Disproving (Positive) Almost-Sure Termination of Probabilistic Term Rewriting via Random Walks*

J.-C. Kassing(✉), H. Nagel, A. Schlecht, and J. Giesl(✉)

RWTH Aachen University, Aachen, Germany

Abstract. While numerous techniques have been developed to automatically prove termination of probabilistic programs, there are only few automated methods to *disprove* their termination. In this paper, we present the first techniques to automatically disprove (positive) almost-sure termination of probabilistic term rewriting. Disproving termination of non-probabilistic systems requires finding a finite representation of an infinite computation, e.g., a loop of the rewrite system. We extend such qualitative techniques to probabilistic term rewriting, where a quantitative analysis is required. In addition to the existence of a loop, we have to count the number of such loops in order to embed suitable random walks into a computation, thereby disproving termination. To evaluate their power, we implemented all our techniques in the tool AProVE.

1 Introduction

While termination is undecidable in general, automatic techniques to prove termination are crucial in many applications. However, proving *non-termination* is equally important, e.g., to detect bugs. While (non-)termination of ordinary programs has been studied for decades, the work on automated termination analysis of *probabilistic programs* is quite recent. Based on *probabilistic choices*, such programs may proceed in several different ways. Probabilistic programming is used to deal with uncertainty in data and has many applications, see [3, 20].

In the probabilistic setting, one usually considers notions like (positive) almost-sure termination. A program is *almost-surely terminating* (`AST`) if every computation terminates with probability 1. A strictly stronger notion is *positive* `AST` (`PAST`) [7, 41], where every computation must have a finite expected number of steps. It is well known that `PAST` implies `AST` but not vice versa.

Term rewriting [2] is a fundamental concept to transform and evaluate expressions, and techniques to analyze termination of term rewrite systems (TRSs) are used for termination analysis of programs in many languages. Term rewriting was adapted to the probabilistic setting in [1, 6, 7]. While techniques to automatically prove `AST` and `PAST` of probabilistic TRSs (PTRSs) were developed in [1, 27–30], up to now there were no approaches for non-termination of PTRSs. In this paper, we develop the first techniques to automatically disprove `AST` and `PAST` of PTRSs.

Contribution. We introduce a general approach to disprove termination of PTRSs by embedding non-terminating random walks into rewrite sequences of

* funded by the DFG Research Training Group 2236 UnRAVeL

PTRSs (Thm. 7). First, we embed random walks via *occurrences* of *loops* (Thm. 11, 15, and 17), which were previously used to disprove termination of TRSs, see, e.g., [16]. Second, we embed random walks via looping *pattern terms* (Thm. 28), which were previously used to prove non-looping non-termination of TRSs [12]. While the original techniques were *qualitative* (*find the existence of a loop*), the probabilistic setting requires *quantitative* analyses (*find the number of possible loops*). We develop a dynamic programming algorithm (Alg. 1) to solve the counting problems that arise in this setting, e.g., to determine how many instantiations of a term t occur in a term s at orthogonal positions. We implemented our techniques in the tool AProVE and demonstrate their applicability by an experimental evaluation.

Related Work. There exist several techniques to prove non-termination of non-probabilistic term rewriting, see, e.g., [12, 13, 16, 39, 40]. Tools analyzing (non-)termination of TRSs participate annually in the *Termination Competition* (*TermComp*) [18]. A related problem to proving non-termination is analyzing reachability (or (in)feasibility) for TRSs, see, e.g., [23, 36, 43, 48].

For imperative programs, there are also numerous techniques for proving non-termination, e.g., [9, 11, 14, 15, 22, 32, 34], and tools for termination analysis of imperative programs compete annually in *TermComp* and in the software verification competition (*SV-COMP*) [5]. There also exist *decision* procedures for termination on certain subclasses of programs, e.g., [8, 24, 25, 35, 45, 47]. To automatically disprove `(P)AST`, we are only aware of approaches for probabilistic *imperative* programs: The technique of [10] works via repulsing supermartingales, and the technique of [44] uses fixpoints and Park induction. Based on [10], martingale-based proof rules which can also disprove `AST` and `PAST` were implemented in the tool Amber [38]. The hardness of analyzing `AST` and `PAST` was investigated in [26, 37].

The most common non-termination technique for TRSs is the detection of loops. Thus, to disprove `AST` or `PAST` of probabilistic term rewriting, in this work we lift the idea of disproving termination via loops to the probabilistic setting, and consider embeddings of random walks based on loops.

In [4], random walks are also used to analyze `AST` (of higher-order probabilistic functional programs). However, here one over-approximates all runs by random walks in order to prove `AST` by counting recursive calls. Instead, our goal is to under-approximate a possible run in order to disprove `AST`. This requires additional conditions to ensure that the counted calls are indeed executable.

Structure. We present preliminaries on probability theory, random walks, and (probabilistic) term rewriting in Sect. 2. Then, we introduce our novel approach to disprove termination of PTRSs via an embedding of random walks in Sect. 3. More precisely, in Sect. 4 we show how to embed random walks based on loops and develop a dynamic programming algorithm to solve the arising counting problems for occurrences of terms. In Sect. 5, we show how to embed random walks based on looping pattern terms. We evaluate our implementation and conclude in Sect. 6. All proofs can be found in [31].

2 Preliminaries

In Sect. 2.1, we start with basic concepts of probability theory and define the

notion of *random walk programs (without direct termination)* as in [19], which we will use as lower bounds to disprove (positive) almost-sure termination of probabilistic term rewrite systems in Sect. 4 and 5. For an introduction to general random walks, see, e.g., [21, 33, 42]. We recapitulate ordinary term rewriting in Sect. 2.2, and probabilistic term rewriting in Sect. 2.3.

2.1 Probability Theory and Random Walks

In probability theory, one considers *outcomes* $\alpha \in \Omega$ of a random process and sets of *events* $\mathfrak{A} \subseteq 2^\Omega$ (where an event is a subset of outcomes), and one assigns probabilities to each event. Such an *event space* $\mathfrak{A}$ has to contain Ω and it must be closed under complement and countable unions. The pair $(\Omega, \mathfrak{A})$ is called a *measurable space*. A *probability space* $(\Omega, \mathfrak{A}, \mathbb{P})$ extends a measurable space $(\Omega, \mathfrak{A})$ by a *probability measure* $\mathbb{P}$ which maps every event from $\mathfrak{A}$ to a probability between 0 and 1 such that $\mathbb{P}(\Omega) = 1$, $\mathbb{P}(\varnothing) = 0$, and $\mathbb{P}(\biguplus_{i\geq 0} A_i) = \sum_{i\geq 0} \mathbb{P}(A_i)$ for pairwise disjoint events $A_i \in \mathfrak{A}$. As in [19], we use the measurable space $(\mathbb{Z}^\omega, \mathfrak{A}_{\text{cyl}})$ on infinite words defined by the typical cylinder construction used in MDP theory. We call the possible outcomes $\alpha \in \mathbb{Z}^\omega$ *runs* and a finite word $\alpha \in \mathbb{Z}^*$ a *prefix run*. Let $\alpha(i)$ be the number at position i in α.

A *random walk (program)* μ is a function $\mu : \mathbb{Z} \to \mathbb{R}_{\geq 0}$ whose *support* $\text{Supp}(\mu) = \{x \in \mathbb{Z} \mid \mu(x) > 0\}$ is finite and satisfies $\sum_{x\in\text{Supp}(\mu)} \mu(x) = 1$. The function μ represents the transition relation of a random walk on $\mathbb{Z}$. If the current value of the random walk is $y > 0$, then for any $x \in \mathbb{Z}$, $\mu(x)$ is the probability that the random walk transitions from y to $y + x$. We stop when we reach a number ≤ 0 and do not perform any transition steps anymore.[1] For a prefix run $\alpha \in \mathbb{Z}^*$, let $\langle\alpha\rangle \subseteq \mathbb{Z}^\omega$ denote the set of all infinite words with prefix α (called a *cylinder set*). The probability measure $\mathbb{P}^\mu_{x_0}$ (given some start value $x_0 \in \mathbb{Z}$) of a random walk is defined on the event space $\mathfrak{A}_{\text{cyl}} = \{\langle\alpha\rangle \mid \alpha \in \mathbb{Z}^*\}$, i.e., on the cylinder sets $\langle\alpha\rangle$ of all prefix runs α. Given a finite word $\alpha = \alpha(0)\dots\alpha(k)$, we define $\mathbb{P}^\mu_{x_0}(\langle\alpha\rangle) = 0$ if $\alpha(0) \neq x_0$, and $\mathbb{P}^\mu_{x_0}(\langle\alpha\rangle) = \prod_{i=0}^{k-1} \mu(\alpha(i+1) - \alpha(i))$ otherwise.

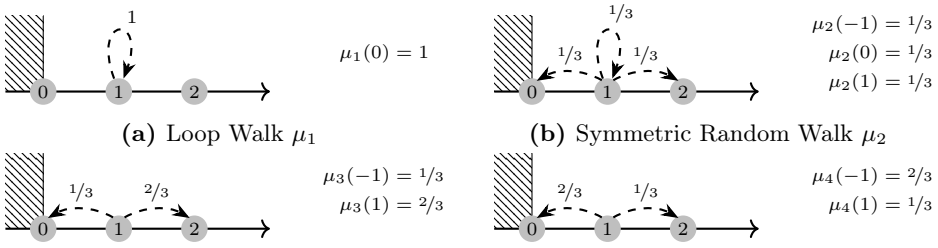
Depending on μ and its *expected value* $\mathbb{E}(\mu) = \sum_{x\in\text{Supp}(\mu)} x \cdot \mu(x)$, we characterize four different types of random walks:

1. If $\mu(0) = 1$, then μ is a *loop walk*.
2. If $\mu(0) < 1$ and $\mathbb{E}(\mu) = 0$, then μ is a *symmetric random walk*.
3. If $\mu(0) < 1$ and $\mathbb{E}(\mu) > 0$, then μ is a *positively biased random walk*.
4. If $\mu(0) < 1$ and $\mathbb{E}(\mu) < 0$, then μ is a *negatively biased random walk*.

Fig. 1 shows examples for all four different types, where we start at $x_0 = 1$.

We are interested in the probability of termination and the expected number of steps it takes to terminate. The random walk μ *certainly terminates* if there is no infinite word α with $\alpha(i) > 0$ and $\mu(\alpha(i+1) - \alpha(i)) > 0$ for all $i \in \mathbb{N}$. This only holds if $\mu(x) = 0$ for all $x \geq 0$. Since this is a rather restrictive requirement, we are more interested in the *probability of termination*. A run α *terminates* if there exists an $n \in \mathbb{N}$ such that $\alpha(n) \leq 0$, and the termination length $|\alpha|$ is defined as the smallest such n. Let $\langle\mathtt{SN}\rangle = \biguplus_{n\in\mathbb{N}} \biguplus_{\alpha\in\mathbb{Z}^\omega,|\alpha|=n} \langle\alpha\rangle$ be the event of

[1] Ordinary random walks [42] do not stop and are identically distributed everywhere.



(a) Loop Walk μ_1 **(b)** Symmetric Random Walk μ_2

(c) Positively Biased Random Walk μ_3 **(d)** Negatively Biased Random Walk μ_4

Fig. 1: Four Different Random Walks μ_1, μ_2, μ_3, and μ_4

all terminating runs. (As usual, "SN" stands for "strong normalization", which is equivalent to "termination".) Then the *probability of termination* of a random walk μ starting in x_0 is $\mathbb{P}^{\mu}_{x_0}(\langle \mathtt{SN} \rangle) = \sum_{n \in \mathbb{N}} \sum_{\alpha \in \mathbb{Z}^\omega, |\alpha|=n} \mathbb{P}^{\mu}_{x_0}(\langle \alpha \rangle)$.

A random walk μ is *almost-surely terminating* (AST) if $\mathbb{P}^{\mu}_{x_0}(\langle \mathtt{SN} \rangle) = 1$ for all $x_0 \in \mathbb{Z}$. In addition, one is also interested in the expected number of steps it takes to terminate. We define the *expected derivation length* $\mathrm{edl}(\mu, x_0) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ of a random walk μ starting in x_0 as $\mathrm{edl}(\mu, x_0) = \infty$ if μ is not AST, and otherwise, as $\mathrm{edl}(\mu, x_0) = \sum_{n \in \mathbb{N}} \sum_{\alpha \in \mathbb{Z}^\omega, |\alpha|=n} n \cdot \mathbb{P}^{\mu}_{x_0}(\langle \alpha \rangle)$. A random walk μ is *positively almost-surely terminating* (PAST) if we have $\mathrm{edl}(\mu, x_0) < \infty$ for all $x_0 \in \mathbb{Z}$.

With this characterization, as in [19, Thm. 18], classical results on random walks [42] yield the following classification w.r.t. AST and PAST.

Theorem 1 (AST and PAST of Random Walks). *Let μ be a random walk.*

1. *If μ is a loop walk, then μ is neither* AST *nor* PAST.
2. *If μ is a symmetric random walk, then μ is* AST *but not* PAST.
3. *If μ is a positively biased random walk, then μ is neither* AST *nor* PAST.
4. *If μ is a negatively biased random walk, then μ is* AST *and* PAST.

Example 2. Based on Thm. 1, μ_1 and μ_3 are neither AST nor PAST. The random walk μ_2 is AST but not PAST, and μ_4 is both AST and PAST.

2.2 Rewriting

Next, we recapitulate abstract rewriting and term rewriting [2]. An *abstract reduction system* (ARS) on a set A is a binary relation $\to \subseteq A \times A$. Instead of $(a, b) \in \to$ one also writes $a \to b$. For any $n \in \mathbb{N}$, we define $\to^n$ as $\to^0 = \{(a, a) \mid a \in A\}$ and $\to^{n+1} = \to^n \circ \to$, where "$\circ$" denotes composition of relations. Moreover, $\to^* = \bigcup_{n \in \mathbb{N}} \to^n$ and $\to^+ = \bigcup_{n \in \mathbb{N}_{>0}} \to^n$. $\mathtt{NF}_{\to}$ denotes the set of all objects in *normal form* w.r.t. $\to$, i.e., $a \in \mathtt{NF}_{\to}$ if there is no $b \in A$ with $a \to b$.

The set $\mathcal{T}(\Sigma, \mathcal{V})$ of all *terms* over a finite set of *function symbols* $\Sigma = \biguplus_{k \in \mathbb{N}} \Sigma_k$ and an infinite set of *variables* $\mathcal{V}$ is the smallest set with $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$, and if $f \in \Sigma_k$ and $t_1, \ldots, t_k \in \mathcal{T}(\Sigma, \mathcal{V})$, then $f(t_1, \ldots, t_k) \in \mathcal{T}(\Sigma, \mathcal{V})$. If Σ and $\mathcal{V}$ are clear, we just write $\mathcal{T}$ instead of $\mathcal{T}(\Sigma, \mathcal{V})$. For example, $\mathsf{gt}(\mathsf{s}(x), \mathsf{0})$ is a term over a signature Σ where $\mathsf{gt} \in \Sigma_2$, $\mathsf{s} \in \Sigma_1$, and $\mathsf{0} \in \Sigma_0$. A *substitution* is a function $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$ where $\mathrm{dom}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is finite, and we

often write $x\sigma$ instead of $\sigma(x)$. If $\mathrm{dom}(\sigma) = \{x_1, \dots, x_n\}$ and $\sigma(x_i) = s_i$ for all $1 \leq i \leq n$, we also write $\sigma = [x_1/s_1, \dots, x_n/s_n]$. Substitutions homomorphically extend to terms: if $t = f(t_1, \dots, t_k) \in \mathcal{T}$ then $t\sigma = f(t_1\sigma, \dots, t_k\sigma)$. Thus, for a substitution σ with $x\sigma = \mathsf{s}(\mathsf{0})$ we obtain $\mathsf{s}(x)\sigma = \mathsf{s}(\mathsf{s}(\mathsf{0}))$. For any term $t \in \mathcal{T}$, the set of *positions* $\mathrm{Pos}(t)$ is the smallest subset of $\mathbb{N}^*$ with $\varepsilon \in \mathrm{Pos}(t)$, and if $t = f(t_1, \dots, t_k)$ then for all $1 \leq j \leq k$ and all $\pi \in \mathrm{Pos}(t_j)$ we have $j.\pi \in \mathrm{Pos}(t)$. $\mathrm{Pos}_X(t) \subseteq \mathrm{Pos}(t)$ denotes all positions of symbols or variables from $X \subseteq \Sigma \cup \mathcal{V}$. A position π_1 is *above* π_2 if π_1 is a (not necessarily proper) prefix of π_2. If π_1 is not above π_2 and π_2 is not above π_1, then they are *orthogonal* (denoted $\pi_1 \perp \pi_2$). If $\pi \in \mathrm{Pos}(t)$ then $t|_\pi$ denotes the subterm at position π and $t[r]_\pi$ denotes the term that results from replacing the subterm $t|_\pi$ at position π with the term $r \in \mathcal{T}$. We write $t \trianglelefteq s$ if t is a subterm of s and $t \triangleleft s$ if t is a *proper* subterm of s (i.e., if $t \trianglelefteq s$ and $t \neq s$). For example, we have $\mathrm{Pos}(\mathsf{gt}(\mathsf{s}(x), \mathsf{0})) = \{\varepsilon, 1, 1.1, 2\}$, $\mathsf{gt}(\mathsf{s}(x), \mathsf{0})|_2 = \mathsf{0}$, $\mathsf{gt}(\mathsf{s}(x), \mathsf{0})[\mathsf{s}(y)]_2 = \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))$, and $\mathsf{s}(y) \triangleleft \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))$. A *context* C is a term from $\mathcal{T}(\Sigma \uplus \{\Box\}, \mathcal{V})$ which contains exactly one occurrence of the constant $\Box$ (called "hole"). If $C|_\pi = \Box$, then $C[s]$ is a shorthand for $C[s]_\pi$.

A *term rewrite rule* $\ell \to r$ is a pair of terms $(\ell, r) \in \mathcal{T} \times \mathcal{T}$ with $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ and $\ell \notin \mathcal{V}$, where $\mathcal{V}(t)$ is the set of all variables occurring in $t \in \mathcal{T}$. A *term rewrite system* (TRS) $\mathcal{R}$ is a finite set of term rewrite rules, and it induces an ARS $(\mathcal{T}, \to_\mathcal{R})$ where $s \to_\mathcal{R} t$ holds if there is a $\pi \in \mathrm{Pos}(s)$, a rule $\ell \to r \in \mathcal{R}$, and a substitution σ with $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$. We sometimes simply refer to $\mathcal{R}$ instead of $\to_\mathcal{R}$. For example, the following TRS $\mathcal{R}_{\mathsf{gt}}$ computes the "**g**reater **t**han" function on natural numbers (represented by $\mathsf{0}$ and the successor function s).

$$\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{gt}(x, y) \qquad \mathsf{gt}(\mathsf{s}(x), \mathsf{0}) \to \mathsf{true} \qquad \mathsf{gt}(\mathsf{0}, y) \to \mathsf{false}$$

Here, we have $\mathsf{gt}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0})) \to_{\mathcal{R}_{\mathsf{gt}}} \mathsf{gt}(\mathsf{0}, \mathsf{0}) \to_{\mathcal{R}_{\mathsf{gt}}} \mathsf{false}$, where $\mathsf{false} \in \mathtt{NF}_{\mathcal{R}_{\mathsf{gt}}}$.

2.3 Probabilistic Rewriting

A *probabilistic* ARS has finite multi-distributions[2] on the right-hand sides of its rewrite rules. A finite *multi-distribution* μ on a set $A \neq \varnothing$ is a finite multiset of pairs $(p : a)$, with a probability $0 < p \leq 1$ and $a \in A$, such that $\sum_{(p:a) \in \mu} p = 1$. $\mathrm{FDist}(A)$ denotes the set of all finite multi-distributions on A. For $\mu \in \mathrm{FDist}(A)$, its *support* is the multiset $\mathrm{Supp}(\mu) = \{a \mid (p : a) \in \mu \text{ for some } p\}$. A *probabilistic abstract reduction system* (PARS) is a pair $(A, \to)$ with $\to \subseteq A \times \mathrm{FDist}(A)$.

A *probabilistic term rewrite rule* $\ell \to \mu$ is a pair $(\ell, \mu) \in \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for all $r \in \mathrm{Supp}(\mu)$, and a *probabilistic TRS* (PTRS) is a finite set of probabilistic term rewrite rules. Similar to TRSs, a PTRS $\mathcal{P}$ induces a PARS $(\mathcal{T}, \to_\mathcal{P})$ with $s \to_\mathcal{P} \{p_1 : t_1, \dots, p_k : t_k\}$ if there is a position $\pi \in \mathrm{Pos}(s)$, a rule $\ell \to \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{P}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. Again, we sometimes refer to $\mathcal{P}$ instead of $\to_\mathcal{P}$. Consider the PTRS $\mathcal{P}_{\mathsf{geo}}$ with the only rule $\mathsf{geo}(x) \to \{1/2 : \mathsf{geo}(\mathsf{s}(x)),\ 1/2 : x\}$. When starting with the term $\mathsf{geo}(\mathsf{0})$, repeated rewriting yields $\mathsf{s}^k(\mathsf{0})$ (representing $k \in \mathbb{N}$) with a probability of $(1/2)^{k+1}$, i.e., a geometric distribution.

[2] We use multi-distributions instead of ordinary distributions. Here, the same object can occur multiple times and, due to non-determinism, be evaluated differently.

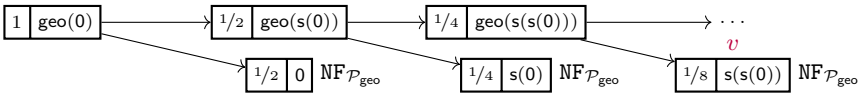


Fig. 2: A $\mathcal{P}_{\mathsf{geo}}$-RST $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = \mathsf{geo}(\mathsf{0})$ and $\mathrm{h}(\mathfrak{T}) = \omega$. The node v is labeled by the probability $p_v = 1/8$ and the term $a_v = \mathsf{s}(\mathsf{s}(\mathsf{0}))$, and it has depth $\mathrm{d}(v) = 3$.

To track rewrite sequences of a PARS $(A, \to)$ with their probabilities, we consider (possibly infinite) *rewrite sequence trees (RSTs)* [30]. The nodes v of a $\to$-RST are labeled by pairs $(p_v : a_v)$ of a probability $p_v \in (0, 1]$ and an object $a_v \in A$, where the probability at the root is 1. For each node v with successors $w_1, \ldots, w_k$, the edge relation represents a rewrite step, i.e., $a_v \to \{\frac{p_{w_1}}{p_v} : a_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : a_{w_k}\}$. For a $\to$-RST $\mathfrak{T}$, $V(\mathfrak{T})$ denotes its set of nodes, $\mathrm{root}(\mathfrak{T})$ is the object at its root, and $\mathrm{Leaf}(\mathfrak{T})$ denotes its set of leaves. The *depth* $\mathrm{d}(v)$ of a node $v \in V(\mathfrak{T})$ is the number of steps it takes to reach v from the root. The *height* of a tree $\mathfrak{T}$ is $\mathrm{h}(\mathfrak{T}) = \sup_{v \in V(\mathfrak{T})} \mathrm{d}(v) \in \mathbb{N} \cup \{\omega\}$. An example for a $\mathcal{P}_{\mathsf{geo}}$-RST is shown in Fig. 2.

The *termination probability* of a $\to$-RST $\mathfrak{T}$ is $|\mathfrak{T}| = \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v$. A PARS $(A, \to)$ is *almost-surely terminating* (AST) if $|\mathfrak{T}| = 1$ for all $\to$-RSTs $\mathfrak{T}$, i.e., the termination probability is 1. However, AST does not imply that the expected number of rewrite steps is finite. The *expected derivation length* of a $\to$-RST $\mathfrak{T}$ is $\mathrm{edl}(\mathfrak{T}) = \infty$ if $|\mathfrak{T}| < 1$, and $\mathrm{edl}(\mathfrak{T}) = \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} \mathrm{d}(v) \cdot p_v$, otherwise. For the $\mathcal{P}_{\mathsf{geo}}$-RST $\mathfrak{T}$ in Fig. 2 we get $\mathrm{edl}(\mathfrak{T}) = 1 \cdot 1/2 + 2 \cdot 1/4 + 3 \cdot 1/8 + \ldots = 2$, so in expectation we perform 2 rewrite steps. A PARS $(A, \to)$ is *positively almost-surely terminating* (PAST) if $\mathrm{edl}(\mathfrak{T})$ is finite for all $\to$-RSTs $\mathfrak{T}$. These notions are equivalent to the ones in [1, 7, 27] where (P)AST is defined via a lifting of $\to$ to multisets or via stochastic processes. Note that requiring *fully evaluated* RSTs (where all leaves are normal forms) does not change the class of PTRSs that are (P)AST.

Remark 3. When rewriting a term, both the position of the rewrite step and the applied rule are chosen *non-deterministically*. PTRSs extend this non-determinism by an additional probabilistic choice. Thus, (P)AST for PTRSs corresponds to (P)AST under all schedulers that resolve the non-probabilistic non-determinism.

3 Disproving AST and PAST via Loops and Embeddings

The most common approach to disprove termination of a TRS $\mathcal{R}$ is by finding *loops*. A loop is a sequence $t \to_{\mathcal{R}}^{+} C[t\sigma]$ for some term t, context C, and substitution σ. It gives rise to an infinite rewrite sequence $t \to_{\mathcal{R}}^{n} C[t\sigma] \to_{\mathcal{R}}^{n} C[C[t\sigma]\sigma] \to_{\mathcal{R}} \ldots$ for some $n \in \mathbb{N}_{>0}$. However, to disprove (P)AST of a PTRS $\mathcal{P}$, a loop in the *non-probabilistic variant* $\mathrm{np}(\mathcal{P}) = \{\ell \to r \mid \ell \to \mu \in \mathcal{P}, r \in \mathrm{Supp}(\mu)\}$ is not sufficient.

Example 4. Consider the PTRS $\mathcal{P}_1$ with the rule $\mathsf{g}(x) \to \{2/3 : x, 1/3 : \mathsf{g}(\mathsf{g}(x))\}$ modeling a negatively biased random walk on the number of g's. Here, $\mathrm{np}(\mathcal{P}_1)$ contains the two rules $\mathsf{g}(x) \to x$ and $\mathsf{g}(x) \to \mathsf{g}(\mathsf{g}(x))$. The latter gives rise to the loop $\mathsf{g}(x) \to C[\mathsf{g}(x)\sigma]$ with $C = \mathsf{g}(\Box)$ and the identity substitution σ. However, $\mathcal{P}_1$ corresponds to the random walk μ_4 from Fig. 1, and thus, it is PAST.

Instead of $\mathrm{np}(\mathcal{P})$, one has to consider $\mathcal{P}$-RSTs. If there is a $\mathcal{P}$-RST with root t and an instance of t in every leaf, then $\mathcal{P}$ is not AST and thus, also not PAST.

Theorem 5 (Disproving (P)AST via Loops). *Let $\mathcal{P}$ be a PTRS and $\mathfrak{T}$ be a $\mathcal{P}$-RST with $\mathrm{h}(\mathfrak{T}) > 0$ where $\mathrm{root}(\mathfrak{T}) = t$ and for every $v \in \mathrm{Leaf}(\mathfrak{T})$ there is a context C_v and a substitution σ_v with $t_v = C_v[t\sigma_v]$. Then $\mathcal{P}$ is neither* AST *nor* PAST.

However, Thm. 5 can only be used to disprove (P)AST for a very restricted class of PTRSs. To handle more complex examples, instead of requiring that a looping term t occurs in every leaf of a $\mathcal{P}$-RST, one has to find a PARS $(A, \to)$ and a $\to$-RST $\mathfrak{T}$ where it is known that $|\mathfrak{T}| < 1$ (to disprove AST) or $\mathrm{edl}(\mathfrak{T}) = \infty$ (to disprove PAST) holds. In addition, one has to find a $\mathcal{P}$-RST $\mathfrak{T}'$ and an *embedding* $\mathsf{e} : V(\mathfrak{T}) \to V(\mathfrak{T}')$ which ensures $|\mathfrak{T}'| \leq |\mathfrak{T}|$ and $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}')$. If $|\mathfrak{T}| < 1$, then this disproves AST of $\mathcal{P}$, and if $\mathrm{edl}(\mathfrak{T}) = \infty$, then it disproves PAST of $\mathcal{P}$.

Definition 6 (RST-Embedding). *Let $i \in \{1, 2\}$, let $(A_i, \to_i)$ be a PARS, and let $\mathfrak{T}_i$ be a $\to_i$-RST. An embedding from $\mathfrak{T}_1$ to $\mathfrak{T}_2$ is an injective mapping $\mathsf{e} : V(\mathfrak{T}_1) \to V(\mathfrak{T}_2)$ such that $p_v = p_{\mathsf{e}(v)}$ for all $v \in V(\mathfrak{T}_1)$ (the probability of v in $\mathfrak{T}_1$ and the probability of its image $\mathsf{e}(v)$ in $\mathfrak{T}_2$ are the same) and if there is a path from a node v to w in $\mathfrak{T}_1$, then there is a path from $\mathsf{e}(v)$ to $\mathsf{e}(w)$ in $\mathfrak{T}_2$ as well. Moreover, if w is a direct successor of v in $\mathfrak{T}_1$, then the path from $\mathsf{e}(v)$ to $\mathsf{e}(w)$ in $\mathfrak{T}_2$ does not contain any other nodes from the image $\mathsf{e}(V(\mathfrak{T}_1))$.*

Theorem 7 (Disproving (P)AST via Embeddings). *Let $(A_1, \to_1)$, $(A_2, \to_2)$ be two PARSs, and let $\mathfrak{T}_i$ be a $\to_i$-RST for $i \in \{1, 2\}$ such that there exists an embedding from $\mathfrak{T}_1$ to $\mathfrak{T}_2$. Then, $|\mathfrak{T}_2| \leq |\mathfrak{T}_1|$ and $\mathrm{edl}(\mathfrak{T}_1) \leq \mathrm{edl}(\mathfrak{T}_2)$.*

4 Embedding Random Walks Based on Term Occurrences

To embed symmetric or even positively biased random walks, we again search for a loop where t rewrites to $C[t\sigma]$, but now we also consider the *probabilities* and the *number of loops* represented by the right-hand side $C[t\sigma]$.

Example 8. The PTRS $\mathcal{P}_2$ with the rule $\mathsf{g}(x) \to \{2/3 : x, 1/3 : \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))\}$ models a symmetric random walk on the number of g's, because their number increases in each rule application by $2/3 \cdot (-1) + 1/3 \cdot 2 = 0$ in expectation. While $\mathcal{P}_1$ from Ex. 4 is PAST, $\mathcal{P}_2$ is not. The difference is that the symbol g occurs three instead of two times in the second choice of the distribution on the right-hand side.

Ex. 8 illustrates that it is important how often a looping term like $\mathsf{g}(x)$ occurs on the right-hand side. In this example, the three subterms of $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ that are instantiations of $\mathsf{g}(x)$ can indeed be counted separately, because they do not overlap at a non-variable position. Thus, we first solve the problem of finding the maximal number of such non-overlapping instantiations (called *occurrences*).

Definition 9 (Term Occurrences, $\trianglelefteq_\pi$). *Let $t, s \in \mathcal{T}$. We say that t* occurs *in s at position π (denoted $t \trianglelefteq_\pi s$) if $s|_\pi = t\sigma$ for some substitution σ. Two positions π_1 and π_2 are* overlapping *w.r.t. t if there is a $\tau \in \mathrm{Pos}_\Sigma(t)$ with $\pi_1 = \pi_2.\tau$ or $\pi_2 = \pi_1.\tau$. Let $\mathrm{NO}(t, s)$ be the set of all sets of* pairwise **n**on-overlapping **o**ccurrences *of t in s. So for $S \subseteq \mathrm{Pos}(s)$ we have $S \in \mathrm{NO}(t, s)$ iff $t \trianglelefteq_\pi s$ for all $\pi \in S$, and $\pi_1, \pi_2 \in S$ with $\pi_1 \neq \pi_2$ implies that π_1, π_2 are non-overlapping w.r.t. t. Let $\mathrm{maxNO}(t, s) = \max_{S \in \mathrm{NO}(t,s)} |S|$ be the maximal cardinality of sets in $\mathrm{NO}(t, s)$.*

Algorithm 1: Compute $\mathrm{maxNO}(t, s)$ for $t \notin \mathcal{V}$

```
1  q ← EmptyList()
2  for s′ ⊴ s do
3      α_{s′} ← 0, β_{s′} ← 0, γ_{s′} ← 0                          // Initialize values
4  q.enqueue(leaves)                                              // Start with the leaves
5  while not q.isEmpty() do
6      s′ ← q.dequeue()
7      γ_{s′} ← 1
8      if s′ is leaf then
9          α_{s′} ← 1,  if t ◀_ε s′                               // t matches s′
                    0,  otherwise                                 // t does not match s′
10     else
11         β_{s′} ← Σ_{s′=f(v_1,…,v_k)} Σ_{j=1}^{k} α_{v_j}
12         α_{s′} ← max{β_{s′}, Σ_{π∈Pos_V(t)} α_{s′|_π} + 1},  if t ◀_ε s′   // t matches s′
                    β_{s′},                                       otherwise    // t does not match s′
13     if ∀v ∈ s′.parent().children() : γ_v = 1                   // All siblings of s′ processed
14     then
15         q.enqueue(s′.parent())                                 // Enqueue parent
16 return α_s
```

So if $t \trianglelefteq_{\pi_1} s$ and $t \trianglelefteq_{\pi_2} s$ where π_1, π_2 are non-overlapping w.r.t. t, then π_1, π_2 are either orthogonal, or we have $s|_{\pi_1} = t\sigma_1$ and $t\sigma_2 \trianglelefteq \sigma_1(x)$ (or $s|_{\pi_2} = t\sigma_2$ and $t\sigma_1 \trianglelefteq \sigma_2(x)$) for some variable $x \in \mathcal{V}(t)$ and substitutions σ_1, σ_2.

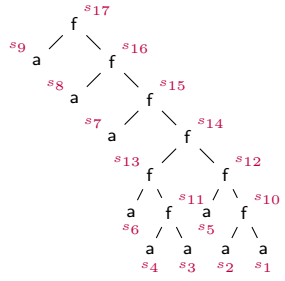
For example, we have $\mathsf{g}(x) \trianglelefteq_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ for all $\pi \in \{\varepsilon, 1, 1.1\}$ and $\mathrm{NO}(\mathsf{g}(x), \mathsf{g}(\mathsf{g}(\mathsf{g}(x))))$ consists of all subsets of $\{\varepsilon, 1, 1.1\}$. Thus, $\mathrm{maxNO}(\mathsf{g}(x), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = 3$. So there are three non-overlapping occurrences of the term $\mathsf{g}(x)$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$, i.e., these occurrences can indeed all be counted when analyzing non-termination.

On the other hand, $\mathsf{g}(\mathsf{g}(x)) \trianglelefteq_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ only holds for $\pi \in \{\varepsilon, 1\}$, and $\mathrm{NO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = \{\varnothing, \{\varepsilon\}, \{1\}\}$, but $\{\varepsilon, 1\} \notin \mathrm{NO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x))))$, because ε and 1 are overlapping w.r.t. $\mathsf{g}(\mathsf{g}(x))$, since $1 \in \mathrm{Pos}_\Sigma(\mathsf{g}(\mathsf{g}(x)))$. Thus, we just obtain $\mathrm{maxNO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = 1$, i.e., we can only count one instead of two occurrences of the term $\mathsf{g}(\mathsf{g}(x))$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$. Such overlapping occurrences cannot be counted separately, because rewriting one occurrence may interfere with the possibility to rewrite the other one, see Ex. 13.

Computing $\mathrm{maxNO}(t, s)$ can be done via a dynamic programming algorithm traversing the tree structure of s from the leaves to the root, see Alg. 1. For every node, i.e., every subterm $s' \trianglelefteq s$, we compute (1) $\alpha_{s'} = \mathrm{maxNO}(t, s')$ and (2) $\beta_{s'} = \max_{S \in \mathrm{NO}(t,s'), \varepsilon \notin S} |S|$, i.e., the maximal number of non-overlapping occurrences of t in s' if we do not consider an occurrence of t at the root of s'. For the leaves, we set $\beta_{s'} = 0$ and $\alpha_{s'} = 1$ if $t \trianglelefteq_\varepsilon s'$ and $\alpha_{s'} = 0$ otherwise. For each inner node where s' has the form $f(v_1, \ldots, v_k)$, we set $\beta_{s'} = \sum_{1 \le j \le k} \alpha_{v_j}$, and then check whether there is an occurrence of t at the root of s', i.e., whether $t \trianglelefteq_\varepsilon s'$. If not, then we set $\alpha_{s'} = \beta_{s'} = \sum_{1 \le j \le k} \alpha_{v_j}$. If there is an occurrence of t at the root of s', then $\alpha_{s'} = \max\{\beta_{s'}, \sum_{\pi \in \mathrm{Pos}_\mathcal{V}(t)} \alpha_{s'|_\pi} + 1\}$, i.e., $\alpha_{s'}$ is the maximum of $\beta_{s'}$ and the number obtained when considering the root position and maxNO for all subterms of s' corresponding to variable positions of t. In Alg. 1, we use a flag $\gamma_{s'}$ where $\gamma_{s'} = 1$ indicates that we have already computed $\alpha_{s'}$ and $\beta_{s'}$, and otherwise we have $\gamma_{s'} = 0$. Concerning the runtime, Alg. 1 has

Subterm s'	$\alpha_{s'}$	$\beta_{s'}$
s_1–s_{11}	0	0
s_{12}–s_{13}	1	0
s_{14}	2	2
s_{15}	2	2
s_{16}	3	2
s_{17}	3	3

(a) Dynamic Programming Table



(b) Tree Structure of s

(c) Tree Structure of t

Fig. 3: Computation of $\mathrm{maxNO}(t, s)$

to consider each subterm s' of s and check whether t matches s', which runs in time $\mathcal{O}(|s'|)$, where $|s'|$ is the number of positions in s'. So Alg. 1 runs in $\mathcal{O}(|s|^2)$.

Example 10. Fig. 3 shows the dynamic programming table for $t = \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, x))$ and $s = \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a})), \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))))))$ computed by Alg. 1. There is no occurrence of t in s_i for any $1 \leq i \leq 11$. For s_{12} and s_{13} there is one occurrence of t at the root. Therefore, $\alpha_{s_{12}} = \alpha_{s_{13}} = 1$. At s_{14} we have no occurrence of t at the root, but two occurrences below the root, thus $\alpha_{s_{14}} = \beta_{s_{14}} = 2$. The same holds for s_{15}. We have another occurrence at the root of s_{16} that does not overlap with the occurrences at the roots of s_{12} and s_{13}, so $\alpha_{s_{16}} = 3$. Finally, there is another occurrence at the root of s_{17}, but it is overlapping (w.r.t. t) with the preceding occurrence at the root of s_{16}. Thus, $\mathrm{maxNO}(t, s) = \alpha_s = \alpha_{s_{17}} = 3$.

We now embed random walks based on the maximal number of non-overlapping occurrences of a looping term t. We start with a $\mathcal{P}$-RST $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = t$ where t is *linear* (see Ex. 14 for the reason). As usual, $t \in \mathcal{T}$ is *linear* if $|t|_x \leq 1$ for all $x \in \mathcal{V}$, where $|t|_x$ is the number of positions $\pi \in \mathrm{Pos}(t)$ such that $t|_\pi = x$.

Then, we extend the RST $\mathfrak{T}$ by rewriting the terms in its leaves, until there are enough non-overlapping occurrences of t in the leaves to disprove (P)AST. In practice, one stops after reaching a suitable finite RST $\mathfrak{T}$, but Thm. 11 also holds for infinite RSTs $\mathfrak{T}$. We can then use the (finite) tree $\mathfrak{T}$ as a representation of an infinite $\mathcal{P}$-RST $\mathfrak{T}^\infty$, where we can embed a random walk that disproves (P)AST.

Theorem 11 (Embedding Random Walks via Occurrences (1)). *Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be a $\mathcal{P}$-RST with $\mathrm{h}(\mathfrak{T}) > 0$ and $\mathrm{root}(\mathfrak{T}) = t$, where t is linear. If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) > 1$, then $\mathcal{P}$ is not* AST. *Moreover, if $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) \geq 1$, then $\mathcal{P}$ is not* PAST.

So for every leaf v, we multiply its probability p_v with the number of non-overlapping occurrences of t in the term t_v of the leaf. This number of occurrences yields a random walk with $\mu(x) = \sum_{v \in \mathrm{Leaf}(\mathfrak{T}), x = \mathrm{maxNO}(t, t_v) - 1} p_v$ for all $x \in \mathbb{Z}$. By repeatedly rewriting an innermost[3] occurrence of t in the leaves of the tree according to the rules used in $\mathfrak{T}$, we obtain an infinite $\mathcal{P}$-RST $\mathfrak{T}^\infty$, where we can embed

[3] The innermost strategy must be used due to rules where a variable occurs more often on the left- than on the right-hand side. Such rules might decrease the number of occurrences of t in the arguments when rewriting at a non-innermost position.

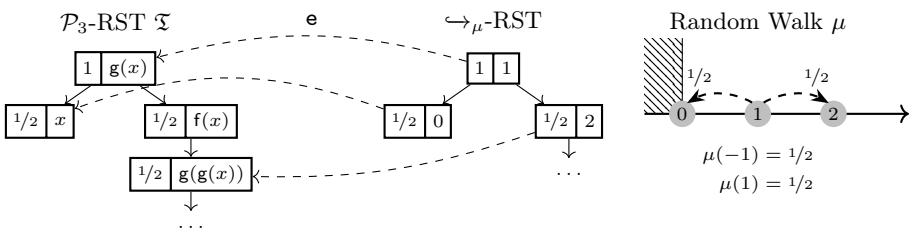


Fig. 4: Embedding of a Random Walk into a $\mathcal{P}_3$-RST to Disprove PAST of $\mathcal{P}_3$

the computation of the random walk μ. This yields the desired lower bound. Here, it is not a problem if different instantiations of t occur at different positions in a leaf. The reason is that if t starts a loop, then so does every instantiation of t.

Example 12 (Embedding via Occurrences and Innermost Rewriting). Consider the PTRS $\mathcal{P}_3$ with the rules $\mathsf{g}(x) \to \{1/2 : x, 1/2 : \mathsf{f}(x)\}$ and $\mathsf{f}(x) \to \{1 : \mathsf{g}(\mathsf{g}(x))\}$. If we count the occurrences of $\mathsf{g}(x)$, then the $\mathcal{P}_3$-RST $\mathfrak{T}$ gives rise to the symmetric random walk μ both depicted in Fig. 4, which represents a "lower bound" for the termination behavior of $\mathfrak{T}$. Here, we represent μ as a PARS $(\mathbb{Z}, \hookrightarrow_\mu)$ with $y \hookrightarrow_\mu \{(p : y + \mathrm{maxNO}(t, t_v) - 1) \mid v \in \mathrm{Leaf}(\mathfrak{T})\}$ for every $y > 0$. The $\hookrightarrow_\mu$-RST with infinite expected derivation length can be embedded in the tree that results from extending $\mathfrak{T}$ by rewriting the innermost subterm $\mathsf{g}(x)$ repeatedly according to the rules used in $\mathfrak{T}$. So since μ is not PAST, $\mathcal{P}_3$ is not PAST either. However, since μ is AST, this does not yield any information about whether $\mathcal{P}_3$ is AST.

Example 13 (Non-Overlappingness is Required). Non-overlappingness of the different occurrences of t in a leaf guarantees that rewriting an innermost occurrence of t does not interfere with the possibility to rewrite the other occurrences of t later on. To see this, consider the PTRS $\mathcal{P}_4$ with the rule $\mathsf{g}(\mathsf{g}(x)) \to \{1/3 : x, 2/3 : \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))\}$, which is AST. If one counted both occurrences of $\mathsf{g}(\mathsf{g}(x))$ in the term $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ in spite of their overlap, then one could embed the random walk μ_3 from Fig. 1, and thus, falsely disprove AST of $\mathcal{P}_4$. Here, the problem is that rewriting the innermost subterm $\mathsf{g}(\mathsf{g}(x))$ of $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ could yield $\mathsf{g}(x)$, i.e., then the outermost occurrence of $\mathsf{g}(\mathsf{g}(x))$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ would be "destroyed".

Example 14 (Linearity of t is Required). Linearity of t is required in Thm. 11, because otherwise rewriting an innermost occurrence of t in a leaf may "destroy" other occurrences of t in that leaf. For example, consider the PTRS $\mathcal{P}_5$ with the rule $\mathsf{f}(x, x) \to \{1/3 : \mathsf{a}, 1/3 : \mathsf{b}, 1/3 : \mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))\}$. If we count the occurrences of $\mathsf{f}(x, x)$, then the $\mathcal{P}_5$-RST $\mathfrak{T}$ where we perform a single rewrite step starting in $\mathsf{f}(x, x)$ gives rise to the random walk μ_5 with $\mu_5(-1) = 2/3$ and $\mu_5(2) = 1/3$, since $\mathrm{maxNO}(\mathsf{f}(x, x), \mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))) = 3$. Since μ_5 is not PAST, we would falsely disprove PAST of $\mathcal{P}_5$. But in fact, $\mathcal{P}_5$ is PAST. The problem is that rewriting the proper subterms of $\mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))$ may yield terms like $\mathsf{f}(\mathsf{a}, \mathsf{b})$, where the two arguments of f are not equal. Thus, rewriting an innermost occurrence of $t = \mathsf{f}(x, x)$ in $\mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))$ may "destroy" the occurrence of t at the root.

So when rewriting innermost occurrences of t according to the rules used in $\mathfrak{T}$, we need linearity of t. Instead, one could also rewrite outermost occurrences.

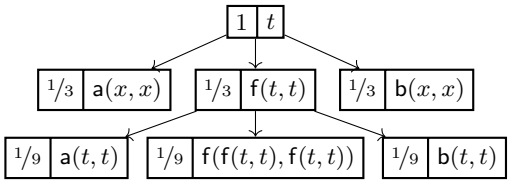


Fig. 5: $\mathcal{P}_5'$-RST

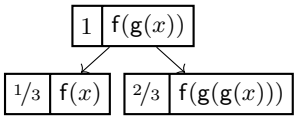


Fig. 6: $\mathcal{P}_6$-RST

Then, instead of linearity, we have to require that there is no $v \in \mathrm{Leaf}(\mathfrak{T})$ where a variable occurs more often in the looping term t than in the term t_v of the leaf v. A $\mathcal{P}$-RST $\mathfrak{T}$ is ***n**on-**v**ariable-**d**ecreasing* (nvd) if $|\,\mathrm{root}(\mathfrak{T})|_x \leq |t_v|_x$ for all $v \in \mathrm{Leaf}(\mathfrak{T})$ and all $x \in \mathcal{V}$. Rewriting an outermost occurrence of t according to an nvd $\mathcal{P}$-RST $\mathfrak{T}$ does not affect any other non-overlapping occurrences of t.

Theorem 15 (Embedding Random Walks via Occurrences (2)). *Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be an* nvd *$\mathcal{P}$-RST with $\mathrm{h}(\mathfrak{T}) > 0$ and $\mathrm{root}(\mathfrak{T}) = t$. If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) \gtrsim 1$, then $\mathcal{P}$ is not* (P)AST.

Example 16 (Embedding and Outermost Rewriting). The PTRS $\mathcal{P}_5'$ (similar to $\mathcal{P}_5$ from Ex. 14) has the rule $\mathsf{f}(x,x) \to \{1/3 : \mathsf{a}(x,x), 1/3 : \mathsf{b}(x,x), 1/3 : \mathsf{f}(\mathsf{f}(x,x), \mathsf{f}(x,x))\}$. Rewriting at the outermost position yields the nvd $\mathcal{P}_5'$-RST $\mathfrak{T}$ in Fig. 5, where $t = \mathsf{f}(x,x)$. As $\mathrm{maxNO}(t, \mathsf{a}(t,t)) = \mathrm{maxNO}(t, \mathsf{b}(t,t)) = 2$, $\mathrm{maxNO}(t, \mathsf{f}(\mathsf{f}(t,t), \mathsf{f}(t,t))) = 7$, and $\mathrm{maxNO}(t, \mathsf{a}(x,x)) = \mathrm{maxNO}(t, \mathsf{b}(x,x)) = 0$, we get $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) = 11/9 > 1$. So by Thm. 15 this disproves AST of $\mathcal{P}_5'$.

If t is not linear and the RST $\mathfrak{T}$ is not nvd, then we can only count ***o**rthogonal **o**ccurrences*. The maximal number of orthogonal occurrences of t in s is $\mathrm{maxOO}(t,s) = \max\{|S| \mid S \in \mathrm{NO}(t,s), \forall \pi_1, \pi_2 \in S \text{ with } \pi_1 \neq \pi_2 : \pi_1 \perp \pi_2\}$.

To compute $\mathrm{maxOO}(t,s)$, we can adjust Alg. 1 at Line 12 in the case of $t \trianglelefteq_\varepsilon s'$. Instead of setting $\alpha_{s'}$ to the maximum of $\beta_{s'}$ and $\sum_{\pi \in \mathrm{Pos}_{\mathcal{V}}(t)} \alpha_{s'|_\pi} + 1$, we set $\alpha_{s'}$ to $\max\{\beta_{s'}, 1\}$, because now we do not count occurrences below another occurrence anymore. The runtime of the adjusted algorithm is still in $\mathcal{O}(|s|^2)$.

Theorem 17 (Embedding Random Walks via Occurrences (3)). *Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be a $\mathcal{P}$-RST with $\mathrm{h}(\mathfrak{T}) > 0$ and $\mathrm{root}(\mathfrak{T}) = t$. If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOO}(t, t_v) \gtrsim 1$, then $\mathcal{P}$ is not* (P)AST.

To automate Thm. 11, 15, and 17, we have to find a $\mathcal{P}$-RST satisfying one of the two constraints. To this end, we first search for a looping term t. Here, we reuse the non-probabilistic loop detection algorithms from [16] on the non-probabilistic variant $\mathrm{np}(\mathcal{P})$. After finding a loop of $\mathrm{np}(\mathcal{P})$ starting with a term t (which is already an undecidable problem in general), we first check whether t is linear. If t is linear, then we reconstruct the corresponding $\mathcal{P}$-RST $\mathfrak{T}$ for this path. On all remaining terms u in the leaves of $\mathfrak{T}$, we check whether we can possibly reach a term s from u such that $t \trianglelefteq_\pi s$ for some $\pi \in \mathrm{Pos}(s)$. This is undecidable in general, but we use the *symbol transition graph* from [43] as a sufficient criterion. On those terms u where we may potentially reach such a term s, we extend the RST by performing further *rewrite steps*, to obtain leaves that contain more

occurrences of t. This is performed repeatedly until we have constructed an RST that satisfies one of the conditions of Thm. 11 or until we reach a threshold for the number of rewrite steps. In the latter case, we try to find another looping term to generate an RST in order to embed a suitable random walk. If the looping term t is not linear, then we proceed in an analogous way to apply Thm. 15 or Thm. 17, depending on whether the tree is nvd or not. Here, we compute both $\mathrm{maxNO}(t, t_v)$ and $\mathrm{maxOO}(t, t_v)$ for every leaf v, since by rewriting the leaves, it may change whether the tree is nvd or not.

5 Embedding Random Walks Based on Pattern Terms

Instead of counting the occurrences of a single term, we can also count the number of *instantiations* applied to a certain *base term*. To simplify the presentation, we only consider orthogonal occurrences in this section. Similar to Sect. 4, corresponding approaches that consider pairwise non-overlapping occurrences and use innermost or outermost rewriting are possible as well. However, how to automate such improvements is open and an interesting direction for future work.

Example 18. Consider $\mathcal{P}_6 = \{\mathsf{f}(\mathsf{g}(x)) \to \{1/3 : \mathsf{f}(x), 2/3 : \mathsf{f}(\mathsf{g}(\mathsf{g}(x)))\}\}$ modeling a positively biased random walk on the number of g's *directly below an* f in a term and the corresponding $\mathcal{P}_6$-RST in Fig. 6. Here, it is not enough to count the occurrences of $\mathsf{f}(\mathsf{g}(x))$ to disprove $\mathtt{AST}$, but instead we have to count the occurrences of $\mathsf{g}(x)$ below an f symbol. Moreover, the f in $\mathsf{f}(x)$ is crucial. The PTRS $\mathcal{P}_6'$ with the rule $\mathsf{f}(\mathsf{g}(x)) \to \{1/3 : x, 2/3 : \mathsf{f}(\mathsf{g}(\mathsf{g}(x)))\}$ does not model a random walk anymore, since for any term $\mathsf{f}(\mathsf{g}^n(x))$ we can directly stop and rewrite to $\mathsf{g}^{n-1}(x)$ (removing the outer f). So $\mathcal{P}_6$ is not $\mathtt{AST}$, but $\mathcal{P}_6'$ is even $\mathtt{PAST}$.

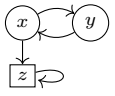
We formalize this with the concept of a *pattern term* $\langle t, \sigma\rangle$ that represents all terms which result from applying the *pumping substitution* σ repeatedly to the *base term* t, i.e., $\langle t, \sigma\rangle$ represents the terms t, $t\sigma$, $t\sigma^2$, etc. So if $t = \mathsf{f}(x)$ and $\sigma = [x/\mathsf{g}(x)]$, then $\langle t, \sigma\rangle$ represents the terms $\mathsf{f}(x)$, $\mathsf{f}(\mathsf{g}(x))$, $\mathsf{f}(\mathsf{g}(\mathsf{g}(x)))$, etc. A similar notion was defined in [12] to disprove *non-looping non-termination*. In contrast, here we want to *count loops*, and we want to count how often the substitution σ is applied. Therefore, we have to distinguish $t\sigma^m$ and $t\sigma^{m'}$ whenever $m \neq m'$.

Definition 19 (Pattern Term). *A pair* $\langle t, \sigma\rangle$ *is a* pattern term *if* $t\sigma^m \neq t\sigma^{m'}$ *holds whenever* $m \neq m'$. *We call* t *the* base term *and* σ *the* pumping substitution.

To check whether $\langle t, \sigma\rangle$ is a pattern term, we use the *variable transition graph*.

Definition 20 (Variable Transition Graph). *For any substitution* σ*, the graph* G_σ *has all variables as nodes and there is an edge from* x *to* y *if* $y \in \mathcal{V}(x\sigma)$. *For any term* t*, the* variable transition graph *of* σ *w.r.t.* t *is the subgraph* $G_{\sigma,t}$ *of* G_σ *which contains only those nodes that are reachable from a node in* $\mathcal{V}(t)$.

Example 21. Let $t = \mathsf{f}(x, y)$ and $\sigma = [x/\mathsf{f}(z, y), y/x]$. Then $G_{\sigma,t}$ is shown on the right. There is an edge from z to z because $z \notin \mathrm{dom}(\sigma)$, and hence, $z\sigma = z$. Nodes from $\mathcal{V}(t) = \{x, y\}$ are marked by circles.



$G_{\sigma,t}$ yields a computable, sound, and complete criterion for pattern terms.

Lemma 22 (Detecting Pattern Terms). *For a term t and a substitution σ, $\langle t,\sigma\rangle$ is a pattern term iff some cycle of $G_{\sigma,t}$ contains a variable x with $x\sigma \notin \mathcal{V}$.*

Example 23. The pair $\langle t,\sigma\rangle$ with $t = \mathsf{f}(x,y)$ and $\sigma = [x/\mathsf{f}(z,y), y/x]$ is a pattern term, because $G_{\sigma,t}$ has a cycle that contains the variable x and $x\sigma = \mathsf{f}(z,y) \notin \mathcal{V}$.

Next, we consider the problem of counting orthogonal pattern term occurrences.

Definition 24 (Pattern Occurrences). *Let $\langle t,\sigma\rangle$ be a pattern term and let $s \in \mathcal{T}$ be a term. We say that $\langle t,\sigma\rangle$* occurs with multiplicity $m_\pi \in \mathbb{N}$ *at position π in s (denoted by $\langle t,\sigma\rangle \trianglelefteq_\pi^{m_\pi} s$) if there is an occurrence $t \trianglelefteq_\pi s$ and m_π is the maximal number such that $t\sigma^{m_\pi} \trianglelefteq_\pi s$. For every set $S \in \mathrm{NO}(t,s)$, let $m_S = \sum_{\pi\in S} m_\pi$. Let $\mathrm{maxOPO}(t,\sigma,s) = \max\{m_S \mid S \in \mathrm{NO}(t,s), \forall \pi_1,\pi_2 \in S \text{ with } \pi_1 \neq \pi_2 : \pi_1 \perp \pi_2\}$ denote the maximal value that one can obtain by adding all multiplicities for a set S of pairwise orthogonal occurrences of $\langle t,\sigma\rangle$ in s.*

Example 25. As an example, consider the term $t = \mathsf{f}(x)$, the substitution $\sigma = [x/\mathsf{g}(x)]$, and $s = \mathsf{c}(\mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{g}(\mathsf{g}(x))))$. Then $\langle t,\sigma\rangle$ occurs in s at position 1 with multiplicity 1, and at position 2 with multiplicity 2. Thus, we have $m_1 = 1$ and $m_2 = 2$. Since $\mathrm{NO}(t,s) = \{\varnothing, \{1\}, \{2\}, \{1,2\}\}$ and both occurrences are at orthogonal positions, we have $\mathrm{maxOPO}(t,\sigma,s) = m_{\{1,2\}} = m_1 + m_2 = 1 + 2 = 3$.

To compute $\mathrm{maxOPO}(t,\sigma,s)$, we use Alg. 1 with $t\sigma$ instead of t. Moreover, we adjust Alg. 1 at Line 12 if $t\sigma \trianglelefteq_\varepsilon s'$. Instead of setting $\alpha_{s'}$ to $\max\{\beta_{s'}, 1\}$ as in the computation of $\mathrm{maxOO}(t\sigma, s)$, we set $\alpha_{s'}$ to $\max\{\beta_{s'}, m\}$ where m is the multiplicity of the occurrence of $\langle t,\sigma\rangle$ at the root of s'. Note that we only consider orthogonal occurrences here. So if we consider the occurrence at the root of s', then occurrences below the root are ignored. To find m, we check for occurrences of $t\sigma$, $t\sigma^2$, …, $t\sigma^{|s|}$. Compared to Alg. 1, we perform at most $|s|-1$ additional matching checks in each iteration, resulting in a runtime of $\mathcal{O}(|s|^3)$.

Example 26. Consider the pattern term $\langle t',\sigma\rangle$ with base term $t' = \mathsf{f}(\mathsf{a},x)$ and pumping substitution $\sigma = [x/\mathsf{f}(\mathsf{a},x)]$, and let $t = \mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},x))$ and s be as in Ex. 10. Thus, $t = \mathsf{f}(\mathsf{a},x)[x/\mathsf{f}(\mathsf{a},x)]$. We have occurrences of $\langle t',\sigma\rangle$ with multiplicity 0 at s_{10}, s_{11}, s_{15}; with multiplicity 1 at s_{12}, s_{13}, s_{16}; and with multiplicity 2 at the root s_{17}. Since we only consider occurrences at orthogonal positions, the maximum is obtained by adding the multiplicities for the orthogonal subterms s_{12} and s_{13} or by considering the multiplicity at the root. Thus, $\mathrm{maxOPO}(t',\sigma,s) = \alpha_s = \alpha_{s_{17}} = 2$.

As demonstrated by the PTRS $\mathcal{P}_6'$ in Ex. 18, if a term $t\sigma^m$ can be rewritten to a term without any occurrence of t, then this does not mean that the multiplicity is reduced by m, but it may mean that one directly reaches a normal form. Hence, then this pattern cannot be used to disprove (P)AST. So a pattern $\langle t,\sigma\rangle$ may only be used for disproving (P)AST if we have a $\mathcal{P}$-RST where every leaf contains an occurrence of t. Moreover, if we have a $\mathcal{P}$-RST $\mathfrak{T}$ that starts with $t\sigma^1$ and has an occurrence $\langle t,\sigma\rangle \trianglelefteq_\pi^q t_v$ in a leaf v, then we also need that the tree can be "generalized" from 1 to an arbitrary multiplicity $m \in \mathbb{N}_{>0}$, i.e., rewriting the term $t\sigma^m$ using the same rules as in $\mathfrak{T}$ at the same positions must result in a leaf v' with an occurrence $\langle t,\sigma\rangle \trianglelefteq_\pi^{q+m-1} t_{v'}$. For any occurrence $\langle t,\sigma\rangle \trianglelefteq_\pi^q t_v$, let $\kappa_{v,\pi}$ be the substitution such that $t\sigma^q\kappa_{v,\pi} = t_v|_\pi$. Then we ensure this generalization

property by requiring commutation of σ with all these substitutions $\kappa_{v,\pi}$.

Definition 27 (Pattern Tree). *Let $\mathcal{P}$ be a PTRS, let $\mathfrak{T}$ be a $\mathcal{P}$-RST, and let $\langle t, \sigma\rangle$ be a pattern term. $\mathfrak{T}$ is a* $\mathcal{P}$-pattern tree *for $\langle t, \sigma\rangle$ if $\mathrm{root}(\mathfrak{T}) = t\sigma$, for every leaf $v \in \mathrm{Leaf}(\mathfrak{T})$ there exists an occurrence $t \trianglelefteq_\pi t_v$, and whenever $t\sigma^q\kappa_{v,\pi} = t_v|_\pi$ for some $q \geq 0$, some $\pi \in \mathrm{Pos}(t_v)$, and some substitution $\kappa_{v,\pi}$, then the pumping substitution σ commutes with $\kappa_{v,\pi}$, i.e., $\sigma\kappa_{v,\pi} = \kappa_{v,\pi}\sigma$.*

For Ex. 18, the $\mathcal{P}_6$-RST in Fig. 6 is a pattern tree for the pattern term $\langle t, \sigma\rangle$ where $t = \mathsf{f}(x)$ and $\sigma = [x/\mathsf{g}(x)]$. Here, the substitutions $\kappa_{v_1,\varepsilon}$ and $\kappa_{v_2,\varepsilon}$ for the leaves are the identity, which commutes with σ. Indeed, this tree can be "generalized" to arbitrary multiplicities $m > 0$, because the corresponding $\mathcal{P}_6$-RST starting with $\mathsf{f}(\mathsf{g}^m(x))$ at the root has the leaves $\mathsf{f}(\mathsf{g}^{m-1}(x))$ and $\mathsf{f}(\mathsf{g}^{2+m-1}(x))$.

Similar to Thm. 17, by rewriting orthogonal occurrences, we result in a technique to embed random walks via patterns in a $\mathcal{P}$-RST.

Theorem 28 (Embedding RWs via Patterns). *Let $\mathcal{P}$ be a PTRS, $\langle t, \sigma\rangle$ be a pattern term, and let $\mathfrak{T}$ be a $\mathcal{P}$-pattern tree for $\langle t, \sigma\rangle$ with $\mathrm{h}(\mathfrak{T}) > 0$. If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOPO}(t, \sigma, t_v) \gtrsim_{(\dots)} 1$, then $\mathcal{P}$ is not* (P)AST.

Example 29. By Thm. 28, we embed the positively biased random walk μ_3 of Fig. 1 with $\mu_3(-1) = 1/3$ and $\mu_3(1) = 2/3$ in the $\mathcal{P}_6$-RST of Fig. 6 and disprove AST.

Example 30. By counting the number of applications of the pumping substitution $\sigma = [x/\mathsf{g}(y), y/\mathsf{f}(x)]$ to the base term $t = \mathsf{c}(y, x)$, we can disprove AST of the PTRS $\mathcal{P}_7$ with the only rule $\mathsf{c}(\mathsf{f}(x), \mathsf{g}(y)) \to \{1/3 : \mathsf{c}(y, x), 2/3 : \mathsf{c}(\mathsf{f}(\mathsf{g}(y)), \mathsf{g}(\mathsf{f}(x)))\}$ via Thm. 28. Again, we can embed the random walk μ_3 in the RST corresponding to the only rewrite rule, because the child $t = \mathsf{c}(y, x)$ has the probability $1/3$, and the second child $t\sigma^2 = \mathsf{c}(\mathsf{f}(\mathsf{g}(y)), \mathsf{g}(\mathsf{f}(x)))$ has the probability $2/3$.

Remark 31. Thm. 28 is *not* a generalization of Thm. 17. We can disprove PAST of $\mathcal{P}_8$ with the rule $\mathsf{g} \to \{1/2 : \mathsf{a}, 1/2 : \mathsf{c}(\mathsf{g}, \mathsf{g})\}$ via Thm. 17 by the tree $\mathfrak{T}$ corresponding to the only rule and counting the occurrences of g. However, there is no pattern term $\langle t, \sigma\rangle$ with $t\sigma = \mathsf{g}$. Indeed, PAST of $\mathcal{P}_8$ cannot be disproven via Thm. 28.

To automate Thm. 28, we adapt our implementation of Thm. 17. After finding the pattern $t\sigma$, we have to rewrite the leaves until we obtain a pattern tree. Note that we can directly stop if our reachability analysis shows that some leaf cannot reach a term containing an occurrence of t.

6 Evaluation and Conclusion

We presented the first techniques to disprove (P)AST of PTRSs automatically. To this end, we embed random walks in RSTs, based on counting occurrences of terms or multiplicities of patterns. In this way, qualitative approaches to detect non-termination of non-probabilistic TRSs based on loops or pattern terms can be lifted to a quantitative analysis in the probabilistic setting. Our approach can be based on any algorithm to detect loops for standard non-probabilistic TRSs.

We implemented our new contributions in our termination prover AProVE [17]. Currently, we run our techniques to prove and to disprove termination of PTRSs

in parallel and stop once one of the techniques succeeds. For proving termination, we use the probabilistic *dependency pair* (DP) framework [29, 30], which allows to apply different techniques to different sub-problems. In the future, we plan to integrate our new techniques to disprove (P)AST into the DP frameworks for AST [30] and PAST [29], respectively. Then the DP framework can help to restrict the search for non-termination proofs to those parts of a PTRS which are potentially non-terminating. Moreover, we also plan to analyze whether there are interesting subclasses of PTRSs where AST or PAST is decidable.

To evaluate the power of our new techniques, we used all 138 PTRSs from the *Termination Problem Data Base* [46], i.e., the benchmarks from the annual *Termination Competition* [18]. They contain 138 typical probabilistic programs, including examples with complicated probabilistic structure and probabilistic algorithms on lists and trees. This set was mainly developed to evaluate techniques that can prove (P)AST. Therefore, most of the examples are indeed AST, and we added 20 more examples that are not AST or not PAST, including the PTRSs from this paper and examples which express typical bugs in implementations.

Category	Thm. 5	Thm. 11	Thm. 15	Thm. 17	Thm. 28	AProVE
not-AST	8 (2)	17 (3)	18 (4)	16 (3)	12 (5)	24 (8)
not-PAST	8 (2)	33 (7)	34 (8)	28 (6)	30 (8)	49 (14)

We performed our experiments on a computer with an Apple M4 CPU and 16 GB of RAM, and a timeout of 30 seconds was used for each example. The table above shows the individual results for each of our novel theorems, and "AProVE" denotes the combination[4] of all techniques as implemented in our tool. The numbers in brackets denote the results when just considering the 20 new examples. Note that AProVE can *prove* AST for 70 of the 158 examples and *prove* PAST for 31 of the 158 examples. So at most $158 - 70 = 88$ examples may be non-AST and AProVE can disprove AST for 24 of them. Similarly, at most 127 examples may be non-PAST and AProVE disproves PAST for 49 of them.

The experimental results for Thm. 5 confirm that one indeed needs more elaborate techniques than just a direct lifting of the loop detection technique to the probabilistic setting. Our experiments show that each of our five theorems has its own benefits. More precisely, for each theorem, there exist examples that can only be solved by this theorem but not by any of the other four theorems.

Since ours is the first approach to disprove (P)AST of PTRSs automatically, we could not compare with other tools for termination analysis of PTRSs. While there exist techniques [10, 44] and the tool Amber [38] for disproving (P)AST of imperative programs, an experimental comparison would be problematic due to the fundamental differences between the considered languages.

For further details on our experiments and for instructions on how to run AProVE via its *web interface* or locally, we refer to: https://aprove-developers.github.io/DisprovingPTRSTermination/

[4] So "AProVE" combines our five new theorems for PTRSs. Without these theorems, AProVE could only disprove (P)AST for examples that have no probabilities except 1.

Acknowledgments. We thank Florian Frohn and Carsten Fuhs for a joint discussion on initial ideas for this paper during CADE '23 in Rome.

Disclosure of Interests. The authors have no competing interests to declare.

References

[1] M. Avanzini, U. Dal Lago, and A. Yamada. "On Probabilistic Term Rewriting". In: *Sci. Comput. Program.* 185 (2020). DOI: 10.1016/j.scico.2019.102338.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. DOI: 10.1017/CBO9781139172752.

[3] G. Barthe, J.-P. Katoen, and A. Silva (eds.) *Foundations of Probabilistic Programming*. Cambridge Univ. Press, 2020. DOI: 10.1017/9781108770750.

[4] R. Beutner and L. Ong. "On Probabilistic Termination of Functional Programs with Continuous Distributions". In: *Proc. PLDI '21*. 2021, pp. 1312–1326. DOI: 10.1145/3453483.3454111.

[5] D. Beyer and J. Strejček. "Improvements in Software Verification and Witness Validation: *SV-COMP* 2025". In: *Proc. TACAS '25*. LNCS 15698. Website of *SV-COMP*: https://sv-comp.sosy-lab.org/. 2025, pp. 151–186. DOI: 10.1007/978-3-031-90660-2_9.

[6] O. Bournez and C. Kirchner. "Probabilistic Rewrite Strategies. Applications to ELAN". In: *Proc. RTA '02*. LNCS 2378. 2002, pp. 252–266. DOI: 10.1007/3-540-45610-4_18.

[7] O. Bournez and F. Garnier. "Proving Positive Almost-Sure Termination". In: *Proc. RTA '05*. LNCS 3467. 2005, pp. 323–337. DOI: 10.1007/978-3-540-32033-3_24.

[8] M. Braverman. "Termination of Integer Linear Programs". In: *Proc. CAV '06*. LNCS 4144. 2006, pp. 372–385. DOI: 10.1007/11817963_34.

[9] M. Brockschmidt, T. Ströder, C. Otto, and J. Giesl. "Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode". In: *Proc. FoVeOOS '12*. LNCS 7421. 2012, pp. 123–141. DOI: 10.1007/978-3-642-31762-0_9.

[10] K. Chatterjee, P. Novotný, and Đ. Žikelić. "Stochastic Invariants for Probabilistic Termination". In: *Proc. POPL '17*. 2017, pp. 145–160. DOI: 10.1145/3009837.3009873.

[11] H.-Y. Chen, B. Cook, C. Fuhs, K. Nimkar, and P. O'Hearn. "Proving Nontermination via Safety". In: *Proc. TACAS '14*. LNCS 8413. 2014, pp. 156–171. DOI: 10.1007/978-3-642-54862-8_11.

[12] F. Emmes, T. Enger, and J. Giesl. "Proving Non-Looping Non-Termination Automatically". In: *Proc. IJCAR '12*. LNCS 7364. 2012, pp. 225–240. DOI: 10.1007/978-3-642-31365-3_19.

[13] J. Endrullis and H. Zantema. "Proving Non-Termination by Finite Automata". In: *Proc. RTA '15*. LIPIcs 36. 2015, pp. 160–176. DOI: 10.4230/LIPICS.RTA.2015.160.

[14] F. Frohn and C. Fuhs. "A Calculus for Modular Loop Acceleration and Non-Termination Proofs". In: *Int. J. Softw. Tools Technol. Transf.* 24 (2022), pp. 691–715. DOI: 10.1007/s10009-022-00670-2.

[15] F. Frohn and J. Giesl. "Proving Non-Termination by Acceleration Driven Clause Learning (Short Paper)". In: *Proc. CADE '23*. LNCS 14132. 2023, pp. 220–233. DOI: 10.1007/978-3-031-38499-8_13.

[16] J. Giesl, R. Thiemann, and P. Schneider-Kamp. "Proving and Disproving Termination of Higher-Order Functions". In: *Proc. FroCoS '05*. LNCS 3717. 2005, pp. 216–231. DOI: 10.1007/11559306_12.

[17] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. "Analyzing Program Termination and Complexity Automatically with AProVE". In: *J. Autom. Reason.* 58.1 (2017), pp. 3–31. DOI: 10.1007/s10817-016-9388-y.

[18] J. Giesl, A. Rubio, C. Sternagel, J. Waldmann, and A. Yamada. "The Termination and Complexity Competition". In: *Proc. TACAS '19*. LNCS 11429. Website of *TermComp*: https://termination-portal.org/wiki/Termination_Competition. 2019, pp. 156–166. DOI: 10.1007/978-3-030-17502-3_10.

[19] J. Giesl, P. Giesl, and M. Hark. "Computing Expected Runtimes for Constant Probability Programs". In: *Proc. CADE '19*. LNCS 11716. 2019, pp. 269–286. DOI: 10.1007/978-3-030-29436-6_16.

[20] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. "Probabilistic Programming". In: *Proc. FOSE '14*. 2014, pp. 167–181. DOI: 10.1145/2593882.2593900.

[21] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2020.

[22] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R.-G. Xu. "Proving Non-Termination". In: *Proc. POPL '08*. 2008, pp. 147–158. DOI: 10.1145/1328438.1328459.

[23] R. Gutiérrez and S. Lucas. "Automatically Proving and Disproving Feasibility Conditions". In: *Proc. IJCAR '20*. LNCS 12167. 2020, pp. 416–435. DOI: 10.1007/978-3-030-51054-1_27.

[24] M. Hark, F. Frohn, and J. Giesl. "Termination of Triangular Polynomial Loops". In: *Formal Methods Syst. Des.* 65.1 (2025), pp. 70–132. DOI: 10.1007/S10703-023-00440-Z.

[25] M. Hosseini, J. Ouaknine, and J. Worrell. "Termination of Linear Loops over the Integers". In: *Proc. ICALP '19*. LIPIcs 132. 2019. DOI: 10.4230/LIPIcs.ICALP.2019.118.

[26] B. L. Kaminski, J.-P. Katoen, and C. Matheja. "On the Hardness of Analyzing Probabilistic Programs". In: *Acta Informatica* 56.3 (2019), pp. 255–285. DOI: 10.1007/s00236-018-0321-1.

[27] J.-C. Kassing and J. Giesl. "Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs". In: *Proc. CADE '23*. LNCS 14132. 2023, pp. 344–364. DOI: 10.1007/978-3-031-38499-8_20.

[28] J. Kassing and J. Giesl. "From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity". In: *Log. Methods Comput. Sci.* 21.4 (2025). DOI: 10.46298/LMCS-21(4:28)2025.

[29] J.-C. Kassing, L. Spitzer, and J. Giesl. "Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting". In: *Proc. PPDP '25*. 2025. DOI: 10.1145/3756907.3756917.

[30] J.-C. Kassing and J. Giesl. "The Annotated Dependency Pair Framework for Almost-Sure Termination of Probabilistic Term Rewriting". In: *Sci. Comput. Program.* 251 (2026), p. 103417. DOI: 10.1016/j.scico.2025.103417.

[31] J.-C. Kassing, H. Nagel, A. Schlecht, and J. Giesl. "Disproving (Positive) Almost-Sure Termination of Probabilistic Term Rewriting via Random Walks". In: *CoRR* abs/2602.16522 (2026). DOI: 10.48550/arXiv.2602.16522.

[32] D. Larraz, K. Nimkar, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. "Proving Non-termination Using Max-SMT". In: *Proc. CAV '14*. LNCS 8559. 2014, pp. 779–796. DOI: 10.1007/978-3-319-08867-9_52.

[33] G. F. Lawler and V. Limic. *Random Walk: A Modern Introduction*. Cambridge University Press, June 2010.

[34] J. Leike and M. Heizmann. "Geometric Nontermination Arguments". In: *Proc. TACAS '18*. LNCS 10806. 2018, pp. 266–283. DOI: 10.1007/978-3-319-89963-3_16.

[35] N. Lommen, É. Meyer, and J. Giesl. "Targeting Completeness: Automated Complexity Analysis of Integer Programs". In: *J. Autom. Reason.* 70 (2026). DOI: 10.1007/S10817-026-09751-2.

[36] S. Lucas and R. Gutiérrez. "Use of Logical Models for Proving Infeasibility in Term Rewriting". In: *Inf. Process. Lett.* 136 (2018), pp. 90–95. DOI: 10.1016/j.ipl.2018.04.002.

[37] R. Majumdar and V. R. Sathiyanarayana. "Positive Almost-Sure Termination: Complexity and Proof Rules". In: *Proc. ACM Program. Lang.* 8.POPL (2024), pp. 1089–1117. DOI: 10.1145/3632879.

[38] M. Moosbrugger, E. Bartocci, J. Katoen, and L. Kovács. "Automated Termination Analysis of Polynomial Probabilistic Programs". In: *Proc. ESOP '21*. LNCS 12648. 2021, pp. 491–518. DOI: 10.1007/978-3-030-72019-3_18.

[39] É. Payet. "Loop Detection in Term Rewriting Using the Eliminating Unfoldings". In: *Theor. Comput. Sc.* 403 (2008), pp. 307–327. DOI: 10.1016/j.tcs.2008.05.013.

[40] É. Payet. "Non-Termination in Term Rewriting and Logic Programming". In: *J. Autom. Reason.* 68.4 (2024). DOI: 10.1007/s10817-023-09693-z.

[41] N. Saheb-Djahromi. "Probabilistic LCF". In: *Proc. MFCS '78*. LNCS 64. 1978, pp. 442–451. DOI: 10.1007/3-540-08921-7_92.

[42] F. Spitzer. *Principles of Random Walk*. Vol. 34. Springer, 2001.

[43] C. Sternagel and A. Yamada. "Reachability Analysis for Termination and Confluence of Rewriting". In: *Proc. TACAS '19*. LNCS 11427. 2019, pp. 262–278. DOI: 10.1007/978-3-030-17462-0_15.

[44] T. Takisaka, Y. Oyabu, N. Urabe, and I. Hasuo. "Ranking and Repulsing Supermartingales for Reachability in Randomized Programs". In: *ACM Trans. Program. Lang. Syst.* 43 (2021). DOI: 10.1145/3450967.

[45] A. Tiwari. "Termination of Linear Programs". In: *Proc. CAV '04*. LNCS 3114. 2004, pp. 70–82. DOI: 10.1007/978-3-540-27813-9_6.

[46] TPDB. *Termination Problem Data Base*. 2025. URL: https://github.com/TermCOMP/TPDB-ARI.

[47] M. Xu and Z.-B. Li. "Symbolic Termination Analysis of Solvable Loops". In: *J. Symb. Comput.* 50 (2013), pp. 28–49. DOI: 10.1016/j.jsc.2012.05.005.

[48] A. Yamada. "Term Orderings for Non-Reachability of (Conditional) Rewriting". In: *Proc. IJCAR '22*. 13385. 2022, pp. 248–267. DOI: 10.1007/978-3-031-10769-6_15.