the stochastic module. A stochastic module is called a *lasso module* if it is constructed from a lasso-shape ω-statement trace directly. The control-flow structure of a lasso module is rather simpler than that of an entire probabilistic program. There is no interleaving of the non-probabilistic branches in the *loop* part of the traces. So, synthesizing a.s. terminating arguments for lasso modules should be rather easy.

Example 3.1 (Stochastic Modules)**.** Consider the *pCFA* $\mathcal{P}$ in Figure 1. It can be decomposed into two stochastic modules. Let $\rho_s = \boxed{x \geq 0}\ \boxed{y \geq 0}$ represent the guards before the main loop. Let $\rho_1 = \rho_s\ [\,(\ \boxed{\mathit{prob}(0.5)}\ \boxed{x := x - 2}\ )\mid(\ \boxed{\mathit{prob}(0.5)}\ \boxed{\mathit{skip}}\ )\,]$ represent the trace in the main loop with a probabilistic branching statement. Let $\rho_2 = \rho_s\ \boxed{*}\ \boxed{x := 2x}\ \boxed{y := y - 1}$ represent another trace in the main loop. The set of non-terminating traces in this program is $\mathcal{T} = \rho_s(\rho_1\rho_2)^\omega$. We can decompose $\mathcal{T}$ into two subset $\mathcal{T}_1 = \rho_s(\rho_1|\rho_2)^*{\rho_1}^\omega$ and $\mathcal{T}_2 = \rho_s({\rho_1}^*\rho_2)^\omega$ such that $\mathcal{T} \subseteq \mathcal{T}_1 \cup \mathcal{T}_2$. Then we have the first stochastic module $\mathcal{P}_1$ recognizing $\mathcal{T}_1$, which is presented in Figure 4. We will show how to build it automatically in Section 4. The second stochastic module $\mathcal{P}_2$ recognizes $\mathcal{T}_2$. Thus $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\mathcal{P}_1) \cup \mathcal{L}(\mathcal{P}_2)$.

3.1.1 Localized Ranking Supermartingales.

The ranking supermartingale [1, 5, 13] for probabilistic programs requires its expectation value to decrease by at least ϵ after every step. However, our new notion of the *localized ranking supermartingales* (*LocRSM*) for stochastic modules does not require such a strict condition. It is more like the variant of the supermartingale in the work of McIver et al. [23], i.e., its expectation value decreases after every loop iteration. Our *LocRSM* just requires its expectation value to decrease strictly after the accepting state ℓ_a is visited.

Example 3.2 (*LocRSM*)**.** In Figure 3, the accepting state is ℓ_1. We can define a stochastic process R over the values of the blue expressions, e.g. $x+3$, $x+2$, 0, on each step. We can find that the expectation value of R decreases by at least 1 after ℓ_1 is visited ($x+3 \longrightarrow x+2$ and $x+3 \longrightarrow 0$) and dose not increase after other locations are visited. So, R is a *LocRSM*.

Given a stochastic module $\mathcal{A}$ associated with an initial vector $\mathbf{x}_{init} \in \Xi_{init}$ and a scheduler σ, we can also define a filtered probability space $(\Omega_\mathcal{A}, \mathcal{F}_\mathcal{A}, \{\mathcal{F}_i\}_{i=0}^\infty, \mathbb{P}^\sigma_{\mathbf{x}_{init}})$. It is clear that $\{\mathbf{L}_n = \ell_a\}$ is $\mathcal{F}_n$-measurable. Now we can give a formal definition of *LocRSM*.

Definition 3.3 (Localized Ranking Supermartingale)**.** Given a constant $\epsilon > 0$ and a stochastic module $\mathcal{A}$, a stochastic process $\{R_i\}_{i=0}^\infty$ w.r.t. $\{\mathcal{F}_i\}_{i=0}^\infty$ is a *localized ranking supermartingale* for $\mathcal{A}$ if $R_n \geq 0$ and $\mathbb{E}(R_{n+1}|\mathcal{F}_n) \leq R_n - \epsilon \mathbf{1}_{\{\mathbf{L}_n = \ell_a\}}$.

3.1.2 A.S. Termination of Stochastic Module.

The notion of *LocRSM* can be used to prove a.s termination. The following lemma provides us the mathematical foundation to prove a.s. termination of stochastic modules.

Lemma 3.4. *Given a stochastic module $\mathcal{A}$ associated with an initial vector $\mathbf{x}_{init} \in \Xi_{init}$ and a scheduler σ, let $(\Omega_\mathcal{A}, \mathcal{F}_\mathcal{A}, \{\mathcal{F}_i\}_{i=0}^\infty, \mathbb{P}^\sigma_{\mathbf{x}_{init}})$ be the filtered probability space. If there is a localized ranking supermartingale $\{R_i\}_{i=0}^\infty$ for $\mathcal{A}$ w.r.t. $\{\mathcal{F}_i\}_{i=0}^\infty$, then we have $\mathbb{P}^\sigma_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$.*

Proof. For all $\omega \in \Omega$, let $\alpha_k(\omega) = \sup\{i \in \mathbb{N}_0 \mid i < k \wedge \mathbf{L}_i(\omega) = \ell_a\}$, $\beta_k(\omega) = \inf\{i \in \mathbb{N}_0 \mid i > k \wedge \mathbf{L}_i(\omega) = \ell_a\}$ and $\Delta_k(\omega) = \beta_k(\omega) - \alpha_k(\omega)$. All of α_k, β_k and Δ_k are $\mathcal{F}_\mathcal{A}$-measurable. Because all of β_k and Δ_k are finite, we suppose there exists an $M \in \mathbb{N}_0$ such that M is the upper bound of β_k and Δ_k. Then we define a stochastic process $\{Y_i\}_{i=0}^\infty$ as follows:

$$Y_k(\omega) = \begin{cases} R_k(\omega) + (1 - \frac{k}{M})\epsilon & \text{if } \alpha_k(\omega) \text{ does not exist} \\ R_k(\omega) & \text{if } \mathbf{L}_k(\omega) = \ell_a \text{ or} \\ & Tm(\omega) \leq k \\ R_k(\omega) + (1 - \frac{k - \alpha_k(\omega)}{M})\epsilon & \text{if } \alpha_k(\omega) \text{ exists and} \\ & Tm(\omega) > k \end{cases}$$

First, we can easily have $Y_n \geq R_n \geq 0$ and $\mathbb{E}(|Y_n|) < \mathbb{E}(|R_n|) + 2\epsilon < \infty$. Since α_k and $\{Tm \leq k\}$ are $\mathcal{F}_k$-measurable, then Y_k is also $\mathcal{F}_k$-measurable. Let $\epsilon_m = \epsilon/M$, we have $\epsilon_m > 0$. For $\{\mathbf{L}_n = \ell_a\}$, we have $\alpha_{n+1} = n$ and $Y_n - \mathbb{E}(Y_{n+1}|\mathcal{F}_n) = R_n - \mathbb{E}(R_{n+1} + (1 - \frac{n+1-n}{M})\epsilon|\mathcal{F}_n) = R_n - \mathbb{E}(R_{n+1}|\mathcal{F}_n) - (1 - \frac{1}{M})\epsilon \geq \epsilon/M = \epsilon_m$. In other cases, we can also prove $Y_n - \mathbb{E}(Y_{n+1}|\mathcal{F}_n) \geq \epsilon_m$. For $\{Tm \leq n\}$, we can prove $Y_n - \mathbb{E}(Y_{n+1}|\mathcal{F}_n) \geq 0$. Then by the definition of ϵ-ranking supermartingales, $\{Y_i\}_{i=0}^\infty$ is obviously an ϵ_m-ranking supermartingale w.r.t. $\{\mathcal{F}_i\}_{i=0}^\infty$. Hence, we have $\mathbb{P}^\sigma_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$ according to Theorem 2.6. □

3.2 Certified Stochastic Module

In this section, we present the notion of *certified stochastic module*. It is a stochastic module together with a.s. terminating arguments (Lemma 3.4), i.e., the *LocRSM* and its *support invariants*. The a.s. terminating arguments are based on the notions of *expression maps* and *invariant maps*. Moreover, a certified stochastic module built from a lasso module can be extend to a more general certified stochastic module, which contains more traces sharing the identical a.s. terminating arguments. This extending is very important to the efficiency of our approach.

3.2.1 Expression Map on Stochastic Modules. An expression map for a stochastic module $\mathcal{A}$ is a function $\eta : L \to \mathbb{R}^{|V|} \to \mathbb{R}$ assigning to each location $\ell \in L$ a $\mathcal{B}(\mathbb{R})$-measurable function $\eta(\ell)$. Additionally, if all of the functions $\eta(\ell)$ are affine, we call η a *linear expression map*. We can build a stochastic process $\{R_i\}_{i=0}^{\infty}$ based on an expression map η and the canonical stochastic process $\{\mathrm{C}_i \triangleq (\ell_i, \mathbf{x}_i)\}_{i=0}^{\infty}$, i.e., $R_i \triangleq \eta(\ell_i)(\mathbf{x}_i)$ for all $i \in \overline{\mathbb{N}}_0$. An example of this construction can be found in Example 3.6. Moreover, if we want to prove that $\{R_i\}_{i=0}^{\infty}$ is a *LocRSM*, its *ϵ-ranking* and *lower bounded* properties should be validated. These are achieved by the notions of so-called *pre-expectation transformers* and *support invariants*.

3.2.2 Pre-Expectation on Stochastic Modules. The pre-expectation transformer of stochastic module is slightly different from that of *pCFG*. The main difference is there can be multiple enabled transitions from a single configuration on a stochastic module. We follow the pre-expectation transformer style of [1] and present the pre-expectation calculas for stochastic modules. Let $\mathcal{A}$ be a stochastic module and η be a (linear) expression map over the location set L of $\mathcal{A}$. Let $\tau = (\ell, st, \ell')$ be a transition enabled on $(\ell, \mathbf{x})$. The *pre-expectation transformer* over η is a function $\mathrm{Pre} : L \times \mathbb{R}^{|V|} \times \Delta \to \mathbb{R}$ defined as follows:

- If $St(\tau)$ is an assignment statement with an update tuple (j, u), we have:
$$\mathrm{Pre}(\ell, \mathbf{x}, \tau) \triangleq \eta(\ell')(\mathbf{x} \text{ with } \mathbf{x}[j] \leftarrow z)$$
where z equals to $u(\mathbf{x})$ if u is a $\mathcal{B}(\mathbb{R})$-measurable function, or equals to $\mathbb{E}(u)$ if u is a distribution.
- If $St(\tau)$ is a probabilistic branching statement, let Δ_p be the set of τ and its complementary transitions,
$$\mathrm{Pre}(\ell, \mathbf{x}, \tau) \triangleq \sum_{\tau'=(\ell, st, \ell'') \in \Delta_p} Pr(\tau') \cdot \eta(\ell'')(\mathbf{x})$$
It is clear that $\mathrm{Pre}(\ell, \mathbf{x}, \tau) = \mathrm{Pre}(\ell, \mathbf{x}, \tau')$ for all $\tau' \in \Delta_p$.
- if $St(\tau)$ is a deterministic or non-deterministic branching statement, then
$$\mathrm{Pre}(\ell, \mathbf{x}, \tau) \triangleq \eta(\ell')(\mathbf{x})$$

We say a transition $\tau = (\ell_i, st, \ell_j)$ is *ϵ-ranked* by η from $(\ell_i, \mathbf{x})$ if $\mathrm{Pre}(\ell_i, \mathbf{x}, \tau) \le \eta(\ell_i)(\mathbf{x}) - \epsilon$. It is the core concept for validating the ϵ-ranking property of a *LocRSM* map, i.e., $\mathbb{E}(R_{n+1} | \mathcal{F}_n) \le R_n - \epsilon \mathbf{1}_{\{\mathrm{L}_n = \ell_a\}}$.

$$\textbf{Skip}\ \frac{\top}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \textbf{Assign}\ \frac{\top}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\textbf{AssignDist}\ \frac{\top}{\{\forall e \in dist.P[e/x]\}\ x := dist\ \{P\}}$$

$$\textbf{Cond}\ \frac{\{P \wedge b\}\ \texttt{skip}\ \{Q\}}{\{P\}\ \texttt{assume}(b)\ \{Q\}}$$

$$\textbf{Prob}\ \frac{\top}{\{P\}\ \texttt{prob(p)}\ \{P\}} \qquad \textbf{NonDt}\ \frac{\top}{\{P\}\ \texttt{*}\ \{P\}}$$

$$\textbf{Conseq}\ \frac{P \to P' \quad \{P'\}\ st\ \{Q'\} \quad Q' \to Q}{\{P\}\ st\ \{Q\}}$$

Figure 2. Probabilistic Floyd-Hoare Logic Rules

3.2.3 Invariant Map on Stochastic Modules. In general, a *LocRSM* $\{R_i\}_{i=0}^{\infty}$ always needs support invariants to guarantee that all of its random variables have a lower bound, i.e., $R_i \ge 0$ for every $i \in \overline{\mathbb{N}}_0$. In the *CFG* of classical programs, Floyd-Hoare logic is engaged to check the inductive property of the invariants. But it is not compatible with probabilistic programs. Some slight modifications should be applied to it. We call the new rules the *probabilistic Floyd-Hoare Logic*, which is defined on the level of the unitary statements of probabilistic programs. Its inference rules are presented in Figure 2. A probabilistic Floyd-Hoare triple $\{P\}$ *st* $\{Q\}$ is *valid* if it can be proved by the inference rules. An *invariant map* for a stochastic module $\mathcal{A}$ is a function $\mathcal{I}$ assigning to each location $\ell \in L$ a predicate such that for every transition $\tau = (\ell, st, \ell') \in \Delta$, the probabilistic Hoare triple $\{\mathcal{I}(\ell)\}\ st\ \{\mathcal{I}(\ell')\}$ is valid. Each $\mathcal{I}(\ell)$ is called the invariant on ℓ. Additionally, if all the predicates of $\mathcal{I}$ are linear predicates, we call $\mathcal{I}$ the *linear invariant map*.

Remark 4. *The rule "**AssignDist**" for probabilistic assignment seems so complicated since it needs reasoning about the first-order logic formula with an universal quantifier. But in general, the pre- and post-conditions of these rules are simple linear invariants to characterizing the boundary of the affine expressions. Thus, the domains of these invariants are always simple convex sets. So, we just need to check the validity of the quantifier-free formulas with the variable x replaced by the minimal or the maximal value in its distribution.*

3.2.4 A.S. Termination of Certified Stochastic Module. Now we have all ingredients to give a formal definition of the certified stochastic modules.

Definition 3.5 (Certified Stochastic Module)**.** A *certified stochastic module* is a tuple $(\mathcal{A}, \eta, \mathcal{I})$, where $\mathcal{A}$ is a stochastic module, η is an expression map and $\mathcal{I}$ is an invariant map over L. We call η the *LocRSM map* and $\mathcal{I}$ the *support invariant* *map*. For all location $\ell \in L$ and all $\mathbf{x} \models \mathcal{I}(\ell)$, the map η has the following properties,

- (**Lower bounded**) We require $\eta(\ell)(\mathbf{x}) \geq 0$.
- (ϵ**-ranking**) All of the transition τ with source location ℓ are ϵ-ranked,
  - if $\ell = \ell_a$, we require $\epsilon > 0$,
  - if $\ell \neq \ell_a$, we require $\epsilon = 0$.

Example 3.6. An example of the certified stochastic module is presented in Figure 3. Both of the *LocRSM* map η and its support invariant map $\mathcal{I}$ are presented in the framed labels (blue and red respectively) near the nodes. The accepting state ℓ_a is the node labeled with ℓ_1. All of the $\eta(\ell)$ are lower bounded by $\mathcal{I}(\ell)$. All transitions outgo from ℓ_1 are 1-ranked and others are 0-ranked. For a certain initial vector together with a scheduler, the stochastic process $R_i \triangleq \eta(\ell_i)(\mathbf{x}_i)$ for all $i \in \overline{\mathbb{N}}_0$ is $R_1 \triangleq \infty, R_2 \triangleq x+3, \cdots$. It is a *LocRSM*.

We have the following theorem to prove that the certified stochastic modules are a.s. terminating.

Theorem 3.7 (A.S. Termination of Certified Stochastic Module)**.** *Let $(\mathcal{A}, \eta, \mathcal{I})$ be a certified stochastic module. For any initial vector $\mathbf{x}_{init} \in \Xi_{init}$ and any scheduler σ, in the filtered probability space $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$, we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$.*

Proof. We first prove that the stochastic process $\{R_i\}_{i=0}^{\infty}$, where $R_i \triangleq \eta(\ell_i)(\mathbf{x}_i)$, is a *LocRSM* for $\mathcal{A}$ w.r.t. $\mathcal{F}_{\mathcal{A}}$. Afterwards this theorem is a direct consequence of Lemma 3.4. It is clear that $\{R_i\}_{i=0}^{\infty}$ is real-valued and $R_i \geq 0$, as well as R_i is $\mathcal{F}_i$-measurable for $i \in \overline{\mathbb{N}}_0$ by definitions. Then, it remains to prove $\mathbb{E}(R_{i+1}|\mathcal{F}_i) \leq R_i - \epsilon \mathbf{1}_{\{\mathbf{L}_i = \ell_a\}}$ for $i \in \overline{\mathbb{N}}_0$. Let Δ_{ℓ_i} be the set of the transitions enabled on ℓ_i by $\mathbf{x} \models \mathcal{I}(\ell_i)$. For $\{\mathbf{L}_i = \ell_a\}$, $\mathbb{E}(R_{i+1}|\mathcal{F}_i) = \mathbb{E}(\eta(\ell_{i+1})(\mathbf{x}_{i+1})|\mathcal{F}_n) \leq \max\{\mathrm{Pre}(\ell_i, \mathbf{x}_i, \tau) \mid \tau \in \Delta_{\ell_i}\} \leq \eta(\ell_i)(\mathbf{x}_i) - \epsilon = R_i - \epsilon$. Meanwhile, for $\{\mathbf{L}_i \neq \ell_a\}$, by the same reason we can get $\mathbb{E}(R_{i+1}|\mathcal{F}_i) \leq \eta(\ell_i)(\mathbf{x}_i) - 0 = R_i$. Thus, $\{R_i\}_{i=0}^{\infty}$ is a ϵ-*LocRSM* for $\mathcal{A}$ with σ and $\mathbf{x}_{init}$. Hence, we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$ by Lemma 3.4. □

4 Decomposition-Based Approach

Section 3 provides all of the theoretic foundations and the probabilistic notions of our decomposition-based approach. In this section, we present the overall technical algorithm in detail, such as constructing and extending the stochastic modules, and synthesizing ϵ-*LocRSM* for stochastic modules.

4.1 Overall Algorithm

The overall algorithm of our decomposition-based approach is presented in Algorithm 1. In the main loop, the *pCFA* $\mathcal{P}$ of the probabilistic program P is decomposed into a set of certified stochastic modules $\{\mathcal{P}_i\}_{i=1}^{n}$. The satisfiability checking of the formula on line 3 is achieved by the Büchi automata operations on $\mathcal{P}$ and $\{\mathcal{P}_i\}_{i=1}^{n}$. If there does not exist

Algorithm 1: Decomposition algorithm.

```
input  : A probabilistic program $P$
output : {$P$ is a.s terminating, unknown}
1  $n \leftarrow 0$
2  $\mathcal{P} \leftarrow buildpCFA(P)$
3  while $\exists \rho.\ \rho \in \mathcal{L}(\mathcal{P}) \setminus \bigcup_{i=1}^{n} \mathcal{L}(\mathcal{P}_i)$ do
4      $\rho_c \leftarrow complementTrace(\mathcal{P}, \rho)$
5      $(\mathcal{A}_n, \eta_n, \mathcal{I}_n) \leftarrow buildCertifiedModule(\mathcal{A}(\rho_c))$
6      if $(\mathcal{A}_n, \eta_n, \mathcal{I}_n)$ exists then
7          $\mathcal{P}_n \leftarrow extendCertifiedModule(\mathcal{A}_n, \eta_n, \mathcal{I}_n)$
8          $n \leftarrow n + 1$
9      else return unknown
10 return $P$ is a.s terminating
```

an ω-statement trace ρ belongs to $\mathcal{L}(\mathcal{P}) \setminus \bigcup_{i=1}^{n} \mathcal{L}(\mathcal{P}_i)$, the a.s. termination of P is proved. Otherwise, we get a probability complete ω-statement trace ρ_c from $\mathcal{P}$. The trace ρ_c is also lasso-shape. We first try to prove the a.s. termination of the lasso module $\mathcal{A}(\rho_c)$ by synthesizing a *LocRSM* for it. If the synthesis is successful, the lasso module $\mathcal{A}_n$ is extended to a general stochastic module $\mathcal{P}_n$. Otherwise, the algorithm returns *unknown.* [1] The synthesis algorithm is polynomial-time [1]. On average, we need synthesize $O(n)$ *LocRSMs*, where n is the number of the branches in P. Thus, the overall algorithm is also polynomial-time in expectation. But we can sample multiple different ω-traces and synthesize the certified stochastic modules simultaneously. The natural parallelization can greatly speed up our algorithms for large-scale probabilistic programs. The details of the algorithm are presented in the following.

4.2 Constructing Stochastic Modules

On line 2 of Algorithm 1, we build a *pCFA* $\mathcal{P}$ from the probabilistic program P. Again, we emphasize that $\mathcal{P}$ is also a Büchi automaton such that all of the locations of $\mathcal{P}$, expect the terminating location ℓ_{term}, are accepting states. Thus, all of the non-terminating traces are accepted by $\mathcal{P}$.

As we mentioned before, a stochastic module is a *pCFA* built from a lasso-shape ω-statement trace ρ. It has exact one accepting state ℓ_a. In our algorithm, ρ is sampled from the Büchi automaton of $\mathcal{L}(\mathcal{P}) \setminus \bigcup_{i=1}^{n} \mathcal{L}(\mathcal{P}_i)$. However, the *pCFA* built from ρ may not satisfy the probability completeness requirement, i.e., ρ can only go through one of the probabilistic branching statements but never go through its counterpart. Such a sample trace ρ have no information about another branch of the whole probabilistic branching statement. Thus, we should complement ρ by simulating it on the *pCFA* and

[1] Actually, before return *unknown*, we can try to refute a.s. termination of ρ_c by the notion of the *stochastic invariants* and the *repulsing supermartingales* [8]. Refuting a.s. termination of ρ_c should also be simpler than the entire probabilistic programs.

when we meet a probabilistic transition without complementary transition, we add its complementary transition and the following transitions to ρ. After that, we can get a complemented ω-trace ρ_c. On line 4 of Algorithm 1, we complement ρ on $\mathcal{P}$ and get a probability complete ω-statement trace ρ_c. On line 5, we build a stochastic module $\mathcal{A}(\rho_c)$.

Example 4.1. Consider the *pCFA* $\mathcal{P}$ in Figure 1, let ρ_s = $\boxed{x \geq 0}$ $\boxed{y \geq 0}$ represent the sequential guards. An ω-trace sampled from $\mathcal{P}$ could be $\rho = \rho_s\ (\rho_s\ \boxed{*}\ \boxed{prob(0.5)}\ \boxed{x := x - 2}\ )^\omega$. It is obviously that ρ is not a probability complete trace. Thus, we can simulate ρ on $\mathcal{P}$ and get a new ω-trace $\rho_c = \rho_s\ (\rho_s\ \boxed{*}\ [\ (\ \boxed{prob(0.5)}\ \boxed{x := x - 2}\ )\ |\ (\ \boxed{prob(0.5)}\ \boxed{skip}\ )\ ]\ )^\omega$. The stochastic module $\mathcal{A}_1$ constructed from ρ_c is presented in Figure 3 (The colorized labels with gray frame can be ignored). The accepting state ℓ_a of $\mathcal{A}_1$ is ℓ_1.

4.3 Building Certified Stochastic Modules

On line 5 of Algorithm 1, a certified stochastic module ($\mathcal{A}_n$, $\eta_n, \mathcal{I}_n$) is built. Since $\mathcal{A}_n$ is a lasso module, synthesizing a.s. terminating arguments for it should be rather easy. In this step, we synthesize a ϵ-*LocRSM* map η_n and its support invariant map $\mathcal{I}_n$ for $\mathcal{A}_n$. These tasks can be accomplished by the standard linear constraints based synthesis method using Farkas's Lemma. The details of these methods can be found in [1, 5, 10, 27]. Nevertheless, the notion of LocRSMs is little different to that of RSMs. We only require the expectation value of LocRSM to strictly decrease when the accepting state is visited. Specifically, we use the linear templates to express LocRSM maps and invariant maps. The constraints of Definition 3.5 can be encoded as following formulas.

$$\forall \mathbf{x}, \ell . \mathcal{I}(\ell)(\mathbf{x}) \rightarrow \eta(\ell)(\mathbf{x}) \geq 0 \tag{1}$$

$$\forall \mathbf{x}, \tau . \mathcal{I}(\ell_a)(\mathbf{x}) \rightarrow \mathrm{Pre}(\ell_a, \mathbf{x}, \tau) \leq \eta(\ell_a)(\mathbf{x}) - \epsilon \tag{2}$$

$$\forall \mathbf{x}, \ell, \tau . \mathcal{I}(\ell)(\mathbf{x}) \rightarrow \mathrm{Pre}(\ell, \mathbf{x}, \tau) \leq \eta(\ell)(\mathbf{x}) \tag{3}$$

The universal quantifiers on $\mathbf{x}$ can be eliminated by Farkas's Lemma. Finally, we can get all of the parameters of *LocRSM* expressions by constraint solving.

Example 4.2. The certified stochastic module built from the lasso module $\mathcal{A}_1$ in Example 4.1 is presented in Figure 3. The colorized labels near each node ℓ are the *LocRSM* expression $\eta(\ell)$ and the invariant $\mathcal{I}(\ell)$ respectively. In this example, the linear template for synthesizing *LocRSM* expression on each location ℓ_i is $a_i x + b_i y$. Let ϵ be 1. First, the expressions on all locations should satisfy the lower bounded property (1), e.g., we have $\forall x, y. \mathcal{I}(\ell_1)(x, y) \rightarrow a_1 x + b_1 y \geq 0$ on ℓ_1. Second, all transitions outgoing from the accepting state ℓ_1 should be 1-ranked (2), e.g., for the transition $(\ell_1, x \geq 0, \ell_{1'})$, we have $\forall x, y. \mathcal{I}(\ell_1)(x, y) \rightarrow a_{1'} x + b_{1'} y \leq a_1 x + b_1 y - 1$. Moreover, the transitions outgoing from other states can be 0-ranked (3), e.g., for the probabilistic branching transition outgoing from ℓ_3, we have $\forall x, y. \mathcal{I}(\ell_3)(x, y) \rightarrow 0.5 \cdot (a_4 x + b_4 y) + 0.5 \cdot (a_5 x + b_5 y) \leq a_3 x + b_3 y$. By applying Farkas's Lemma and solving these constraints, we can get the *LocRSM* map in Figure 3.

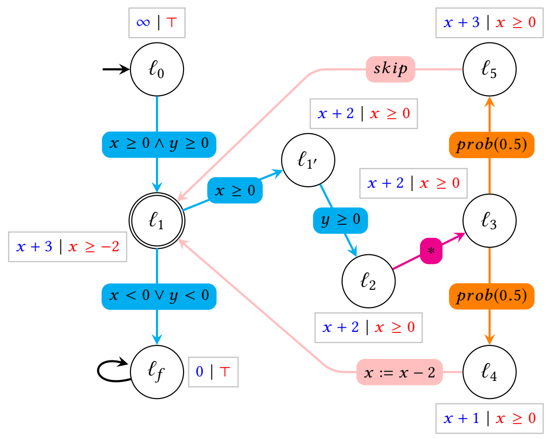

Figure 3. Certified Stochastic Module ($\mathcal{A}_1$)

In addition, it is important to consider a special case separately in order to improve the synthesis efficiency. Recall that an ω-statement trace ρ represents a set of runs beginning with some initial vectors. But the set can be empty, i.e., the run is blocked on a configuration. For example, the statement trace ρ with a prefix $\boxed{x = uniform(0, 1)}$ $\boxed{x < 0}$ is blocked after the statement $\boxed{x < 0}$. Given a statement trace ρ, if there exists an integer $k \in \mathbb{N}$ such that the probability of reaching the configuration $(\ell_k, \mathbf{x})$ is zero, we say ρ is (*almost-surely*) *infeasible*. In our algorithm, let ρ_c be the form of $\rho_s . \rho_l{}^\omega$ where ρ_s is the *stem* and ρ_l is the *loop*. Suppose ρ_s is a.s. infeasible, for every location ℓ in ρ_l, its invariant $\mathcal{I}(\ell)$ should be $\perp$. As a result, arbitrary expression map η can satisfy the requirements in Definition 3.3. Thus, we should check the feasibility of ρ_s before synthesis. Furthermore, we can also check the feasibility of $\rho_s . \rho_l$, which can be achieved by solving the constraints of the path formulas. The skills for checking the validity of the probabilistic Floyd-Hoare triples with universal quantifiers can also be used here. We call the certified stochastic module of this case the *trivial module*. Otherwise, we call it the *non-trivial module*.

4.4 Extending Certified Stochastic Modules

On line 7 of Algorithm 1, we extend the lasso stochastic module $\mathcal{A}_n$ to a general stochastic module $\mathcal{P}_n$ such that $\mathcal{P}_n$ also has the same a.s terminating arguments $(\eta_n, \mathcal{I}_n)$. The generalization is a very important technique to improve the efficiency of our algorithm. We can rule out the terminating runs as many as possible by generalizing a lasso module. To achieve this, we propose two generalization rules to modify $\mathcal{A}_n$ as follows:

Rule 1: Merge Locations. If there exists a transition $\tau = (\ell_i, st, \ell_j)$ in $\mathcal{A}_n$ such that $\eta_n(\ell_i) = \eta_n(\ell_j)$ and $\mathcal{I}_n(\ell_i) = \mathcal{I}_n(\ell_j)$,

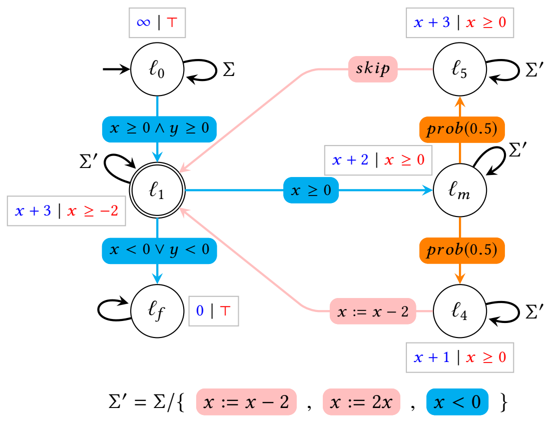

Figure 4. Extended Certified Stochastic Module ($\mathcal{P}_1$)

then we can merge the nodes of ℓ_i and ℓ_j. Besides, the transition τ should also be transformed to a self-loop transition on the merged location.

Rule 2: Add Transitions. Let st be an unitary statement, and ℓ_i and ℓ_j be the locations of $\mathcal{A}_n$. Let ℓ_a be the accepting state of $\mathcal{A}_n$. We can add a transition $\tau = (\ell_i, st, \ell_j)$ to $\mathcal{A}_n$ if the probabilistic Hoare triple $\{\mathcal{I}(\ell_i)\}\ st\ \{\mathcal{I}(\ell_j)\}$ is valid and τ is ϵ-ranked by η from $(\ell_i, \mathbf{x})$ for all $\mathbf{x} \models \mathcal{I}(\ell_i)$,

- when $\ell_i = \ell_a$, we require $\epsilon > 0$,
- when $\ell_i \neq \ell_a$, we require $\epsilon = 0$.

Moreover, if st is a probabilistic branching statement, we should also add the corresponding complementary transition after τ is added.

The above rules modify the control-flow structure of the $\mathcal{A}_n$ while maintaining the properties of certified stochastic modules required in Definition 3.5. After the modifications, the extended stochastic module $\mathcal{P}_n$ accepts more traces than the lasso module $\mathcal{A}_n$ does. We have $\mathcal{L}(\mathcal{A}_n) \subseteq \mathcal{L}(\mathcal{P}_n)$.

Example 4.3 (Extending Certified Stochastic Module)**.** The extended certified stochastic module $\mathcal{P}_1$ is presented in Figure 4. Compared to the certified stochastic module in Figure 3, the locations ℓ_1, $\ell_{1'}$ and ℓ_3 are merged into a single location ℓ_m. The transitions from ℓ_1 to $\ell_{1'}$ and from $\ell_{1'}$ to ℓ_2 are transformed to the self-loop transitions on ℓ_m. Besides, some transitions which do not affect the variable x are trivially added to some locations. Finally, the languages recognized by $\mathcal{P}_1$ is $\mathcal{L}(\mathcal{P}_1) = \Sigma^* (x \geq 0\ \Sigma'^* [(prob(0.5)\ \Sigma'^*\ x := x - 2) \mid (prob(0.5)\ \Sigma'^*\ skip)])^\omega$. Let $\mathcal{T}_1$ be the first trace set in Example 4.1. It is clear that $\mathcal{T}_1 \subseteq \mathcal{L}(\mathcal{P}_1)$. We can also find another certified stochastic module $\mathcal{P}_2$ by our algorithm such that $\mathcal{T}_2 \subseteq \mathcal{L}(\mathcal{P}_2)$.

5 Soundness and Completeness

5.1 Soundness

We employ Theorem 5.2 to ensure the soundness of our decomposition approach. To prove this theorem, we first prove the following lemma.

Lemma 5.1. *Given a pCFA $\mathcal{P}$ together with an initial valuation $\mathbf{x}_{init} \in \Xi_{init}$ and a scheduler σ, we have a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i=0}^{\infty}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$. For any $\{st_j\}$, we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\bigcap_{i=0}^{k}\{\mathsf{S}_i \rhd st_j\}) = \prod_{i=0}^{k}\mathbb{P}_{\sigma_i}(\{\mathsf{S}_i \rhd st_j\})$, where $k \in \overline{\mathbb{N}}_0$ and $\rhd \in \{=, \neq\}$.*

Proof. The measurable space $(\Omega, \mathcal{F})$ can be considered as the *cylinder construction* on the production of the measurable spaces $(\Omega_{\sigma_i}, \mathcal{F}_{\sigma_i})$ of the scheduler on i-th steps. Let $A_i \triangleq \{\mathsf{S}_i \rhd st_j\}$. It is clear that A_i is $\mathcal{F}_{\sigma_i}$-measurable. Then our goal is a direct consequence of Corollary 2.7.3 in [2]. □

Theorem 5.2 (Soundness)**.** *Let P be a probabilistic program and $\mathcal{P}$ the pCFA of P. If $\mathcal{P}$ can be decomposed into a finite set of a.s. terminating stochastic modules $\{\mathcal{P}_i\}_{i=1}^{n}$ such that $\mathcal{L}(\mathcal{P}) \subseteq \bigcup_{i=1}^{n}\mathcal{L}(\mathcal{P}_i)$, then P is a.s. terminating.*

Proof. We can associate $\mathcal{P}$ with arbitrary scheduler σ and an initial vector $\mathbf{x}_{init} \in \Xi_{init}$, then there is a probability space $(\Omega, \mathcal{F}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$. By Definition 2.5, P is a.s. terminating if we can prove $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$. Associate each $\mathcal{P}_i$ with the same σ and $\mathbf{x}_{init}$, we get a sequence of probability spaces $\{(\Omega^i, \mathcal{F}^i, \mathbb{P}^i)\}_{i=1}^{n}$ and we have $\mathbb{P}_i(\{\pi \mid Tm(\pi) = \infty\}) = 0$ by the definition of a.s. termination. However, $(\Omega, \mathcal{F}, \mathbb{P}^{\sigma}_{\mathbf{x}_{init}})$ and $\{(\Omega^i, \mathcal{F}^i, \mathbb{P}^i)\}_{i=1}^{n}$ are different probability space. So, our goal $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$ is not a direct consequence of $\mathbb{P}^i(\{\pi \mid Tm(\pi) = \infty\}) = 0$ for $0 \leq i \leq n$. Although we can achieve it by constructing a uniform probability space such that each $\mathbb{P}_i$ is a conditional probability in the uniform probability space, this should be somewhat intricate and technical. We prove this by contradiction. Given a unitary statement st, the set $\{\mathsf{S}_j = st\}$ is $\mathcal{F}$-measurable. Moreover, given a run π_Σ, the set $\lim_k \bigcap_{j=0}^{k}\{\mathsf{S}_j = \mathsf{S}_j(\pi_\Sigma)\}$ is also $\mathcal{F}$-measurable. Because the initial vector $\mathbf{x}_{init}$ has been uniquely determined, there is a one-to-one correspondence between the sets of ρ and π_Σ, denoted as f. We have $\lim_k \bigcup_{\rho \in \mathcal{L}(\mathcal{P}_i)} \bigcap_{j=0}^{k}\{\mathsf{S}_j = \mathsf{S}_j(f(\rho))\} = \Omega_{|i} \subseteq \Omega^i$ and we can prove $\Omega_{|i}$ is both $\mathcal{F}$- and $\mathcal{F}^i$-measurable. Since $\mathcal{L}(\mathcal{P}) \subseteq \bigcup_{i=1}^{n}\mathcal{L}(\mathcal{P}_i)$, we can partition Ω into n pairwise disjoint subset $\{\Omega_{|i}\}_{i=1}^{n}$ such that all of them are $\mathcal{F}$-measurable. Let $A^i = \{\pi \mid Tm(\pi) = \infty\} \cap \Omega_{|i}$. It is clear that A^i is $\mathcal{F}$-measurable. Suppose $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) > 0$, since $\{\pi \mid Tm(\pi) = \infty\} = \bigcup_{i=1}^{n} A^i$ in Ω, then there must be some i^* such that $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(A^{i^*}) > 0$. Because $\mathcal{P}$ and $\mathcal{P}_{i^*}$ are associated with the same σ and $\mathbf{x}_{init}$, let $B^i = \bigcup_{\rho \in \mathcal{L}(\mathcal{P}_i)} \{\mathsf{S}_j = \mathsf{S}_j(f(\rho))\}$. Then by Lemma 5.1 we have $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(A^{i^*}) = \mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\lim_k \bigcap_{j=0}^{k}\{\mathsf{S}_k \neq st_{id}\} \cap B^{i^*}) = \lim_k \mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\bigcap_{j=0}^{k}\{\mathsf{S}_j \neq st_{id}\} \cap B^{i^*}) = \lim_k \prod_{j=0}^{k}\mathbb{P}_{\sigma_j}(\{\mathsf{S}_j \neq st_{id}\} \cap B^{i^*}) = \mathbb{P}^{i^*}(\lim_k \bigcap_{j=0}^{k}$

$\{S_j \neq st_{id}\} \cap B^{i^*}) \leq \mathbb{P}^{i^*}(\{\pi \mid Tm(\pi) = \infty\}) = 0$. Then we get a contradiction and thus $\mathbb{P}^{\sigma}_{\mathbf{x}_{init}}(\{\pi \mid Tm(\pi) = \infty\}) = 0$. Hence, P is a.s. terminating. □

5.2 Relative Completeness

There is no result that an a.s. terminating probabilistic program, real-valued and with non-determinism, has a ranking supermartingale. Actually, all of the existing methods for this problem are incomplete. At this point, we only prove the relative completeness of our approach, i.e., if P has a *LexRSM*, it can be proved to be a.s terminating by our approach. Moreover, Example 2.1 and the cases in Section 6.1 show that our approach is strict stronger than the *LexRSM*-based method. The *LexRSM*-based method for proving a.s. termination is proposed by Agrawal et al. [1]. We first present the basic concepts of it.

Definition 5.3 (Lexicographic Ranking Supermartingale Map)**.** Let $\epsilon > 0$. An n-dimensional lexicographic ϵ-ranking supermartingale map (ϵ-*LexRSM* map) for a *pCFA* $\mathcal{P}$ supported by an invariant map $\mathcal{I}$ is an n-dimensional measurable map $\vec{\eta} = (\eta_1, \cdots, \eta_n)$ on $\mathcal{P}$ such that for each configuration $(\ell, \mathbf{x})$ where $\ell \neq \ell_{term}$ and $\mathbf{x} \models \mathcal{I}(\ell)$ the following conditions are satisfied:

- (**Lower bounded**) for all $1 \leq j \leq n$, $\eta_j(\ell)(\mathbf{x}) \geq 0$.
- for each transition τ enabled on $(\ell, \mathbf{x})$, there exists $1 \leq j \leq n$ such that
  - (ϵ-**Ranking**) τ is ϵ-ranked by η_j from $(\ell, \mathbf{x})$.
  - (**Unaffecting**) for all $1 \leq j' < j$ we have that τ is 0-ranked by $\eta_{j'}$ from $(\ell, \mathbf{x})$.

Theorem 5.4 ([1] Theorem 4.8)**.** *Let $\mathcal{P}$ be the pCFA of a probabilistic program. Assume that there exists an $\epsilon > 0$ and an n-dimensional ϵ-LexRSM map $\vec{\eta} = (\eta_1, \ldots, \eta_n)$ for $\mathcal{P}$ supported by an invariant map $\mathcal{I}$. Then $\mathcal{P}$ terminates almost-surely.*

Given a *pCFA* $\mathcal{P}$ with an n-dimensional ϵ-*LexRSM* map $\vec{\eta}$, the transition set $|\Delta|$ is finite. For a transition $\tau \in \Delta$, let $(\ell, \mathbf{x}) \models \mathcal{I}(\ell)$ where ℓ is the source location of τ. We say τ has *level* j if τ is ϵ-ranked by η_j from $(\ell, \mathbf{x})$ and 0-ranked by $\eta_{j'}$ from $(\ell, \mathbf{x})$ for all $1 \leq j' < j$. The minimal level of τ is denoted as $lev(\tau)$. We can group the transition set Δ into n subsets such that $\Delta_i \triangleq \{\tau \mid lev(\tau) = i\}$ for $1 \leq i \leq n$. In the following part, we do not distinguish between the transition τ and the statement $St(\tau)$ labeled on τ.

Theorem 5.5 (Relative Completeness to *LexRSM* [2])**.** *Let $\mathcal{P}$ be the pCFA of a probabilistic program. If $\mathcal{P}$ can be proved to be a.s. terminating by Theorem 5.4, then $\mathcal{P}$ can also be decomposed into n certified stochastic modules.*

Proof. Let $\vec{\eta} = (\eta_1, \ldots, \eta_n)$ be the ϵ-*LexRSM* map and $\mathcal{I}$ its support invariant map for $\mathcal{P}$. Let $\mathcal{T}(\mathcal{P})$ be the set of all traces in $\mathcal{P}$. It is obviously that $\mathcal{T}(\mathcal{P}) \subseteq \Delta^{\omega}$. Let $\mathcal{T}_i \triangleq \mathcal{T} \cap ((\Delta_{i+1} \mid \cdots \mid \Delta_n)^*.\Delta_i)^{\omega}$. Because $\Delta = \bigcup_{i=1}^{n} \Delta_i$, we have $\bigcup_{i=1}^{n} \mathcal{T}_i = \mathcal{T} \cap \Delta^{\omega} = \mathcal{T}$. Thus, $\mathcal{T}$ can be decomposed into n subsets $\mathcal{T}_i$ for $1 \leq i \leq n$. Next, we show that each trace set $\mathcal{T}_i$ can be transformed into a certified stochastic module. It is clear that each $\mathcal{T}_i$ is an ω-regular languages over the alphabet Σ and there exists a stochastic module $\mathcal{A}_i$ recognizes $\mathcal{T}_i$, i.e., $\mathcal{T}_i = \mathcal{L}(\mathcal{A}_i)$. We notice that each η_i and $\mathcal{I}$ are well-defined on the location set L_i of each $\mathcal{A}_i$. The projections of η_i and $\mathcal{I}$ to L_i are η'_i and $\mathcal{I}_i$. Then, by Definition 3.5 and Definition 5.3, we can find that each η'_i is a ϵ-*LocRSM* map for $\mathcal{A}_i$ supported by the invariant map $\mathcal{I}_i$. Then $(\mathcal{A}_i, \eta'_i, \mathcal{I}_i)$ is a certified stochastic module. Hence, $\mathcal{P}$ can be decomposed into n certified stochastic modules. □

[2]The proof of this theorem can be naturally extended to the specialized form for classical programs.

6 Evaluations

In this section, we show the efficiency of our approach by some hard cases and the benchmarks from [1].

6.1 Case Studies

Case 1. The probabilistic program of this case is presented in Case 1, where the variables are integer-valued. It is a.s terminating but also have no *LexRSM*. The basic traces in this case are:

$\rho_0 =$ $0 \leq id \wedge id \leq max$ $tmp := id + uniform\{0, 1\}$

$\rho_1 =$ $tmp \neq id$ $tmp \leq max$ $tmp := tmp + uniform\{0, 1\}$

$\rho_2 =$ $tmp \neq id$ $tmp > max$ $tmp := 0$

Then we have $\mathcal{L}(\mathcal{P}) = \rho_0(\rho_1|\rho_2)^{\omega}$. Suppose we first get an ω-trace $\rho_{c1} = \rho_0\rho_1^{\omega}$, the stochastic module $\mathcal{A}(\rho_{c1})$ can be proved to be a.s. terminating by $\eta(\ell_a) = tmp$ with the support invariant $\mathcal{I}(\ell_a) = tmp \geq 0$. Then the corresponding certified module can be extended to a general module such that $\mathcal{L}(\mathcal{P}_1) = (\rho_0|\rho_1|\rho_2)^*{\rho_1}^{\omega}$. We can find another certified stochastic module $\mathcal{L}(\mathcal{P}_2) = \rho_0\rho_1^*\rho_2({\rho_1}^*\rho_2)^{\omega}$. It is a trivial module. Then we have $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\mathcal{P}_1) \cup \mathcal{L}(\mathcal{P}_2)$. As a result, the a.s. termination of Case 1 is proved.

Case 2. The probabilistic program of this case is presented in Case 2. There is a nested loop on line 3. If the branching condition of line 5 is "**`prob(0.5)`**", there should be a 2-dimensional *LexRSM* for this program. Actually the program is a.s terminating but has no *LexRSM*. That is because two branches of the nested loop are executed in the same number of times and the lexicographic method is not aware of this. There are three basic traces in this case:

$\rho_s =$ $i \geq 0 \wedge n > 0$ $j := 2 * n$

$\rho_1 = \rho_s$ $j \geq 0$ $j := j - n$ $j \geq n$ $i := i - 1$ $i := i - uniform(0, 1)$

$\rho_2 = \rho_s$ $j \geq 0$ $j := j - n$ $j < n$ $i := i + 1$ $i := i - uniform(0, 1)$

$\rho_3 = \rho_s$ $j < 0$ $i := i - uniform(0, 1)$

We have $\mathcal{L}(\mathcal{P}) = (\rho_1|\rho_2|\rho_3)^{\omega}$. We can build the first certified module $(\mathcal{P}_1, \eta, \mathcal{I})$ such that $\mathcal{L}(\mathcal{P}_1) = (\rho_1\rho_2)^{\omega}$, $\eta(\ell_a) = i + 1$

Listing 1. Case 1

```
1 if 0 ≤ id and id ≤ max then
2   tmp := id + uniform{0, 1, 2};
3   while tmp ≠ id do
4     if tmp ≤ max then
5       tmp := tmp + uniform{0, 1};
6     else tmp := 0; fi
7   od
8 else skip;
9 fi
```

Listing 2. Case 2

```
1 while i ≥ 0 and n > 0 do
2   j := 2 * n;
3   while j ≥ 0 do
4     j := j - n;
5     if j ≥ n then i := i + 1;
6     else i := i - 1; fi
7   od
8   i := i - uniform(0, 1);
9 od
```

Listing 3. Case 3

```
1 while x ≥ 4  and y ≥ 0 do
2   if * then
3     c := uniform(0, x);
4     if c ≥ 1 then x := x - 1;
5     else x := x + 1; fi
6   else
7     x := *; y := y - 1;
8   fi
9 od
```

and $\mathcal{I}(\ell_a) = i \geq -1$. The traces in $\mathcal{L}(\mathcal{P}) - \mathcal{L}(\mathcal{P}_1)$ can be proved to be a.s. terminating by several trivial modules.

Case 3. Case 3 is a variant of Example 2.1. The probabilistic program in this case is also a.s. terminating but the *LexRSM* can not be synthesized. Although the *unaffecting* property can be satisfied, the *ϵ-ranking* property of the first branch can not be validated by the synthesizing method. That is because the synthesizing method becomes over-approximate when some probabilistic variables get involved in the guard predicate, e.g. the variable c. The invariant about the variable c on line 4 is $0 \leq c \leq x$ and both the guards of following two branches can be satisfied. Then the synthesizing method requires both of these two transitions are ϵ-ranked with $\epsilon > 0$. It is obviously impossible and thus the synthesis fails. However, we can calculate the expectation value of the candidate supermartingale expression x. Though non-linear arithmetics are required, we can find that it decreases by at least 1/3. So, we can still prove the a.s. termination of this case by our decomposition-based approach. Because in Theorem 5.2, we do not require any specific method to prove a.s. termination of stochastic modules. We can decompose $\mathcal{P}$ as Example 4.1 and then prove a.s. termination of the stochastic module $\mathcal{P}_1$ by the method of [23], in which the weakest pre-condition style expectation-transformer can handle the guards with probabilistic variables.

6.2 On Benchmarks

We present the experimental results on the available benchmarks from [1]. Most of these benchmarks can be proved to be a.s. terminating by a n-dimensional *LexRSM*, where $2 \leq n \leq 3$. We evaluate our approach on these benchmarks with the help of Büchi Automizer [15, 18] and make comparison to *LexRSM*-based method [1]. The results are presented in Table 1. The column "#*D*." represents the numbers of the dimensions of the *LexRSM*s. The column "#*T*." and column "#*N*." are the numbers of the *trivial module*s and the *non-trivial module*s respectively used in our method. The right most column are the sets of the *LocRSM* expressions on $\eta(\ell_a)$ of the non-trivial modules. Some non-trivial modules have the same $\eta(\ell_a)$. As we can see, our approach fails on just 1 benchmark while the *LexRSM*-based method on 5. For most

Table 1. Our approach v.s. *LexRSM*-based method

Benchmark	#D.	#T.	#N.	$\{\eta(\ell_a)\}$ for each $\mathcal{P}_i \in N$.
alain	3	3	3	$\{n_1, n_2\}$
catmouse	2	1	2	$\{-m + x, 2m - 2x + 1\}$
counterex1a	-	**2**	**10**	$\{-2y + 2n + 1, x, 2y + 1\}$
counterex1c	2	2	8	$\{-2y + 2n + 1, y, x, 2y + 1\}$
easy1	1	0	1	$\{-2x + 79\}$
exmini	2	0	1	$\{-i - j + k + 101\}$
insertsort	3	2	0	$\emptyset$
ndecr	2	0	1	$\{i\}$
perfect	3	2	3	$\{2y_2 - 2y_1 + 1, y_1\}$
perfect2	3	1	4	$\{4y_2 - 4y_1 + 1, y_2 - y_1 + 1, \cdots\}$
real2	-	-	-	-
realbubble	3	2	3	$\{i, -j + i\}$
realselect	3	3	0	$\emptyset$
realshellsort	-	**4**	**1**	$\{i\}$
serpent	3	3	5	$\{2x + 1, 2y + 1, -2y + 2n + 1\}$
sipmabubble	3	1	3	$\{2i + 1, i - j\}$
speedDis2	-	**0**	**3**	$\{-2z + 2x + 1, n - x\}$
speedN*iple	3	1	2	$\{m - y, n - x\}$
speedpldi2	2	1	1	$\{v_1\}$
speedpldi4	2	1	4	$\{i, i - m\}$
speedS*eDep	-	**2**	**4**	$\{m - y, n - x\}$
speedS*gle2	2	0	2	$\{m - y, n - x\}$
unperfect	2	25	12	$\{20y_2 - 20y_1 + 9, y_2 - y_1 + 1, \cdots\}$
unperfect★	2	4	3	$\{-2y_1 + 2y_2 + 1, y_1\}$
wcet1	2	1	1	$\{i\}$
while2	3	2	3	$\{i, j\}$

of the benchmarks, the numbers of non-trivial modules are closed to the dimensions of the *LexRSM*s. For some benchmarks, our approach just needs no more than one non-trivial module. All above proves the significant efficiency of our approach. Nevertheless, there are fewer benchmarks that our method needs much more non-trivial modules. The "unperfect" case needs reasoning on disjunctive invariant but our invariant synthesis is weak at this. But if we add the invariant $(y_1 \geq 2 \vee y_1 \leq 0)$ to its source code, i.e., "unperfect★", our approach can resolve it efficiently. For the "real2" case, we find that it is not an a.s. terminating program.

7 Related Work

Supermatingale-Based Approaches. This kind of methods was first presented in [5]. It synthesizes a global ranking supermartingale and is applicable to a.s termination of simple probabilistic programs with real-valued variables but without non-determinism. Chatterjee et al. [6] presented a method to synthesize polynomial ranking supermartingales for non-deterministic probabilistic programs with polynomial guards and assignments. Huang et al. [19] proposed new approaches for the a.s. termination problem based supermartingales with lower bounds on conditional absolute difference and Central Limit Theorem. Chatterjee et al. [8] proposed the *stochastic invariant*s and *repulsing supermartingale*s for quantitative termination and reachability problems. In [1, 7, 13, 20, 23], the ranking supermatingale-based approach is extended with non-determinism. The approach of [13] can handle both non-determinism and continuous probability distributions over transitions. It required the uniform integrability condition but was still shown to be somewhat unsound in [20]. Huang et al. [20] strengthened the compositional approach in [13] to a sound approach by the notion of *descent supermartingale map*. It required the so-called *strict decrease* condition but did not require non-negativity of supermartingales. The work [7] is about the termination for probabilistic and non-deterministic recursive programs. The so-called *conditionally difference-bounded ranking supermartingales* was used to prove a.s. termination in this approach. Agrawal et al. [1] extended the ranking supermartingale approach with lexicographic orderings. McIver et al. [23] presented the "parametric" super-martingale methods based on the *expectation-transformer* and weakest precondition reasoning. The most related works on a.s termination are [1, 13, 20]. All of these approaches can be applied to the probabilistic programs with complex control flow. However, all of them require the lexicographic order or the *unaffecting* condition and can not rule out the "non-terminating" cases resulted from the infeasible paths.

Other Methods for A.S. Termination. Besides the ranking supermatingale-based approaches, there are other methods to prove a.s. termination. [12] presented a sound and complete method for proving termination of finite-state probabilistic programs. In [4, 14], *Lyapunov ranking function*s provided a sound and complete method for proving positive termination of probabilistic programs with countable state space and without non-determinism. In [22, 24] proposed a bounded martingale-based method to prove a.s. termination of probabilistic programs with non-determinism but restricted to discrete probabilistic choices. There are also some proof rule-based approaches for a.s. termination analysis, such as [21] for positive termination and [26] for recursive probabilistic programs.

Decomposition-Based Methods. This kind of approach to termination of classical programs was proposed in [18]. Chen et al. [9] presented an advanced version. The basic idea of this method is the so-called *trace abstraction*, which was presented in [16, 17]. All of these methods are not applicable to the a.s terminating problem of probabilistic programs.

8 Conclusion and Future Work

In this work, we present a decomposition-based approach to prove a.s. termination of probabilistic programs. Our approach is based on the notions of stochastic modules and *LocRSM*. We prove the soundness and the relative completeness of our approach. The evaluation on the benchmarks of previous works shows the significant efficiency of our approach. Furthermore, our approach can also be combined with the a.s. termination refuting algorithm and make the refutation more efficient. It can also be used to analysis expected termination time of probabilistic programs.

Acknowledgments

This work was partially funded by the National Key R&D Program of China (No. 2018YFB1308601), the NSF of China (No. 61672310 and No. 61527812), and the Guangdong Science and Technology Department (No. 2018B010107004).

References

[1] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2017. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 34.

[2] Robert B Ash, B Robert, Catherine A Doleans-Dade, and A Catherine. 2000. *Probability and measure theory.* Academic Press.

[3] Dirk Beyer, M Erkan Keremoglu, and Philipp Wendler. 2010. Predicate abstraction with adjustable-block encoding. In *Formal Methods in Computer Aided Design.* IEEE, 189–197.

[4] Olivier Bournez and Florent Garnier. 2005. Proving positive almost-sure termination. In *International Conference on Rewriting Techniques and Applications.* Springer, 323–337.

[5] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic program analysis with martingales. In *International Conference on Computer Aided Verification.* Springer, 511–526.

[6] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination analysis of probabilistic programs through Positivstellensatz's. In *International Conference on Computer Aided Verification.* Springer, 3–22.

[7] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 327–342.

[8] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017. Stochastic invariants for probabilistic termination. In *ACM SIGPLAN Notices*, Vol. 52. ACM, 145–160.

[9] Yu-Fang Chen, Matthias Heizmann, Ondřej Lengál, Yong Li, Ming-Hsien Tsai, Andrea Turrini, and Lijun Zhang. 2018. Advanced automata-based algorithms for program termination checking. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 135–150.

[10] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma. 2003. Linear invariant generation using non-linear constraint solving. In *International Conference on Computer Aided Verification.* Springer, 420–432.

[11] Michael A Colón and Henny B Sipma. 2002. Practical methods for proving program termination. In *International Conference on Computer Aided Verification.* Springer, 442–454.

[12] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. 2012. Proving termination of probabilistic programs using patterns. In *International Conference on Computer Aided Verification*. Springer, 123–138.

[13] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic termination: Soundness, completeness, and compositionality. *ACM SIGPLAN Notices* 50, 1 (2015), 489–501.

[14] Frederic G Foster et al. 1953. On the stochastic matrices associated with certain queuing processes. *The Annals of Mathematical Statistics* 24, 3 (1953), 355–360.

[15] Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Evren Ermis, Jochen Hoenicke, Markus Lindenmann, Alexander Nutz, Christian Schilling, and Andreas Podelski. 2013. Ultimate automizer with SMT-Interpol. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 641–643.

[16] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2009. Refinement of trace abstraction. In *International Static Analysis Symposium*. Springer, 69–85.

[17] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2013. Software model checking for people who love automata. In *International Conference on Computer Aided Verification*. Springer, 36–52.

[18] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2014. Termination analysis by learning terminating programs. In *International Conference on Computer Aided Verification*. Springer, 797–813.

[19] Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018. New approaches for almost-sure termination of probabilistic programs. In *Asian Symposium on Programming Languages and Systems*. Springer, 181–201.

[20] Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2019. Modular verification for almost-sure termination of probabilistic programs. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 129.

[21] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest precondition reasoning for expected run–times of probabilistic programs. In *European Symposium on Programming*. Springer, 364–389.

[22] Annabelle McIver and Carroll Morgan. 2004. Developing and reasoning about probabilistic programs in pGCL. In *Pernambuco Summer School on Software Engineering*. Springer, 123–155.

[23] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2017. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 33.

[24] Annabelle McIver, Carroll Morgan, and Charles Carroll Morgan. 2005. *Abstraction, refinement and proof for probabilistic systems*. Springer Science & Business Media.

[25] Flemming Nielson and Hanne Riis Nielson. 2019. *Formal Methods: An Appetizer*. Springer.

[26] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about recursive probabilistic programs. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–10.

[27] Andreas Podelski and Andrey Rybalchenko. 2004. A complete method for the synthesis of linear ranking functions. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 239–251.