Lexicographic Ranking Supermartingales with Lazy Lower Bounds

Toru Takisaka, Libo Zhang, Changjiang Wang, Jiamou Liu

Lexicographic Ranking SuperMartingale (LexRSM) is a probabilistic extension of *Lexicographic Ranking Function* (LexRF), which is a widely accepted technique for verifying program termination. In this paper, we are the first to propose sound probabilistic extensions of LexRF with a weaker non-negativity condition, called *single-component* (SC) non-negativity. It is known that such an extension, if it exists, will be nontrivial due to the intricacies of the probabilistic circumstances. Toward the goal, we first devise the notion of *fixability*, which offers a systematic approach for analyzing the soundness of possibly negative LexRSM. This notion yields a desired extension of LexRF that is sound for general stochastic processes. We next propose another extension, called *Lazy LexRSM*, toward the application to automated verification; it is sound over probabilistic programs with linear arithmetics, while its subclass is amenable to automated synthesis via linear programming. We finally propose a LexRSM synthesis algorithm for this subclass, and perform experiments.

1 Introduction

Background 1: Lexicographic RFs with different non-negativity conditions. *Ranking function* (RF) is one of the most well-studied tools for verifying program termination. An RF is typically a real-valued function over program states that satisfies: (a) the *ranking condition*, which requires an RF to decrease its value by a constant through each transition; and (b) the *non-negativity condition*, which imposes a lower bound on the value of the RF so that its infinite descent through transitions is prohibited. The existence of such a function implies termination of the underlying program, and therefore, one can automate verification of program termination by RF synthesis algorithms.

Improving the *applicability* of RF synthesis algorithms, i.e., making them able to prove termination of a wider variety of programs, is one of the core interests in the study of RF. A *lexicographic extension* of RF (LexRF) [8,10] is known as a simple but effective approach to the problem. Here, a LexRF is a function to real-valued *vectors* instead of the reals, and its ranking condition is imposed with respect to the lexicographic order. For example, the value of a LexRF may change from $(1, 1, 1)$ to $(1, 0, 2)$ through a state transition; here, the value "lexicographically decreases by 1" through the transition, that is, it decreases by 1 in some dimension while it is non-increasing on the left to that dimension. LexRF is particularly good at handling nested structures of programs, as vectors can measure the progress of

* ゆるやかな下限を持つ辞書式順序の超マルチンゲール
This is an unreferred paper, whose content has been published in the proceedings of The 36th International Conference on Computer Aided Verification (CAV 2024).

Toru Takisaka, Changjiang Wang, 電子科技大学 計算機科学・工学部, School of Computer Science and Engineering, University of Electronic Science and Technology of China.

Libo Zhang, Jimou Liu, オークランド大学 計算機科学部, School of Computer Science, The University of Auckland.

$\ell_1 : \mathbf{skip}\,; \quad //\boldsymbol{\eta} = (a_1, \underline{b_1}, c_1)$

$\ell_2 : x := 1\,; \quad //\boldsymbol{\eta} = (\underline{a_2}, b_2, c_2)$

...

Non-negativity condition	$\boldsymbol{\eta}$ should be non-neg. at
Strong non-neg.	$a_1, b_1, c_1, a_2, b_2, c_2$
Leftward non-neg.	a_1, b_1, a_2
Single-component non-neg.	b_1, a_2

図 1 A demo of different non-negativity conditions for LexRFs. There, the ranking dimensions of the LexRF $\boldsymbol{\eta}$ are indicated by underlines, and the last column of the table shows where each condition requires $\boldsymbol{\eta}$ to be non-negative.

different "phases" of programs separately. LexRF is also used in top-performing termination provers (e.g., [1]).

There are several known ways to impose non-negativity on LexRFs (see also Fig. 1): (a) *Strong non-negativity*, which requires non-negativity in every dimension of the LexRF; (b) *leftward non-negativity*, which requires non-negativity on the left of the *ranking dimension* of each transition, i.e., the dimension where the value of the LexRF should strictly decrease through the transition; and (c) *single-component non-negativity*, which requires non-negativity only in the ranking dimensions. It is known that any of these non-negativity conditions makes the resulting LexRF *sound* [8, 10], i.e., a program indeed terminates whenever it admits a LexRF with either of these non-negativity conditions. For better applicability, single-component non-negativity is the most preferred, as it is the weakest constraint among the three.

Background 2: Probabilistic programs and lexicographic RSMs. One can naturally think of a probabilistic counterpart of the above argument. One can consider *probabilistic programs* that admit randomization in conditional branching and variable updates. The notion of RF is then generalized to *Ranking SuperMartingale* (RSM), a function similar to RFs except that the ranking condition requires an RSM to decrease its value *in expectation*. The existence of an RSM typically implies *almost-sure termination* of the underlying program, i.e., termination of the program with probability 1.

Such a probabilistic extension has been actively studied, in fact: probabilistic programs are used in e.g., stochastic network protocols [33], randomized algorithms [18, 26], security [6, 7, 27], and planning [11]; and there is a rich body of studies in RSM as a tool for automated verification of probabilistic programs (see §8). Similar to the RF case, a lexicographic extension of RSM (*LexRSM*, [2, 15]) is an effective approach to improve its applicability. In addition to its advantages over nested structures, LexRSM can also witness almost-sure termination of certain probabilistic programs with infinite expected runtime [2, Fig. 2]; certifying such programs is known as a major challenge for RSMs.

Problem: Sound probabilistic extension of LexRF with weaker non-negativity. strongly non-negative LexRF soundly extends to LexRSM in a canonical way [2], i.e., basically by changing the clause "decrease by a constant" in the ranking condition of LexRF to "decrease by a constant *in expectation*". In contrast, the similar extension of leftward or single-component non-negative LexRF yields an *unsound* LexRSM notion [15, 20]. To date, a sound LexRSM with the weakest non-negativity in the literature is *Generalized LexRSM* (GLexRSM) [15], which demands leftward non-negativity *and* an additional one, so-called *expected leftward non-negativity*. Roughly speaking, the latter requires LexRSMs to be non-negative in each dimension (in expectation) upon "exiting" the left of the ranking dimension. For example, in Fig. 1, it requires b_2 to be non-negative, as the second dimension of $\boldsymbol{\eta}$ "exits" the left of the ranking dimension upon the transition $\ell_1 \to \ell_2$. GLexRSM does

not generalize either leftward or single-component non-negative LexRF, in the sense that the former is strictly more restrictive than the latter two when it is considered over non-probabilistic programs.

These results do not mean that leftward or single-component non-negative LexRF can never be extended to LexRSM, however. More concretely, the following problem is valid (see the last paragraph of §3 for a formal argument):

> *KEY PROBLEM: Find a sound LexRSM notion that* instantiates[†1] *single-component non-negative LexRF, i.e., a LexRSM notion whose condition is no stronger than that of single-component non-negative LexRF in non-probabilistic settings.*

We are motivated to study this problem for a couple of reasons. First, it is a paraphrase of the following fundamental question: *when do negative values of (Lex)RSM cause trouble*, say, to its soundness? This question is a typical example in the study of RSM where the question becomes challenging due to its probabilistic nature. The question also appears in other topics in RSM; for example, it is known that the classical variant rule of Floyd-Hoare logic does not extend to almost-sure termination of probabilistic programs in a canonical way [24], due to the complicated treatment of negativity in RSMs. To our knowledge, this question has only been considered in an ad-hoc manner through counterexamples (e.g., [15, 20, 24]), and we do not yet have a systematic approach to answering it.

Second, relaxing the non-negativity condition of LexRSM is highly desirable if we wish to fully unlock the benefit of the lexicographic extension in automated verification. A motivating example is given in Fig. 2. The probabilistic program in Fig. 2 terminates almost-surely, but it does not admit any linear GLexRSM (and hence, the GLexRSM synthesis algorithms in [15] cannot witness its almost-sure termination); for example, the function $\boldsymbol{\eta}$ ranks every transition of the program, but violates both leftward and expected leftward non-negativity at the transition $\ell_1 \to \ell_2$ (note $\boldsymbol{\eta}$ ranks this transition in the third dimension; to check the violation of expected leftward non-negativity, also note $\boldsymbol{\eta}$ ranks $\ell_2 \to \ell_4$ in the first dimension). Here, the source of the problem is that the program has two variables whose progress must be measured (i.e., increment y to 10 in ℓ_3; and increment x to 5 in ℓ_4), but one of their progress measures can be arbitrarily small during the program execution (y can be initialized with any value). Not only that this structure is rather fundamental, it is also expected that our desired LexRSM could handle it, if it exists. Indeed, modify the probabilistic program in Fig. 2 into a non-probabilistic one by changing "$\mathit{Unif}[1,2]$" to "1"; then the program admits $\boldsymbol{\eta}$ as a single-component non-negative LexRF.

Contributions. In this paper, we are the first to introduce sound LexRSM notions that instantiate single-component non-negative LexRF. Our contributions are threefold, as we state below.

- First, in response to the first motivation we stated above, we devise a novel notion of *fixability* as a theoretical tool to analyze if negative values of a LexRSM "cause trouble". Roughly speaking, we identify the source of the trouble as "ill" exploitation of unbounded negativity of LexRSM; our *ε-fixing* operation prohibits such exploitation by basically setting all the negative values of a LexRSM into the same negative value $-\varepsilon$, and we say a LexRSM is *ε-fixable* if it retains the ranking condition through such a transformation. We give more details about its concept and key ideas in §2.

  The soundness of ε-fixable LexRSM immediately follows from that of strongly non-

†1 We use the term "instantiate" to emphasize that we compare LexRSM and LexRF.

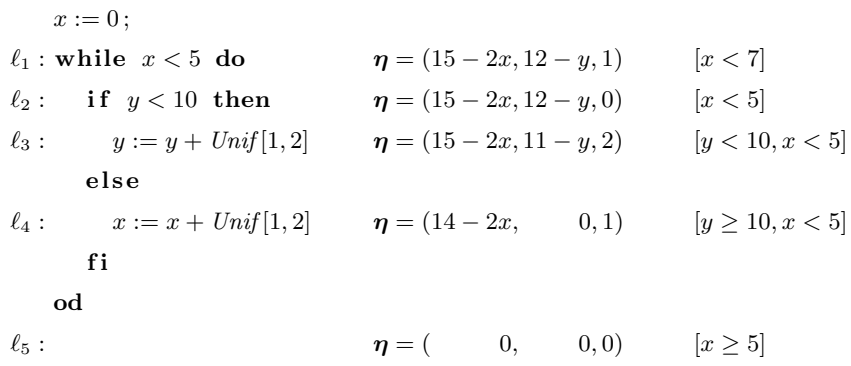

図 2 A probabilistic modification of speedDis1 **[4], where** $\mathit{Unif}[a, b]$ **is a uniform sampling from the (continuous) interval** $[a, b]$**. Inequalities on the right represent invariants. While** $\boldsymbol{\eta}$ **is not a GLexRSM, it is an LLexRSM we propose; thus it witnesses almost-sure termination of the program.**

negative one [2] because any LexRSM becomes strongly non-negative through the ε-fixing operation (after globally adding ε). Fixable LexRSM instantiates single-component non-negative LexRF for general stochastic processes (Thm. 4.3), while also serving as a technical basis for proving the soundness of other LexRSMs. Meanwhile, fixable LexRSM cannot be directly applied to automated verification algorithms due to the inherent non-linearity of ε-fixing; this observation leads us to our second contribution.

- Second, in response to the second motivation we stated above, we introduce *Lazy LexRSM* (LLexRSM) as another LexRSM notion that instantiates single-component non-negative LexRF. LLexRSM does not involve the ε-fixing operation in its definition; thanks to this property, we have a subclass of LLexRSM that is amenable to automated synthesis via linear programming (see §6). The LLexRSM condition consists of the single-component non-negative LexRSM condition and *stability at negativity* we propose (Def. 5.1), which roughly requires the following: Once the value of a LexRSM gets negative in some dimension, it must stay negative until that dimension exits the left of the ranking one. For example, $\boldsymbol{\eta}$ in Fig. 2 is an LLexRSM; indeed, $\ell_2 \to \ell_4$ and $\ell_1 \to \ell_5$ are the only transitions where $\boldsymbol{\eta}$ possibly changes its value from negative to non-negative in some dimension (namely, the second one), which is although the right to the ranking dimension (the first one).

  We prove linear LLexRSM is sound for probabilistic programs over linear arithmetics (see Thm. 5.4 for the exact assumption). The proof is highly nontrivial, which is realized by subtle use of a refined variant of fixability; we explain its core idea in §2. Furthermore, Thm. 5.4 shows that expected leftward non-negativity in GLexRSM [15] is actually redundant under the assumption in Thm. 5.4. This is surprising, as expected leftward non-negativity has been invented to restore the soundness of leftward non-negative LexRSM, which is generally unsound.
- Third, we present a synthesis algorithm for the subclass of LLexRSM we mentioned above, and do experiments; there, our algorithms verified almost-sure termination of various programs

that could not be handled by (a better proxy of) the GLexRSM-based one. The details can be found in §7.

2 Key Observations with Examples

Here we demonstrate by examples how intricate the treatment of negative values of LexRSM is, and how we handle it by our proposed notion of fixability.

Blocking "ill" exploitation of unbounded negativity. Fig. 3 is a counterexample that shows leftward non-negative LexRSM is generally unsound (conceptually the same as [15, Ex. 1]). The probabilistic program in Fig. 3 does not terminate almost-surely because the chance of entering ℓ_4 from ℓ_3 quickly decreases as t increases. Meanwhile, $\boldsymbol{\eta} = (\eta_1, \eta_2, \eta_3)$ in Fig. 3 is a leftward non-negative LexRSM over a global invariant $[0 \leq x \leq 1]$; in particular, observe η_2 decreases by 1 in expectation from ℓ_3, whose successor location is either ℓ_4 or ℓ_1.

$x := 0\,; t := 1\,;$

ℓ_1 : **while** $x = 0$ **do** $\quad \boldsymbol{\eta} = (2 - x, \quad 0, 2)$

ℓ_2 : $\quad t := t + 1\,;$ $\quad \boldsymbol{\eta} = (\quad 2, \quad 0, 1)$

ℓ_3 : $\quad$ **if prob**(2^{-t}) $\quad \boldsymbol{\eta} = (\quad 2, \quad 0, 0)$

ℓ_4 : $\quad\quad$ **then** $x := 1$ $\quad \boldsymbol{\eta} = (\quad 2, -2^t, 0)$

$\quad\quad$ **fi**

$\quad$ **od**

ℓ_5 : $\quad \boldsymbol{\eta} = (\quad 0, \quad 0, 0)$

図 3 An example of "ill" exploitation.

This example reveals an inconsistency between the ways how the single-component non-negativity and ranking condition evaluate the value of a LexRSM, say $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$. The single-component non-negativity claims $\boldsymbol{\eta}$ cannot rank a transition in a given dimension k whenever η_k is negative; intuitively, this means that *any* negative value in the ranking domain $\mathbb{R}$ should be understood as the same state, namely the "bottom" of the domain. Meanwhile, the ranking condition evaluates different negative values differently; a smaller negative value of η_k can contribute more to satisfy the ranking condition, as one can see from the behavior of η_2 in Fig. 3 at ℓ_3. The function $\boldsymbol{\eta}$ in Fig. 3 satisfies the ranking condition over a possibly non-terminating program through "ill" exploitation of this inconsistency; as t becomes larger, the value of η_2 potentially drops more significantly through the transition from ℓ_3, but with a smaller probability.

The first variant of our fixability notion, called *ε-fixability*, enables us to ensure that such exploitation is not happening. We simply set every negative value in a LexRSM $\boldsymbol{\eta}$ to a negative constant $-\varepsilon$, and say $\boldsymbol{\eta}$ is ε-fixable if it retains the ranking condition through the modification[†2]. For example, the ε-fixing operation changes the value of η_2 in Fig. 3 at ℓ_4 from -2^t to $-\varepsilon$, and $\boldsymbol{\eta}$ does not satisfy the ranking condition after that. Therefore, $\boldsymbol{\eta}$ in Fig. 3 is not ε-fixable for any $\varepsilon > 0$ (i.e., we successfully reject this $\boldsymbol{\eta}$ through the fixability check). Meanwhile, an ε-fixable LexRSM witnesses almost-sure termination of the underlying program; indeed, the fixed LexRSM is a strongly non-negative LexRSM (by globally adding ε to the fixed $\boldsymbol{\eta}$), which is known to be sound [2].

The notion of ε-fixability is operationally so simple that one might even feel it is a boring idea; nevertheless, its contribution to revealing the nature of possibly negative LexRSM is already significant in our paper. Indeed, (a) ε-fixable LexRSM instantiates single-component non-negative LexRF with an appropriate ε (Thm. 4.3); (b) ε-fixable LexRSM

†2 To give the key ideas in a simpler way, the description here slightly differs from the actual definition in §4; referred results in §2 are derived from the latter. See Rem. 4.1.

generalizes GLexRSM [15], and the proof offers an alternative proof of soundness of GLexRSM that is significantly simpler than the original one (Thm. 4.4); and (c) its refined variant takes the crucial role in proving soundness of our second LexRSM variant, lazy LexRSM.

Allowing "harmless" unbounded negativity. While ε-fixable LexRSM already instantiates single-component non-negative LexRF, we go one step further to obtain a LexRSM notion that is amenable to automated synthesis, in particular via *Linear Programming* (LP). The major obstacle to this end is the case distinction introduced by ε-fixability, which makes the fixed LexRSM nonlinear. *Lazy LexRSM* (LLexRSM), our second proposed LexRSM, resolves this problem while it also instantiates single-component non-negative LexRF.

Linear LLexRSM is sound over probabilistic programs with linear arithmetics (Thm. 5.4). The key to the proof is, informally, the following observation: *Restrict our attention to probabilistic programs and functions $\boldsymbol{\eta}$ that are allowed in the LP-based synthesis. Then "ill" exploitation in Fig. 3 never occurs*, and therefore, a weaker condition than ε-fixability (namely, the LLexRSM one) suffices for witnessing program termination. In fact, Fig. 3 involves (a) non-linear arithmetics in the program, (b) parametrized if-branch in the program (i.e., the grammar " **if prob**(p) **then** P **else** Q **fi** " with p being a variable), and (c) non-linearity of $\boldsymbol{\eta}$. None of them are allowed in the LP-based synthesis (at least, in the standard LP-based synthesis via Farkas' Lemma [2, 12, 15]). Our informal statement above is formalized as Thm. 5.3, which roughly says: Under such a restriction to probabilistic programs and $\boldsymbol{\eta}$, any LLexRSM is (ε, γ)-*fixable*. Here, (ε, γ)-fixability is a refined version of ε-fixability; while it also ensures that "ill" exploitation is not happening in $\boldsymbol{\eta}$, it is less restrictive than ε-fixability by allowing "harmless" unbounded negative values of $\boldsymbol{\eta}$.

$x := 0\,; t := 1\,;$
ℓ_1 : **while** $x = 0$ **do** $\quad \boldsymbol{\eta} = (2 - x, \quad t + 1)$
ℓ_2 : **if prob**(0.5) $\quad \boldsymbol{\eta} = (\quad 2, \quad t)$
ℓ_3 : **then** $t := 4t$ $\quad \boldsymbol{\eta} = (\quad 2, \quad 4t + 2)$
ℓ_4 : **else** $x := 1$ $\quad \boldsymbol{\eta} = (\quad 2, -2t - 4)$
fi
od
ℓ_5 : $\quad \boldsymbol{\eta} = (\quad 0, \quad 0)$

図 4 An example of "harmless" unbounded negativity.

Fig. 4 gives an example of such a harmless behavior of $\boldsymbol{\eta}$ rejected by ε-fixability. It also shows why we cannot simply use ε-fixability to check an LLexRSM does not do "ill" exploitation. The function $\boldsymbol{\eta} = (\eta_1, \eta_2)$ in Fig. 4 is leftward non-negative over the global invariant $[0 \leq x \leq 1 \land t \geq 1]$, so it is an LLexRSM for the probabilistic program there; the program and $\boldsymbol{\eta}$ are also in the scope of LP-based synthesis; but $\boldsymbol{\eta}$ is not ε-fixable for any $\varepsilon > 0$. Indeed, the ε-fixing operation changes the value of η_2 at ℓ_4 from $-2t - 4$ to $-\varepsilon$, and $\boldsymbol{\eta}$ does not satisfy the ranking condition at ℓ_2 after the change. Here we notice that, however, the unbounded negative values of η_2 are "harmless"; that is, the "ill-gotten gains" by the unbounded negative values of η_2 at ℓ_4 are only "wasted" to unnecessarily increase η_2 at ℓ_3. In fact, $\boldsymbol{\eta}$ still satisfies the ranking condition if we change the value of η_2 at ℓ_1, ℓ_2, ℓ_3 to $2, 1$, and 0, respectively.

We resolve this issue by partially waiving the ranking condition of $\boldsymbol{\eta}$ after the ε-fixing operation. It is intuitively clear that the program in Fig. 4 almost-surely terminates, and the intuition here is that the program essentially repeats an unbiased coin tossing until the tail is observed (here, "observe the tail" corresponds to "observe **prob**(0.5) =

false at ℓ_2"). This example tells us that, to witness the almost-sure termination of this program, we only need to guarantee the program (almost-surely) visits either the terminal location ℓ_5 *or the "coin-tossing location"* ℓ_2 from anywhere else. The ε-fixed $\boldsymbol{\eta}$ in Fig. 4 does witness such a property of the program, as it ranks every transition *except those that are from a coin-tossing location*, namely ℓ_2.

We generalize this idea as follows: Fix $\gamma \in (0,1)$, and say a program state is a "coin-tossing state" for $\boldsymbol{\eta} = (\eta_1, \dots, \eta_n)$ in the k-th dimension if η_k drops from non-negative to negative (i.e., the ranking is "done" in the k-th dimension) with the probability γ or higher. Then we say $\boldsymbol{\eta}$ is (ε, γ)-*fixable* (Def. 4.6) if the ε-fixed $\boldsymbol{\eta}$ is a strongly non-negative LexRSM (after adding ε) *except that*, at each coin-tossing state, we waive the ranking condition of $\boldsymbol{\eta}$ in the corresponding dimension. For example, $\boldsymbol{\eta}$ in Fig. 4 is (ε, γ)-fixable for any $\gamma \in (0, 0.5]$. As expected, (ε, γ)-fixable LexRSM is sound for any $\varepsilon > 0$ and $\gamma \in (0,1)$ (Cor. 4.7).

3 Preliminaries

We recall the technical preliminaries. Omitted details are in [36].

Notations. We assume the readers are familiar with the basic notions of measure theory, see e.g. [5, 9]. The sets of non-negative integers and reals are denoted by $\mathbb{N}$ and $\mathbb{R}$, respectively. The collection of all Borel sets of a topological space $\mathcal{X}$ is denoted by $\mathcal{B}(\mathcal{X})$. The set of all probability distributions over the measurable space $(\Omega, \mathcal{B}(\Omega))$ is denoted by $\mathcal{D}(\Omega)$. The value of a vector $\boldsymbol{x}$ at the i-th index is denoted by $\boldsymbol{x}[i]$ or x_i. A subset $D \subseteq \mathbb{R}$ of the reals is *bounded* if $D \subseteq [-x, x]$ for some $x > 0$.

For a finite variable set V and the set val^V of its valuations, we form *predicates* as first-order formulas with atomic predicates of the form $f \leq g$, where $f, g \colon val^V \to R$ and R is linearly ordered. Often, we are only interested in the value of a predicate φ over a certain subset $\mathcal{X} \subseteq val^V$, in which case, we call φ a predicate *over* $\mathcal{X}$. We identify a predicate φ over $\mathcal{X}$ with a function $\tilde{\varphi} \colon \mathcal{X} \to \{0,1\}$ such that $\tilde{\varphi}(x) = 1$ if and only if $\varphi(x)$ is true. The *semantics* of φ, i.e., the set $\{x \in \mathcal{X} \mid \varphi(x) \text{ is true}\}$, is denoted by $[\![\varphi]\!]$. The *characteristic function* $\mathbf{1}_A : \mathcal{X} \to \{0,1\}$ of a subset A of $\mathcal{X}$ is a function such that $[\![\mathbf{1}_A = 1]\!] = A$. For a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, we say φ over Ω is *($\mathcal{F}$-)measurable* when $[\![\varphi]\!] \in \mathcal{F}$. For such a φ, the *satisfaction probability* of φ w.r.t. $\mathbb{P}$, i.e., the value $\mathbb{P}([\![\varphi]\!])$, is also denoted by $\mathbb{P}(\varphi)$; we say φ *holds $\mathbb{P}$-almost surely* ($\mathbb{P}$-a.s.) if $\mathbb{P}(\varphi) = 1$.

3.1 Syntax and Semantics of Probabilistic Programs

Syntax. We define the syntax of *Probabilistic Programs* (PPs) similarly to e.g., [2, 35]. More concretely, PPs have the standard control structure in imperative languages such as if-branches and while-loops, while the if-branching and variable assignments can also be done in either nondeterministic or probabilistic ways. Namely, '**if** $\star$' describes a nondeterministic branching; '**ndet**(D)' describes a nondeterministic assignment chosen from a bounded[†3] domain $D \subseteq \mathcal{B}(\mathbb{R})$; '**if prob**$(p)$' with a constant $p \in [0,1]$ describes a probabilistic branching that executes the '**then**' branch with probability p, or the '**else**' branch with probability $1-p$; and '**sample**(d)' describes a probabilistic assignment sampled from a distribution $d \in \mathcal{D}(\mathbb{R})$. We consider PPs without *conditioning*, which are also called *randomized programs* [35]; PPs with conditioning are considered in e.g. [32]. The exact grammar is given in [36].

†3 This is also assumed in [15] to avoid a complication in possibly negative LexRSMs.

In this paper, we focus our attention on PPs with linear arithmetics; we say a PP is *linear* if each arithmetic expression in it is linear, i.e., of the form $b + \sum_{i=1}^{n} a_i \cdot v_i$ for constants $a_1, \ldots, a_n, b$ and program variables $v_1, \ldots, v_n$.

Semantics. We adopt *probabilistic control flow graph* (pCFG) as the semantics of PPs, which is standard in existing RSM works (e.g., [12, 15, 35]). Informally, it is a labeled directed graph whose vertices are program locations, and whose edges represent possible one-step executions in the program. Edges are labeled with the necessary information so that one can reconstruct the PP represented by the pCFG; for example, an edge e can be labeled with the assignment commands executed through e (e.g., '$x := x + 1$'), the probability $p \in [0, 1]$ that e will be chosen (through '**if prob**(p)'), the guard condition, and so on. Below we give its formal definition for completeness; see [36] for how to translate PPs into pCFGs.

Definition 3.1 (pCFG)**.** *A* pCFG *is a tuple* (L, V, Δ, Up, G)*, where*

1. L *is a finite set of locations.*
2. $V = \{x_1, \ldots, x_{|V|}\}$ *is a finite set of program variables.*
3. Δ *is a finite set of* (generalized) transitions[†4]*, i.e., tuples* $\tau = (\ell, \delta)$ *of a location* $\ell \in L$ *and a distribution* $\delta \in \mathcal{D}(L)$ *over successor locations.*
4. Up *is a function that receives a transition* $\tau \in \Delta$ *and returns a tuple* (i, u) *of a target variable index* $i \in \{1, \ldots, |V|\}$ *and an update element* u*. Here,* u *is either (a) a Borel measurable function* $u : \mathbb{R}^{|V|} \to \mathbb{R}$*, (b) a distribution* $d \in \mathcal{D}(\mathbb{R})$*, or (c) a bounded measurable set* $R \in \mathcal{B}(\mathbb{R})$*. In each case, we say* τ *is* deterministic*,* probabilistic*, and* non-deterministic*, respectively; the collections of these transitions are denoted by* Δ_d*,* Δ_p*, and* Δ_n*, respectively.*
5. G *is a guard function that assigns a* $G(\tau) : \mathbb{R}^{|V|} \to \{0, 1\}$ *to each* $\tau \in \Delta$*.*

Below we fix a pCFG $\mathcal{C} = (L, V, \Delta, Up, G)$. A *state* of $\mathcal{C}$ is a tuple $s = (\ell, \boldsymbol{x})$ of location $\ell \in L$ and variable assignment vector $\boldsymbol{x} \in \mathbb{R}^{|V|}$. We write $\mathcal{S}$ to denote the state set $L \times \mathbb{R}^{|V|}$. Slightly abusing the notation, for $\tau = (\ell, \delta)$, we identify the set $[\![G(\tau)]\!] \subseteq \mathbb{R}^{|V|}$ and the set $\{\ell\} \times [\![G(\tau)]\!] \subseteq \mathcal{S}$; in particular, we write $s \in [\![G(\tau)]\!]$ when τ is *enabled at* s, i.e., $s = (\ell, \boldsymbol{x})$, $\tau = (\ell, \delta)$ and $\boldsymbol{x} \in [\![G(\tau)]\!]$.

A pCFG $\mathcal{C}$ with its state set $\mathcal{S}$ can be understood as a transition system over $\mathcal{S}$ with probabilistic transitions and nondeterminism (or, more specifically, a Markov decision process with its states $\mathcal{S}$). Standard notions such as *successors* of a state $s \in \mathcal{S}$, *finite paths*, and *(infinite) runs* of $\mathcal{C}$ are defined as the ones over such a transition system. The set of all successors of $s \in [\![G(\tau)]\!]$ via τ is denoted by $\mathrm{succ}_\tau(s)$. The set of runs of $\mathcal{C}$ is denoted by $\Pi_\mathcal{C}$.

Schedulers resolve nondeterminism in pCFGs. Observe there are two types of nondeterminism: (a) nondeterministic choice of $\tau \in \Delta$ at a given state (corresponds to '**if** $\star$'), and (b) nondeterministic variable update in a nondeterministic transition $\tau \in \Delta_n$ (corresponds to '$x_i :=$**ndet**(D)'). We say a scheduler is Δ-*deterministic* if its choice is non-probabilistic in Case (a).

We assume pCFGs are deadlock-free; we also assume that there are designated locations ℓ_{in} and ℓ_{out} that represent program initiation and termination, respectively. An *initial state* is a state of the form $(\ell_{\mathrm{in}}, \boldsymbol{x})$. We assume a transition from ℓ_{out} is unique, denoted by τ_{out}; this transition does not update anything.

By fixing a scheduler σ and an initial state s_I, the infinite-horizon behavior of $\mathcal{C}$ is determined as a distribution $\mathbb{P}^\sigma_{s_I}$ over $\Pi_\mathcal{C}$; that is, for a measurable

†4 Defining these as edges might be more typical, as in our informal explanation. We adopt the style of [2, 15] for convenience; it can handle '**if prob**(p)' by a single τ.

$A \subseteq \Pi_{\mathcal{C}}$, the value $\mathbb{P}^{\sigma}_{s_I}(A)$ is the probability that a run of $\mathcal{C}$ from s_I is in A under σ. We call the probability space $(\Pi_{\mathcal{C}}, \mathcal{B}(\Pi_{\mathcal{C}}), \mathbb{P}^{\sigma}_{s_I})$ the *dynamics of $\mathcal{C}$ under σ and s_I*. See [9] for the formal construction; a brief explanation is in [36].

We define the *termination time* of a pCFG $\mathcal{C}$ as the function $T^{\mathcal{C}}_{\mathrm{term}} : \Pi_{\mathcal{C}} \to \mathbb{N} \cup \{+\infty\}$ such that $T^{\mathcal{C}}_{\mathrm{term}}(s_0 s_1 \ldots) = \inf\{t \in \mathbb{N} \mid \exists \boldsymbol{x}. s_t = (\ell_{\mathrm{out}}, \boldsymbol{x})\}$. Now we formalize our objective, i.e., almost-sure termination of pCFG, as follows.

Definition 3.2 (AST of pCFG)**.** *A run $\omega \in \Pi_{\mathcal{C}}$* terminates *if $T^{\mathcal{C}}_{\mathrm{term}}(\omega) < \infty$. A pCFG $\mathcal{C}$ is* a.s. terminating *(AST) under a scheduler σ and an initial state s_I if a run of $\mathcal{C}$ terminates $\mathbb{P}^{\sigma}_{s_I}$-a.s. We say $\mathcal{C}$ is AST if it is AST for any σ and s_I.*

3. 2 Lexicographic Ranking Supermartingales

Here we recall mathematical preliminaries of the LexRSM theory. A (Lex)RSM typically comes in two different forms: one is a vector-valued function $\boldsymbol{\eta} : \mathcal{S} \to \mathbb{R}^n$ over states $\mathcal{S}$ of a pCFG $\mathcal{C}$, and another is a stochastic process over the runs $\Pi_{\mathcal{C}}$ of $\mathcal{C}$. We recall relevant notions in these formulations, which are frequently used in existing RSM works [12, 15]. We also recall the formal definition of LexRSMs with three different non-negativity conditions in Fig. 1.

LexRSM as a quantitative predicate. Fix a pCFG $\mathcal{C}$. An *(n-dimensional) measurable map* (MM) is a Borel measurable function $\boldsymbol{\eta} : \mathcal{S} \to \mathbb{R}^n$. For a given 1-dimensional MM η and a transition τ, The (maximal) *pre-expectation* of η under τ is a function that formalizes "the value of η after the transition τ". More concretely, it is a function $\overline{\mathbb{X}}_{\tau}\eta : [\![G(\tau)]\!] \to \mathbb{R}$ that returns, for a given state s, the maximal expected value of η at the successor state of s via τ. Here, the maximality refers to the set of all possible nondeterministic choices at s.

A *level map* $\mathsf{Lv} : \Delta \to \{0, \ldots, n\}$ designates the ranking dimension of the associated LexRSM $\boldsymbol{\eta} : \mathcal{S} \to \mathbb{R}^n$. We require $\mathsf{Lv}(\tau) = 0$ if and only if $\tau = \tau_{\mathrm{out}}$. We say an MM $\boldsymbol{\eta}$ *ranks* a transition τ in the dimension k (under Lv) when $k = \mathsf{Lv}(\tau)$. An *invariant* is a measurable predicate $I : \mathcal{S} \to \{0, 1\}$ such that $[\![I]\!]$ is closed under transitions and $\ell_{\mathrm{in}} \times \mathbb{R}^{|V|} \subseteq [\![I]\!]$. The set $[\![I]\!]$ over-approximates the reachable states in $\mathcal{C}$.

Suppose an n-dimensional MM $\boldsymbol{\eta}$ and an associated level map Lv are given. We say $\boldsymbol{\eta}$ satisfies *the ranking condition* (under Lv and I) if the following holds for each $\tau \neq \tau_{\mathrm{out}}$, $s \in [\![I \wedge G(\tau)]\!]$, and $k \in \{1, \ldots, \mathsf{Lv}(\tau)\}$:

$$\overline{\mathbb{X}}_{\tau}\boldsymbol{\eta}[k](s) \leq \begin{cases} \boldsymbol{\eta}[k](s) & \text{if } k < \mathsf{Lv}(\tau), \\ \boldsymbol{\eta}[k](s) - 1 & \text{if } k = \mathsf{Lv}(\tau). \end{cases}$$

We also define the three different non-negativity conditions in Fig. 1, i.e., *STrong (ST)*, *LeftWard (LW)*, and *Single-Component (SC)* non-negativity, as follows:

$$\begin{aligned} \text{(ST non-neg.)}\ & \forall s \in [\![I]\!]. \forall k \in \{1, \ldots, n\}. \\ & \boldsymbol{\eta}[k](s) \geq 0, \\ \text{(LW non-neg.)}\ & \forall \tau \neq \tau_{\mathrm{out}}. \forall s \in [\![I \wedge G(\tau)]\!]. \\ & \forall k \in \{1, \ldots, \mathsf{Lv}(\tau)\}. \\ & \boldsymbol{\eta}[k](s) \geq 0, \\ \text{(SC non-neg.)}\ & \forall \tau \neq \tau_{\mathrm{out}}. \forall s \in [\![I \wedge G(\tau)]\!]. \\ & \boldsymbol{\eta}[\mathsf{Lv}(\tau)](s) \geq 0. \end{aligned}$$

All the materials above are wrapped up in the following definition.

Definition 3.3 ((ST/LW/SC)-LexRSM map)**.** *Fix a pCFG $\mathcal{C}$ with an invariant I. Let $\boldsymbol{\eta}$ be an MM associated with a level map Lv. The MM $\boldsymbol{\eta}$ is called a* STrongly non-negative LexRSM map (ST-LexRSM map) *over $\mathcal{C}$ supported by I if it satisfies the ranking condition and the strong non-negativity under Lv and I. If it satisfies the leftward or single-component non-negativity instead of the strong one, then we call it* LW-LexRSM map *or* SC-LexRSM map*, respectively.*

LexRSM as a stochastic process. When it

comes to automated synthesis, a (Lex)RSM is usually a function $\boldsymbol{\eta}$ over program states, as defined in Def. 3.3. Meanwhile, when we prove the properties of (Lex)RSMs themselves (e.g., soundness), it is often necessary to inspect the behavior of $\boldsymbol{\eta}$ upon the program execution under given scheduler σ and initial state s_I. Such a behavior of $\boldsymbol{\eta}$ is formalized as a sequence $(\mathbf{X}_t)_{t=0}^{\infty}$ of random variables over the dynamics of the underlying pCFG, which forms a *stochastic process*.

A (discrete-time) *stochastic process* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is a sequence $(\mathbf{X}_t)_{t=0}^{\infty}$ of $\mathcal{F}$-measurable random variables $\mathbf{X}_t : \Omega \to \mathbb{R}^n$ for $t \in \mathbb{N}$. In our context, it is typically associated with another random variable $T : \Omega \to \mathbb{N} \cup \{+\infty\}$ that describes the termination time of $\omega \in \Omega$. We say T is *AST (w.r.t. $\mathbb{P}$)* if $\mathbb{P}(T < \infty) = 1$; observe that, if $(\Omega, \mathcal{F}, \mathbb{P})$ is the dynamics of a pCFG $\mathcal{C}$ under σ and s_I, then $\mathcal{C}$ is AST under σ and s_I if and only if $T^{\mathcal{C}}_{\mathrm{term}}$ is AST w.r.t. $\mathbb{P}$. As standard technical requirements, we assume there is a *filtration* $(\mathcal{F}_t)_{t=0}^{\infty}$ in $(\Omega, \mathcal{F}, \mathbb{P})$ such that $(\mathbf{X}_t)_{t=0}^{\infty}$ is *adapted* to $(\mathcal{F}_t)_{t=0}^{\infty}$, T is a *stopping time* w.r.t. $(\mathcal{F}_t)_{t=0}^{\infty}$, and $(\mathbf{X}_t)_{t=0}^{\infty}$ is *stopped* at T; see [36] for their definitions.

For a stopping time T w.r.t. $(\mathcal{F}_t)_{t=0}^{\infty}$, we define a *level map* $(\mathsf{Lv}_t)_{t=0}^{\infty}$ as a sequence of $\mathcal{F}_t$-measurable functions $\mathsf{Lv}_t : \Omega \to \{0, \dots n\}$ such that $[\![\mathsf{Lv}_t = 0]\!] = [\![T \leq t]\!]$ for each t. We call a pair of a stochastic process and a level map an *instance for* T; just like we construct an MM $\boldsymbol{\eta}$ and a level map Lv as an AST certificate of a pCFG $\mathcal{C}$, we construct an instance for a stopping time T as its AST certificate. We say an instance $((\mathbf{X}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ for T *ranks* $\omega \in \Omega$ in the dimension k at time t when $T(\omega) > t$ and $k = \mathsf{Lv}_t(\omega)$.

For $c > 0$, we say an instance $((\mathbf{X}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ satisfies the *c-ranking condition* if, for each $t \in \mathbb{N}$, $\omega \in [\![\mathsf{Lv}_t \neq 0]\!]$, and $k \in \{1, \dots, \mathsf{Lv}_t(\omega)\}$, we have:

$$\mathbb{E}[\mathbf{X}_{t+1}[k] \mid \mathcal{F}_t](\omega) \leq \mathbf{X}_t[k](\omega) - c \cdot \mathbf{1}_{[\![k=\mathsf{Lv}_t]\!]}(\omega) \quad (\mathbb{P}\text{-a.s.}) \quad (1)$$

Here, the function $\mathbb{E}[\mathbf{X}_{t+1}[k] \mid \mathcal{F}_t]$ denotes the *conditional expectation* of $\mathbf{X}_{t+1}[k]$ given $\mathcal{F}_t$, which takes the role of pre-expectation. We mostly let $c = 1$ and simply call it the ranking condition; the only result sensitive to c is Thm. 4.3.

We also define the three different non-negativity conditions for an instance as follows. Here we adopt a slightly general (but essentially the same) variant of strong non-negativity instead, calling it *uniform well-foundedness*; we simply allow the uniform lower bound to be any constant $\bot \in \mathbb{R}$ instead of fixing it to be zero. This makes the later argument simpler.

(UN well-fnd.) $\exists \bot \in \mathbb{R}. \forall t \in \mathbb{N}.$
$\forall \omega \in \Omega. \forall k \in \{1, \dots, n\}.\ \mathbf{X}_t[k](\omega) \geq \bot,$

(LW non-neg.) $\forall t \in \mathbb{N}. \forall \omega \in [\![\mathsf{Lv}_t \neq 0]\!].$
$\forall k \in \{1, \dots, \mathsf{Lv}_t(\omega)\}.\ \mathbf{X}_t[k](\omega) \geq 0,$

(SC non-neg.)
$\forall t \in \mathbb{N}. \forall \omega \in [\![\mathsf{Lv}_t \neq 0]\!].\ \mathbf{X}_t[\mathsf{Lv}_t(\omega)](\omega) \geq 0.$

Definition 3.4 ((UN/LW/SC)-LexRSM)**.** *Suppose the following are given: a probability space $(\Omega, \mathcal{F}, \mathbb{P})$; a filtration $(\mathcal{F}_t)_{t=0}^{\infty}$ on $\mathcal{F}$; and a stopping time T w.r.t. $(\mathcal{F}_t)_{t=0}^{\infty}$. An instance $\mathcal{I} = ((\mathbf{X}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ is called a* UNiformly well-founded LexRSM (UN-LexRSM) *for T with the bottom $\bot \in \mathbb{R}$ and a constant $c \in \mathbb{R}$ if (a) $(\mathbf{X}_t)_{t=0}^{\infty}$ is adapted to $(\mathcal{F}_t)_{t=0}^{\infty}$; (b) for each $t \in \mathbb{N}$ and $1 \leq k \leq n$, the expectation of $\mathbf{X}_t[k]$ exists; (c) $\mathcal{I}$ satisfies the c-ranking condition; and (d) $\mathcal{I}$ is uniformly well-founded with the bottom $\bot$. We define* LW-LexRSM *and* SC-LexRSM *by changing (d) with LW and SC non-negativity, respectively.*

We mostly assume $c = 1$ and omit to mention the constant. UN-LexRSM is known to be sound [2]; meanwhile, LW and SC-LexRSM are generally unsound [15, 20]. We still mention the latter two as parts of sound LexRSMs.

From RSM maps to RSMs. Let $\boldsymbol{\eta}$ be an MM over a pCFG $\mathcal{C}$ with a level map Lv. Together with a Δ-deterministic scheduler σ and initial state s_I, it *induces* an instance $((\mathbf{X}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ over the dynamics of $\mathcal{C}$, by letting $\mathbf{X}_t(s_0 s_1 \ldots) = \boldsymbol{\eta}(s_t)$; it describes the behavior of $\boldsymbol{\eta}$ and Lv through executing $\mathcal{C}$ from s_I under σ. Properties of $\boldsymbol{\eta}$ such as ranking condition or non-negativity are inherited to the induced instance (if the expectation of $\mathbf{X}_t[k]$ exists for each t, k). For example, an instance induced by an ST-LexRSM map is an UN-LexRSM with $\bot = 0$.

Non-probabilistic settings, and instantiation of SC-LexRF. The key question in this paper is to find a LexRSM notion that instantiates SC non-negative LexRF (or SC-LexRF for short); that is, we would like to find a LexRSM notion whose conditions are satisfied by

SC-LexRSM[†5] in the *non-probabilistic setting*, which we formalize as follows. We say a pCFG is a *(non-probabilistic) CFG* if (a) δ is Dirac for each $(\ell, \delta) \in \Delta$, and (b) $\Delta_p = \emptyset$; this roughly means that a CFG is a model of a PP without '**if prob**(p)' and '**sample**(d)'. We say a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is *trivial* if Ω is a singleton, say $\{\omega\}$.

4 Fixable LexRSMs

In §4-6 we give our novel technical notions and results. In this section, we will introduce the notion of fixability and related results. Here we focus on technical rigorousness and conciseness, see §2 for the underlying intuition. Proofs are given in appendices of [36].

We begin with the formal definition of ε-fixability.

Remark 4.1. *As in Footnote 2, our formal definitions of fixability in this section slightly differ from an informal explanation in §2. One difference is that the ε-fixing in Def. 4.2 changes the value of a LexRSM at dimension k whenever it is negative* or *k is strictly on the right to the ranking dimension. This modification is necessary to prove Thm. 4.4. Another is that we define fixability as the notion for an instance $\mathcal{I}$, rather than for an MM $\boldsymbol{\eta}$. While the latter can be also done in an obvious way (as informally done in §2), we do not formally do that because it is not necessary for our technical development. One can "fix" the argument in §2 into the one over instances by translating "fixability of $\boldsymbol{\eta}$" to "fixability of an instance induced by $\boldsymbol{\eta}$".*

Definition 4.2 (ε-fixing of an instance)**.** *Let $\mathcal{I} = ((\mathbf{X}_t)_{t=0}^{\infty}), (\mathsf{Lv}_t)_{t=0}^{\infty})$ be an instance for a stopping time T, and let $\varepsilon > 0$. The* ε-fixing *of $\mathcal{I}$ is another instance $\tilde{\mathcal{I}} = ((\tilde{\mathbf{X}}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ for T, where*

$$\tilde{\mathbf{X}}_t[k](\omega) = \begin{cases} -\varepsilon & \text{if } \mathbf{X}_t[k](\omega) < 0 \text{ or } k > \mathsf{Lv}_t(\omega), \\ \mathbf{X}_t[k](\omega) & \text{otherwise.} \end{cases}$$

We say an SC-LexRSM $\mathcal{I}$ is ε-fixable*, or call it an* ε-fixable LexRSM*, if its ε-fixing $\tilde{\mathcal{I}}$ is an UN-LexRSM with the bottom $\bot = -\varepsilon$.*

Observe that the ε-fixing of any instance is uniformly well-founded with the bottom $\bot = -\varepsilon$, so the ε-fixability only asks if the ranking condition is preserved through ε-fixing. Also, observe that the soundness of ε-fixable LexRSM immediately follows from that of UN-LexRSM [2].

While we do not directly use ε-fixability as a technical tool, the two theorems below show its conceptual value. The first one answers our key problem: ε-fixable LexRSM instantiates SC-LexRF with sufficiently large ε.

Theorem 4.3 (fixable LexRSM instantiates SC-LexRF)**.** *Suppose $\mathcal{I} = ((\boldsymbol{x}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ is an SC-LexRSM for a stopping time T over the trivial probability space with a constant c, and let $\varepsilon \geq c$.*

†5 One would perhaps expect to see "SC-Lex*RF*" here; such a change does not make a difference under a canonical definition of SC-LexRF, so we define the notion of instantiation in this way to save space. See also [36, Appendix A].

Then $\mathcal{I}$ is ε-fixable. □

The second theorem offers a formal comparison between ε-fixable LexRSM and the state-of-the-art LexRSM variant in the literature, namely GLexRSM [15]. We show the former subsumes the latter. In our terminology, GLexRSM is LW-LexRSM that also satisfies the following *expected leftward non-negativity*:

$$\forall t \in \mathbb{N}.\forall \omega \in [\![\mathsf{Lv}_t \neq 0]\!].\forall k \in \{1,\dots,\mathsf{Lv}_t(\omega)\}.$$
$$\mathbb{E}[\mathbf{1}_{[\![k>\mathsf{Lv}_{t+1}]\!]} \cdot \mathbf{X}_{t+1}[k] \mid \mathcal{F}_t](\omega) \geq 0.$$

We note that our result can be also seen as an alternative proof of the soundness of GLexRSM [15, Thm. 1]. Our proof is also significantly simpler than the original one, as the former utilizes the soundness of UN-LexRSM as a lemma, while the latter does the proof "from scratch".

Theorem 4.4 (fixable LexRSM generalizes GLexRSM)**.** *Suppose $\mathcal{I}$ is a GLexRSM for a stopping time T. Then $\mathcal{I}$ is ε-fixable for any $\varepsilon > 0$.* □

Now we move on to a refined variant, (ε,γ)-fixability. Before its formal definition, we give a theorem that justifies the partial waiving of the ranking condition described in §2. Below, $\overset{\infty}{\exists} t.\varphi_t$ stands for $\forall k \in \mathbb{N}.\exists t \in \mathbb{N}.[t > k \wedge \varphi_t]$.

Theorem 4.5 (relaxation of the UN-LexRSM condition)**.** *Suppose the following are given: a probability space $(\Omega,\mathcal{F},\mathbb{P})$; a filtration $(\mathcal{F}_t)_{t=0}^{\infty}$ on $\mathcal{F}$; and a stopping time T w.r.t. $(\mathcal{F}_t)_{t=0}^{\infty}$. Let $\mathcal{I} = ((\mathbf{X}_t)_{t=0}^{\infty},(\mathsf{Lv}_t)_{t=0}^{\infty})$ be an instance for T, and let $\bot \in \mathbb{R}$. For each $k \in \{1,\dots,n\}$, let $(\varphi_{t,k})_{t=0}^{\infty}$ be a sequence of predicates over Ω such that*

$$\overset{\infty}{\exists} t.\varphi_{t,k}(\omega) \Rightarrow$$
$$\overset{\infty}{\exists} t.[\mathbf{X}_t[k](\omega) = \bot \vee k > \mathsf{Lv}_t(\omega)] \quad (\mathbb{P}\text{-a.s.}) \qquad (2)$$

Suppose $\mathcal{I}$ is an UN-LexRSM with the bottom $\bot$ except that, instead of the ranking condition, $\mathcal{I}$ satisfies the inequality (1) only for $t \in \mathbb{N}$, $k \in \{1,\dots,n\}$, and $\omega \in [\![k \leq \mathsf{Lv}_t \wedge \neg(\mathbf{X}_t[k] > \bot \wedge \varphi_{t,k})]\!]$ (with $c = 1$). Then T is AST w.r.t. $\mathbb{P}$. □

The correspondence between the argument in §2 and Thm. 4.5 is as follows. The predicate $\varphi_{t,k}$ is an abstraction of the situation "we are at a coin-tossing state at time t in the k-th dimension"; and the condition (2) corresponds to the infinite coin-tossing argument (for a given k, if $\varphi_{t,k}$ is satisfied at infinitely many t, then the ranking in the k-th dimension is "done" infinitely often, with probability 1). Given these, Thm. 4.5 says that the ranking condition of UN-LexRSM can be waived over $[\![\mathbf{X}_t[k] > \bot \wedge \varphi_{t,k}]\!]$. In particular, the theorem amounts to the soundness of UN-LexRSM when $\varphi_{t,k} \equiv \mathit{false}$ for each t and k.

Based on Theorem 4.5, we introduce (ε,γ)-fixability as follows. There, $\mathbb{P}[\varphi \mid \mathcal{F}'] := \mathbb{E}[\mathbf{1}_{[\![\varphi]\!]} \mid \mathcal{F}']$ is the *conditional probability* of satisfying φ given $\mathcal{F}'$.

Definition 4.6 ((ε,γ)-fixability)**.** *Let $\mathcal{I} = ((\mathbf{X}_t)_{t=0}^{\infty},(\mathsf{Lv}_t)_{t=0}^{\infty})$ be an instance for T, and let $\gamma \in (0,1)$. We call $\mathcal{I}$ a* γ-relaxed UN-LexRSM *for T if $\mathcal{I}$ satisfies the properties in Thm. 4.5, where $\varphi_{t,k}$ is as follows:*

$$\varphi_{t,k}(\omega) \equiv \mathbb{P}[\mathbf{X}_{t+1}[k] = \bot \mid \mathcal{F}_t](\omega) \geq \gamma. \qquad (3)$$

We say $\mathcal{I}$ is (ε,γ)-fixable *if its ε-fixing $\tilde{\mathcal{I}}$ is a γ-relaxed UN-LexRSM.*

The predicate $\varphi_{t,k}(\omega)$ in (3) is roughly read "the ranking by $(\mathbf{X}_t)_{t=0}^{\infty}$ is done at time $t+1$ in dimension k with probability γ or higher, given the information about ω at t". This predicate satisfies Condition (2); hence we have the following corollary, which is the key to the soundness of *lazy LexRSM* in §5.

Corollary 4.7 (soundness of (ε,γ)-fixable instances)**.** *Suppose there exists an instance $\mathcal{I}$ over $(\Omega,\mathcal{F},\mathbb{P})$ for a stopping time T that is (ε,γ)-fixable for any $\varepsilon > 0$ and $\gamma \in (0,1)$. Then T is AST w.r.t. $\mathbb{P}$.* □

5 Lazy LexRSM and Its Soundness

Here we introduce another LexRSM variant, *Lazy LexRSM* (LLexRSM). We need this variant for our LexRSM synthesis algorithm; while ε-fixable LexRSM theoretically answers our key question, it

is not amenable to LP-based synthesis algorithms because its case distinction makes the resulting constraint nonlinear.

We define LLexRSM map as follows; see *Contributions* in §1 for its intuitive meaning with an example. The definition for an instance is in [36, Appendix C].

Definition 5.1 (LLexRSM map)**.** *Fix a pCFG $\mathcal{C}$ with an invariant I. Let $\boldsymbol{\eta}$ be an MM associated with a level map Lv. The MM $\boldsymbol{\eta}$ is called a* Lazy LexRSM map (LLexRSM map) *over $\mathcal{C}$ supported by I if it is an SC-LexRSM map over $\mathcal{C}$ supported by I, and satisfies* stability at negativity *defined as follows:*

$$\forall \tau \neq \tau_{\mathrm{out}}.\forall s \in [\![I \wedge G(\tau)]\!].\forall k \in \{1,\dots,\mathsf{Lv}(\tau)-1\}.$$
$$\boldsymbol{\eta}[k](s) < 0 \Rightarrow \forall s' \in \mathrm{succ}_\tau(s).$$
$$\left[\boldsymbol{\eta}[k](s') < 0 \vee k > \max_{\tau' : s' \in [\![G(\tau')]\!]} \mathsf{Lv}(\tau')\right].$$

We first observe LLexRSM also answers our key question.

Theorem 5.2 (LLexRSM instantiates SC-LexRF)**.** *Suppose $\boldsymbol{\eta}$ is an SC-LexRSM over a non-probabilistic CFG $\mathcal{C}$ supported by an invariant I, with a level map Lv. Then $\boldsymbol{\eta}$ is stable at negativity under I and Lv, and hence, $\boldsymbol{\eta}$ is an LLexRSM map over $\mathcal{C}$ supported by I, with Lv.* □

Below we give the soundness result of LLexRSM map. We first give the necessary assumptions on pCFGs and MMs, namely *linearity* and *well-behavedness*. we say a pCFG is *linear* if the update element of each $\tau \in \Delta_d$ is a linear function (this corresponds to the restriction on PPs to the linear ones); and an MM $\boldsymbol{\eta}$ is *linear* if $\lambda \boldsymbol{x}.\boldsymbol{\eta}(\ell, \boldsymbol{x})$ is linear for each $\ell \in L$. We say a pCFG is *well-behaved* if its variable samplings are done via *well-behaved distributions*, which roughly means that their tail probabilities vanish to zero toward infinity quickly enough. Its formal definition is given in [36, Def. C.4], which is somewhat complex; an important fact from the application perspective is that the class of such distributions covers all distributions with bounded supports *and* some distributions with unbounded supports such as the normal distributions [36, Prop. C.6].

Possibly negative (Lex)RSM typically requires some restriction on variable samplings of pCFG (e.g., the *integrability* in [15]) so that the pre-expectation is well-defined.

The crucial part of the soundness proof is the following theorem, where (ε, γ)-fixability takes the key role. Its full proof is given in [36, Appendix C].

Theorem 5.3. *Let $\boldsymbol{\eta} : \mathcal{S} \to \mathbb{R}^n$ be a linear LLexRSM map for a linear, well-behaved pCFG $\mathcal{C}$. Then for any Δ-deterministic scheduler σ and initial state s_I of $\mathcal{C}$, the induced instance is (ε, γ)-fixable for some $\varepsilon > 0$ and $\gamma \in (0, 1)$.*

Proof (sketch). We can show that the ε-fixing $\tilde{\mathcal{I}} = ((\tilde{\mathbf{X}}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ of an induced instance $\mathcal{I} = ((\mathbf{X}_t)_{t=0}^{\infty}, (\mathsf{Lv}_t)_{t=0}^{\infty})$ almost-surely satisfies the inequality (1) of the ranking condition for each t, ω, and k such that $\tilde{\mathbf{X}}_t[k](\omega) = -\varepsilon$ and $1 \leq k \leq \mathsf{Lv}_t(\omega)$ [36, Prop. C.2]. Thus it suffices to show, for each ω, k, and t such that $\tilde{\mathbf{X}}_t[k](\omega) \geq 0$ and $1 \leq k \leq \mathsf{Lv}_t(\omega)$, either $\tilde{\mathcal{I}}$ satisfies the inequality (1) or (1) as a requirement on $\tilde{\mathcal{I}}$ is waived due to the γ-relaxation.

Now take any such t, ω, and k, and suppose the run ω reads the program line *prog* at time t. Then we can show the desired property by a case distinction over *prog* as follows. Here, recall ω is a sequence $s_0 s_1 \dots s_t s_{t+1} \dots$ of program states; we defined $\mathbf{X}_t$ by $\mathbf{X}_t[k](\omega) = \boldsymbol{\eta}[k](s_t)$; and $\mathbb{E}[\mathbf{X}_{t+1}[k] \mid \mathcal{F}_t](\omega)$ is the expectation of $\boldsymbol{\eta}[k](s')$, where s' is the successor state of $s_0 \dots s_t$ under σ (which is not necessarily s_{t+1}). Also observe the requirement (1) on $\tilde{\mathcal{I}}$ is waived for given t, ω, and k when the value of $\boldsymbol{\eta}[k](s')$ is negative with the probability γ or higher.

1. Suppose *prog* is a non-probabilistic program line, e.g., '$x_i := f(\boldsymbol{x})$' or '**while** φ **do**'. Then

the successor state s' of s_t is unique. If $\boldsymbol{\eta}[k](s')$ is non-negative, then we have $\mathbb{E}[\tilde{\mathbf{X}}_{t+1}[k] \mid \mathcal{F}_t](\omega) = \mathbb{E}[\mathbf{X}_{t+1}[k] \mid \mathcal{F}_t](\omega)$, so the inequality (1) is inherited from $\mathcal{I}$ to $\tilde{\mathcal{I}}$; if negative, then the requirement (1) on $\tilde{\mathcal{I}}$ is waived. The same argument applies to '**if** $\star$ **then**' (recall $\mathcal{I}$ is induced from a Δ-deterministic scheduler).

2. Suppose $prog \equiv$ '**if prob**(p) **then**'. By letting γ strictly smaller than p, we see either $\boldsymbol{\eta}[k](s')$ is never negative, or it is negative with a probability more than γ. Thus we have the desired property for a similar reason to Case 1 (we note this argument requires p to be a constant).
3. Suppose $prog \equiv$ '$x_i :=$ **sample**(d)'. We can show the desired property by taking a sufficiently small γ; roughly speaking, the requirement (1) on $\tilde{\mathcal{I}}$ is waived unless the chance of $\boldsymbol{\eta}[k](s')$ being negative is very small, in which case the room for "ill" exploitation is so small that the inequality (1) is inherited from $\mathcal{I}$ to $\tilde{\mathcal{I}}$. Almost the same argument applies to '$x_i :=$ **ndet**(D)'.

We note, by the finiteness of program locations L and transitions Δ, we can take $\gamma \in (0, 1)$ that satisfies all requirements above simultaneously. □

Now we have soundness of LLexRSM as the following theorem, which is almost an immediate consequence of Thm. 5.3 and Cor. 4.7.

Theorem 5.4 (soundness of linear LLexRSM map over linear, well-behaved pCFG)**.** *Let $\mathcal{C}$ be a linear, well-behaved pCFG, and suppose there is a linear LLexRSM map over $\mathcal{C}$ (supported by any invariant). Then $\mathcal{C}$ is AST.* □

6 Automated Synthesis Algorithm of LexRSM

In this section, we introduce a synthesis algorithm of LLexRSM for automated AST verification of linear PPs. It synthesizes a linear MM in a certain subclass of LLexRSMs. We first define the subclass, and then introduce our algorithm.

Our algorithm is a variant of *linear template-based synthesis*. There, we fix a linear MM $\boldsymbol{\eta}$ with unknown coefficients (i.e., *the linear template*), and consider an assertion "$\boldsymbol{\eta}$ is a certificate of AST"; for example, in the standard 1-dimensional RSM synthesis, the assertion is "η is an RSM map". We then reduce this assertion into a set of linear constraints via *Farkas' Lemma* [34]. These constraints constitute an LP problem with an appropriate objective function. A certificate is synthesized, if feasible, by solving this LP problem. The reduction is standard, so we omit the details; see e.g. [35].

Subclass of LLexRSM for automated synthesis. While LLexRSM resolves the major issue that fixable LexRSM confronts toward its automated synthesis, we still need to tweak the notion a bit more, as the stability at negativity condition involves the value of an MM $\boldsymbol{\eta}$ in its antecedent part (i.e., it says "whenever $\boldsymbol{\eta}[k]$ is negative for some k..."); this makes the reduced constraints via Farkas' Lemma nonlinear. Therefore, we augment the condition as follows.

Definition 6.1 (MCLC)**.** *Let $\boldsymbol{\eta} : \mathcal{S} \to \mathbb{R}^n$ be an MM supported by an invariant I, with a level map* Lv*. We say $\boldsymbol{\eta}$ satisfies the* multiple-choice leftward condition *(MCLC) if, for each $k \in \{1, \dots, n\}$, it satisfies either (4) or (5) below:*

$$\forall \tau \in [\![k < \mathsf{Lv}]\!].\forall s \in [\![I \wedge G(\tau)]\!].\ \boldsymbol{\eta}[k](s) \geq 0, \tag{4}$$

$$\forall \tau \in [\![k < \mathsf{Lv}]\!].\forall s \in [\![I \wedge G(\tau)]\!].\forall s' \in \mathrm{succ}_\tau(s).\ \boldsymbol{\eta}[k](s') \leq \boldsymbol{\eta}[k](s). \tag{5}$$

Condition (4) is nothing but the non-negativity condition in dimension k. Condition (5) augments the ranking condition in the strict leftward of the ranking dimension (a.k.a. the *unaffecting* condition) so that the value of $\boldsymbol{\eta}[k]$ is non-increasing in the worst-case. MCLC implies stability at negativity; hence, by Thm. 5.4, linear SC-LexRSM maps

with MCLC certify AST of linear, well-behaved pCFGs. They also instantiate SC-LexRFs as follows.

Theorem 6.2 (SC-LexRSM maps with MCLC instantiate SC-LexRFs)**.** *Suppose $\boldsymbol{\eta}$ is an SC-LexRSM map over a non-probabilistic CFG $\mathcal{C}$ supported by I, with* Lv*. Then $\boldsymbol{\eta}$ satisfies MCLC under I and* Lv*.* □

The algorithm. Our LexRSM synthesis algorithm mostly resembles the existing ones [2, 15], so we are brief here; a line-to-line explanation with a pseudocode is in [36, Appendix D].

The algorithm receives a pCFG $\mathcal{C}$ and an invariant I, and attempts to construct a SC-LexRSM with MCLC over $\mathcal{C}$ supported by I. The construction is iterative; at the k-th iteration, the algorithm attempts to construct a one-dimensional MM η_k that ranks transitions of $\mathcal{C}$ that are not ranked by the current construction $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_{k-1})$, while respecting MCLC. If the algorithm finds η_k that ranks at least one new transition, then it appends η_k to $\boldsymbol{\eta}$ and goes to the next iteration; otherwise, it reports a failure. Once $\boldsymbol{\eta}$ ranks all transitions, the algorithm reports a success, returning $\boldsymbol{\eta}$ as an AST certificate of $\mathcal{C}$.

Our algorithm attempts to construct η_k in two ways, by adopting either (4) or (5) as the leftward condition at the dimension k. The attempt with the condition (4) is done in the same manner as existing algorithms [2, 15]; we require η_k to rank the unranked transitions *as many as possible*. The attempt with the condition (5) is slightly nontrivial; the algorithm demands a user-defined parameter $\mathrm{Class}(U) \subseteq 2^U$ for each $U \subseteq \Delta \setminus \{\tau_{\mathrm{out}}\}$. The parameter $\mathrm{Class}(U)$ specifies which set of transitions the algorithm should try to rank, given the set of current unranked transitions U; that is, for each $\mathcal{T} \in \mathrm{Class}(U)$, the algorithm attempts to find η_k that *exactly* ranks transitions in $\mathcal{T}$.

There are two canonical choices of $\mathrm{Class}(U)$. One is $2^U \setminus \{\emptyset\}$, the brute-force trial; the resulting algorithm does not terminate in polynomial time, but ranks the maximal number of transitions (by trying each $\mathcal{T}$ in the descending order w.r.t. $|\mathcal{T}|$). This property makes the algorithm complete. Another choice is the singletons of U, i.e., $\{\{\tau\} \mid \tau \in U\}$; while the resulting algorithm terminates in polynomial time, it lacks the maximality property. It is our future work to verify if there is a polynomial complete instance of our proposed algorithm. Still, any instance of it is complete over yet another class of LLexRSMs, namely linear LW-LexRSMs. For a formal statement and its proof, see [36, Thm. D.1].

7 Experiments

We performed experiments to evaluate the performance of our proposed algorithm. The implementation is publicly available[†6].

Our evaluation criteria are twofold: one is how the relaxed non-negativity condition of our LexRSM—SC non-negativity and MCLC—improves the applicability of the algorithm, compared to other existing non-negativity conditions. To this end, we consider two baseline algorithms.

(a) The algorithm *STR*: This is the one proposed in [2], which synthesizes an ST-LexRSM. We use the implementation provided by the authors [3].

(b) The algorithm *LWN*: This synthesizes an LW-LexRSM. LWN is realized as an instance of our algorithm with $\mathrm{Class}(U) = \emptyset$. We use LWN as a proxy of the synthesis algorithm of *GLexRSM* [16, Alg. 2], whose implementation does not seem to exist. We note [16, Alg. 2] synthesizes an LW-LexRSM with some additional conditions; therefore, it is no less restrictive than LWN.

Another criterion is how the choice of $\mathrm{Class}(U)$ af-

†6 `https://doi.org/10.5281/zenodo.10937558`

fects the performance of our algorithm. To this end, we consider two instances of it: (a) *Singleton Multiple Choice* (SMC), given by $\mathrm{Class}(U) = \{\{\tau\} \mid \tau \in U\}$; and (b) *Exhaustive Multiple Choice* (EMC), given by $\mathrm{Class}(U) = 2^U \setminus \emptyset$. SMC runs in PTIME, but we do not know if it is complete; EMC does not run in PTIME, but is complete.

We use benchmarks from [2], which consist of non-probabilistic programs collected in [4] and their probabilistic modifications. The modification is done in two different ways: (a) while loops "**while** φ **do** P **od**" are replaced with probabilistic ones "**while** φ **do** (**if prob**(0.5) **then** P **else skip fi**) **od**"; (b) in addition to (a), variable assignments "$x := f(\boldsymbol{x}) + a$" are replaced with "$x := f(\boldsymbol{x}) + \mathit{Unif}[a-1, a+1]$". We include non-probabilistic programs in our benchmark set because the "problematic program structure" that hinders automated LexRSM synthesis already exists in non-probabilistic programs (cf. our explanation to Fig. 2). We also tried two PPs from [15, Fig. 1], which we call *counterexStr1* and *counterexStr2*.

We implemented our algorithm upon [2], which is available at [3]. Similar to [2], our implementation works as follows: (1) it receives a linear PP as an input, and translates it into a pCFG $\mathcal{C}$; (2) it generates an invariant for $\mathcal{C}$; (3) via our algorithm, it synthesizes an SC-LexRSM map with MCLC. Invariants are generated by ASPIC [19], and all LP problems are solved by CPLEX [25].

Results. In 135 benchmarks from 55 models, STR succeeds in 98 cases, LWN succeeds in 105 cases while SMC and EMC succeed in 119 cases (we did not run STR for *counterexStr1* because it involves a sampling from an unbounded support distribution, which is not supported by STR). Table 1 summarizes the cases where we observe differences in the feasibility of algorithms. As theoretically anticipated, LWN always succeeds in finding a LexRSM whenever STR does; the same relation is observed between SMC vs. LWN and EMC vs. SMC. In most cases, STR, LWN, and SMC return an output within a second[†7], while EMC suffers from an exponential blowup when it attempts to rank transitions with Condition (5) in Def. 6.1. The full results are in [36, Appendix E].

On the first evaluation criterion, the advantage of the relaxed non-negativity is evident: SMC/EMC have unique successes vs. STR on 21 programs ($21/135 = 15.6\%$ higher success rate) from 16 different models; SMC/EMC also have unique successes vs. LWN in 14 programs ($14/135 = 10.4\%$ higher success rate) from 12 models. This result shows that the program structure we observed in Fig. 2 appears in various programs in the real world.

On the second criterion, EMC does not have any unique success compared to SMC. This result suggests that SMC can be the first choice as a concrete instance of our proposed algorithm. Indeed, we suspect that SMC is actually complete—verifying its (in)completeness is a future work. For some programs, EMC found a LexRSM with a smaller dimension than SMC.

Interestingly, LWN fails to find a LexRSM for *counterexStr2*, despite it being given in [15] as a PP for which a GLexRSM (and hence, an LW non-negative LexRSM) exists. This happens because the implementation in [3] translates the PP into a pCFG with a different shape than the one in [15] (for the latter, a GLexRSM indeed exists); the former possesses a similar structure as in Fig. 2 because different locations are assigned for the while loop and if branch. This demonstrates the advantage of our algorithm from another point of view, i.e., robustness against different translations of PPs.

†7 There was a single example for which more time was spent, due to a larger size.

Benchmark spec.			Synthesis result				Benchmark spec.			Synthesis result			
			Baselines		Our algs.					Baselines		Our algs.	
Model	p.l.	p.a.	STR	LWN	SMC	EMC	Model	p.l.	p.a.	STR	LWN	SMC	EMC
complex	-	-	×	×	7	5	serpent	-	-	×	×	3	3
complex	√	-	×	×	7	5	speedDis1	-	-	×	×	4	4
complex	√	√	×	×	3	3	speedDis2	-	-	×	×	4	4
cousot9	-	-	×	3	3	3	spdSimMul	-	-	×	×	4	4
cousot9	√	-	×	×	4	4	spdSimMulDep	-	-	×	×	4	4
loops	-	-	×	×	4	3	spdSglSgl2	√	√	×	×	5	5
nestedLoop	√	√	×	×	4	3	speedpldi3	-	-	×	3	3	3
realheapsort	-	-	×	3	3	3	speedpldi3	√	-	×	×	4	4
RHS_step1	-	-	×	3	3	3	counterexStr1	-	√	N/A	3	3	3
RHS_step1	√	√	×	3	3	3	counterexStr2	-	√	×	×	4	4
realshellsort	√	√	×	2	2	2							

表 1 The list of benchmarks in which a feasibility difference is observed between baselines and proposed algorithms. Ticks in "p.l." and "p.a." indicate the benchmark has a probabilistic loop and assignment, respectively. Numbers in the result indicate that the algorithm found a LexRSM with that dimension; the crosses indicate failures; "N/A" means we did not run the experiment.

8 Related Work

There is a rich body of studies in 1-dimensional RSM [12–14, 17, 20–23, 28–30], while lexicographic RSM is relatively new [2, 15]. Our paper generalizes the latest work [15] on LexRSM as follows: (a) *Soundness of LexRSM as a stochastic process:* soundness of ε-fixable LexRSMs (Def. 4.2) generalizes [15, Thm. 1] in the sense that every GLexRSM is ε-fixable for any $\varepsilon > 0$ (Thm. 4.4); (b) *Soundness of LexRSM as a function on program states:* our result (Thm. 5.4) generalizes [15, Thm. 2] under the linearity and well-behavedness assumptions; (c) *Soundness and completeness of LexRSM synthesis algorithms:* our result generalizes the results for one of two algorithms in [15] that assumes boundedness assumption on assignment distribution [15, Thm. 3].

The work [24] also considers a relaxed non-negativity of RSMs. Their *descent supermartingale*, which acts on while loops, requires well-foundedness only at every entry into the loop body. A major difference from our LexRSM is that they only consider 1-dimensional RSMs; therefore, the problem of relaxing the LW non-negativity does not appear in their setting. Compared with their RSM, our LexRSM has an advantage in verifying PPs with a structure shown in Fig. 2, where the value of our LexRSM can be arbitrarily small upon the loop entrance (at some dimension; see η_2 at ℓ_1 in Fig. 2).

The work [29] extends the applicability of standard RSM on a different aspect from LexRSM. The main feature of their RSM is that it can verify AST of the symmetric random walk. While our LexRSM cannot verify AST of this process, the RSM by [29] is a 1-dimensional one, which typically struggles on PPs with nested structures. Such a difference can be observed from the experiment result in [31] (compare [31, Table 2] and *nested_loops*, *sequential_loops* in [31, Table 1]).

参 考 文 献

[1] : Ultimate Automizer,.

[2] Agrawal, S., Chatterjee, K., and Novotný, P.: Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs, *Proc. ACM Program. Lang.*, Vol. 2, No. POPL(2018), pp. 34:1–34:32.

[3] Agrawal, S., Chatterjee, K., and Novotný, P.: Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs: Implementation, 2018.

[4] Alias, C., Darte, A., Feautrier, P., and Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs, *Static Analysis: 17th International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010. Proceedings 17*, Springer, 2010, pp. 117–133.

[5] Ash, R. and Doléans-Dade, C.: *Probability and Measure Theory*, Harcourt/Academic Press, 2000.

[6] Barthe, G., Gaboardi, M., Grégoire, B., Hsu, J., and Strub, P.-Y.: Proving differential privacy via probabilistic couplings, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, 2016, pp. 749–758.

[7] Barthe, G., Gaboardi, M., Hsu, J., and Pierce, B.: Programming language techniques for differential privacy, *ACM SIGLOG News*, Vol. 3, No. 1(2016), pp. 34–53.

[8] Ben-Amram, A. M. and Genaim, S.: Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, Kroening, D. and Pasareanu, C. S.(eds.), Lecture Notes in Computer Science, Vol. 9207, Springer, 2015, pp. 304–321.

[9] Bertsekas, D. P. and Shreve, S. E.: *Stochastic Optimal Control: The Discrete-Time Case*, Athena Scientific, 2007.

[10] Bradley, A. R., Manna, Z., and Sipma, H. B.: Linear Ranking with Reachability, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, Etessami, K. and Rajamani, S. K.(eds.), Lecture Notes in Computer Science, Vol. 3576, Springer, 2005, pp. 491–504.

[11] Canal, G., Cashmore, M., Krivić, S., Alenyà, G., Magazzeni, D., and Torras, C.: Probabilistic planning for robotics with ROSPlan, *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019, London, UK, July 3–5, 2019, Proceedings, Part I 20*, Springer, 2019, pp. 236–250.

[12] Chakarov, A. and Sankaranarayanan, S.: Probabilistic program analysis with martingales, *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, Springer, 2013, pp. 511–526.

[13] Chatterjee, K., Fu, H., and Goharshady, A. K.: Termination analysis of probabilistic programs through Positivstellensatz's, *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I 28*, Springer, 2016, pp. 3–22.

[14] Chatterjee, K., Fu, H., Novotnỳ, P., and Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016, pp. 327–342.

[15] Chatterjee, K., Goharshady, E. K., Novotný, P., Zárevúcky, J., and Zikelic, D.: On Lexicographic Proof Rules for Probabilistic Termination, *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, Huisman, M., Pasareanu, C. S., and Zhan, N.(eds.), Lecture Notes in Computer Science, Vol. 13047, Springer, 2021, pp. 619–639.

[16] Chatterjee, K., Goharshady, E. K., Novotný, P., Zárevúcky, J., and Zikelic, D.: On Lexicographic Proof Rules for Probabilistic Termination, *CoRR*, Vol. abs/2108.02188(2021).

[17] Chatterjee, K., Novotnỳ, P., and Zikelic, D.: Stochastic invariants for probabilistic termination, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 2017, pp. 145–160.

[18] Dubhashi, D. P. and Panconesi, A.: *Concentration of measure for the analysis of randomized algorithms*, Cambridge University Press, 2009.

[19] Feautrier, P. and Gonnord, L.: Accelerated invariant generation for C programs with Aspic and C2fsm, *Electronic Notes in Theoretical Computer Science*, Vol. 267, No. 2(2010), pp. 3–13.

[20] Ferrer Fioriti, L. M. and Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 489–501.

[21] Fu, H. and Chatterjee, K.: Termination of nondeterministic probabilistic programs, *Verification, Model Checking, and Abstract Interpretation: 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13–15, 2019, Proceedings 20*, Springer, 2019, pp. 468–490.

[22] Giesl, J., Giesl, P., and Hark, M.: Computing expected runtimes for constant probability programs, *Automated Deduction–CADE 27: 27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings 27*, Springer, 2019, pp. 269–286.

[23] Huang, M., Fu, H., and Chatterjee, K.: New approaches for almost-sure termination of probabilistic programs, *Programming Languages and Systems: 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2–6, 2018, Proceedings 16*, Springer, 2018, pp. 181–201.

[24] Huang, M., Fu, H., Chatterjee, K., and Goharshady, A. K.: Modular verification for almost-sure termination of probabilistic programs, *Proc. ACM Program. Lang.*, Vol. 3, No. OOPSLA(2019), pp. 129:1–129:29.

[25] IBM: IBM ILOG CPLEX 12.7 User's Manual (IBM ILOG CPLEX Division, Incline Village, NV), (2017).

[26] Karp, R. M.: An introduction to randomized algorithms, *Discrete Applied Mathematics*, Vol. 34, No. 1-3(1991), pp. 165–201.

[27] Lobo-Vesga, E., Russo, A., and Gaboardi, M.: A programming language for data privacy with accuracy estimations, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 43, No. 2(2021), pp. 1–42.

[28] McIver, A. and Morgan, C.: A new rule for almost-certain termination of probabilistic-and demonic programs, *arXiv preprint arXiv:1612.01091*, (2016).

[29] McIver, A., Morgan, C., Kaminski, B. L., and Katoen, J.-P.: A new proof rule for almost-sure termination, *Proceedings of the ACM on Programming Languages*, Vol. 2, No. POPL(2017), pp. 1–28.

[30] Moosbrugger, M., Bartocci, E., Katoen, J.-P., and Kovács, L.: Automated termination analysis of polynomial probabilistic programs, *Programming Languages and Systems: 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings 30*, Springer International Publishing, 2021, pp. 491–518.

[31] Moosbrugger, M., Bartocci, E., Katoen, J., and Kovács, L.: The Probabilistic Termination Tool Amber, *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, Huisman, M., Pasareanu, C. S., and Zhan, N.(eds.), Lecture Notes in Computer Science, Vol. 13047, Springer, 2021, pp. 667–675.

[32] Olmedo, F., Gretz, F., Jansen, N., Kaminski, B. L., Katoen, J., and McIver, A.: Conditioning in Probabilistic Programming, *ACM Trans. Program. Lang. Syst.*, Vol. 40, No. 1(2018), pp. 4:1–4:50.

[33] Parker, D.: Verification of probabilistic real-time systems, *Proc. 2013 Real-time Systems Summer School (ETR'13)*, (2013).

[34] Schrijver, A.: *Theory of linear and integer programming*, John Wiley & Sons, 1998.

[35] Takisaka, T., Oyabu, Y., Urabe, N., and Hasuo, I.: Ranking and Repulsing Supermartingales for Reachability in Randomized Programs, *ACM Trans. Program. Lang. Syst.*, Vol. 43, No. 2(2021), pp. 5:1–5:46.

[36] Takisaka, T., Zhang, L., Wang, C., and Liu, J.: Lexicographic Ranking Supermartingales with Lazy Lower Bounds, *CoRR*, Vol. abs/2304.11363(2024).