On the Almost-Sure Termination of Probabilistic Counter Programs

Sergei Novozhilov[1,2], Mingqi Yang[1], Mingshuai Chen[1(✉)], Zhiyang Li[1], and Jianwei Yin[1(✉)]

[1] Zhejiang University, Hangzhou, China
{mingqiyang,m.chen,misakalzy,zjuyjw}@zju.edu.cn
[2] Hong Kong University of Science and Technology, Hong Kong, China
snovozhilov@connect.ust.hk

Abstract. This paper introduces k-d PCPs – the class of *probabilistic counter programs* with $k \in \mathbb{N}$ counter variables inducing possibly infinite-state Markov chains. We show that the universal (positive) almost-sure termination problem is *undecidable* for k-d PCPs in general, yet *decidable* for 1-d PCPs. We present an efficient decision procedure for the latter leveraging the technique of *Markov chain finitization*. Moreover, we identify several classes of k-d PCPs that are reducible to 1-d PCPs – thus their termination properties can be inferred automatically. Experiments demonstrate that our decision procedure can certify (positive) almost-sure termination – without resorting to invariants or supermartingales – of non-trivial probabilistic programs beyond the scope of existing tools.

Keywords: Probabilistic (counter) programs · (Positive) almost-sure termination · Decidability · Infinite-state Markov chains

1 Introduction

Probabilistic programming [3,26,34,38,47] is a widely adopted paradigm where programs can make probabilistic choices. Amongst others, the *almost-sure termination* (AST) problem, i.e., whether a program terminates with probability 1, is a fundamental property of probabilistic programs. Both AST and its refined notion called *positive almost-sure termination* (PAST) – whether an AST program terminates within finite expected runtime – are undecidable in general. In fact, reasoning about the termination behavior of probabilistic programs is known to be harder than that for deterministic programs; see [35]. Known classes of probabilistic programs/models admitting decidable (P)AST problems include finite-state Markov chains [2], probabilistic pushdown automata [20], and probabilistic counter programs [9]. However, these decidable fragments are not expressive enough to model randomized algorithms featuring, e.g., infinite state spaces and/or nonlinear guards; see Example 1 in Sect. 2.

S. Novozhilov and M. Yang—Both authors contributed equally to this work.

R. Piskac and Z. Rakamarić (Eds.): CAV 2025, LNCS 15932, pp. 82–104, 2025.
https://doi.org/10.1007/978-3-031-98679-6_4

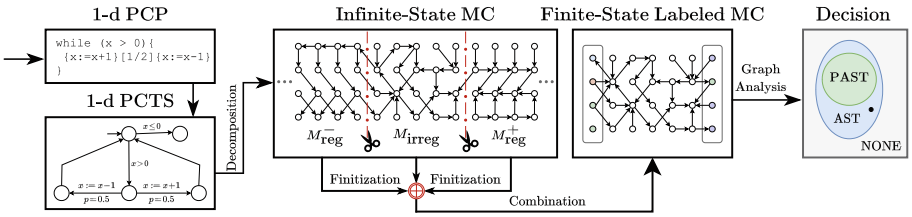


Fig. 1. The general workflow of our procedure for deciding (P)AST of 1-d PCPs.

In this paper, we introduce k-d PCPs – the class of *probabilistic counter programs* with $k \in \mathbb{N}$ counter variables (Sect. 3). This class of programs is a restricted version of the imperative probabilistic programming language pGCL [45] by allowing only assignments of the form $x := x + c$, where x is a counter variable and c is a constant. We show that k-d PCPs induce countable Markov chains with a regular structure and thereby admit an efficient analysis using various techniques established for random walks, queueing processes, finite-state Markov chains, and graph theory. In particular, we show that such Markov chains can be *finitized* while preserving their (positive) almost-sure termination properties, which yields the decidability of both AST and PAST for one-dimensional PCPs (1-d PCPs, Sect. 4) on all inputs (i.e., universal termination). We further identify several classes of k-d PCPs that are reducible to 1-d PCPs (hence called *essentially* 1*-d PCPs*), and thus their termination properties can be inferred automatically (Sect. 5). These classes include non-trivial practical programs such as the bounded retransmission protocol [30] and the Zeroconf protocol [5]. Moreover, we show that AST and PAST problems are undecidable for k-d PCPs in general, which is established by modeling a two-counter program within the 2-d PCP framework [18] ([54, Appendix F]). Finally, we present Pastry, a tool implementing the decision procedure for essentially 1-d PCPs. Experimental results on a collection of benchmarks demonstrate that our decision procedure can certify (positive) almost-sure termination of non-trivial probabilistic programs that remain out of reach for existing tools, without resorting to any form of invariants or supermartingales (Sect. 6).

Main Contributions. Our main contributions are summarized as follows.

- We introduce a new class of probabilistic programs called k-d PCPs.
- We establish the decidability of universal (P)AST for 1-d PCPs and present an efficient decision procedure via Markov chain finitization.
- We identify important classes of k-d PCPs that are reducible to 1-d PCPs.
- We implement and conduct an extensive experimental evaluation of the proposed decision procedure against existing termination analysis tools.

2 A Bird's-Eye Perspective

This section outlines our procedure for deciding (P)AST of probabilistic programs with *one* counter. As depicted in Fig. 1, our algorithm takes a 1-d PCP

```
x := 1;
while ( x^3 − 3x^2 + 2x ≥ 0 ) {
  if ( x mod 2 = 0 ) {
    { x := x + 2 } [0.6] { x := x − 1 }
  } else {
    { x := x + 1 } [0.3] { x := x − 2 }
  }
}
```

Prog. 1. The parity random walk.

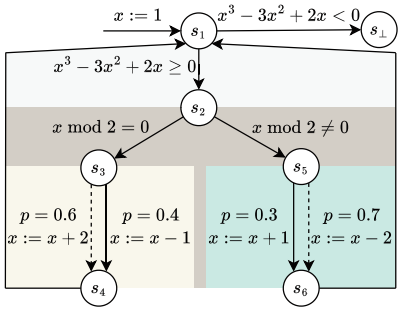


Fig. 2. The PCTS of Prog. 1.

as input and converts it to an intermediate representation called probabilistic counter transition systems (PCTSs). Each PCTS induces a countable yet potentially *infinite-state* Markov chain. Our key observation is that such Markov chain is highly *regular*: It can be decomposed into a finite component M_{irreg} and two infinite components $M^{-}_{\text{reg}}, M^{+}_{\text{reg}}$ featuring *periodic* behaviors. We then abstract the regular components leveraging the theory of quasi-birth-death processes [56], which yields *finite-state* MCs with *labeled* states. By combining these MCs with M_{irreg}, we obtain a single finite-state labeled MC. Finally, we conclude the termination of the original 1-d PCP by conducting a standard *reachability* analysis (via graph-theoretic techniques) over this MC.

We use the following example to demonstrate our decision procedure.

Example 1 (Parity Random Walk). Consider the 1-d PCP Prog. 1 that models a one-dimensional asymmetric random walk subject to two phases (based on the parity of its current position). This program is adapted from [13] by adding features of nonlinear guards and the modulo operation (mod). These features – together with the unbounded nature of x – render it infeasible to analyze the termination behavior using existing procedures such as Amber [52] and KoAT [41]. We show below how our decision procedure sketched in Fig. 1 can be employed to certify positive almost-sure termination of Prog. 1 in steps. ◁

1) Convert 1-d PCP to 1-d PCTS We first adopt a standard procedure to mechanically convert the probabilistic counter program to a probabilistic counter transition system by recursively traversing the underlying syntax tree. The so-obtained 1-d PCTS of Prog. 1 is shown in Fig. 2, where $s_\perp$ denotes the terminal state and a dashed edge denotes a condensed transition of consecutive counting steps, e.g., $s_3 \overset{p=0.6,\, x:=x+2}{\dashrightarrow} s_4$ abbreviates $s_3 \xrightarrow{p=0.6,\, x:=x+1} s' \xrightarrow{x:=x+1} s_4$ for some omitted intermediate state s'. To demonstrate the correspondence between the PCTS and the Markov chain(s) it induces (in later steps), we use colors in Fig. 2 to indicate transitions between different states.

2) Transform 1-d PCTS to Partitioned Infinite-State MC Each 1-d PCTS with state space S induces a countable Markov Chain M over the infinite state space

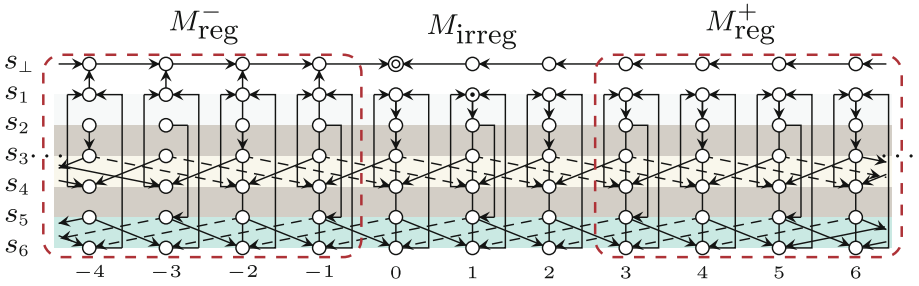
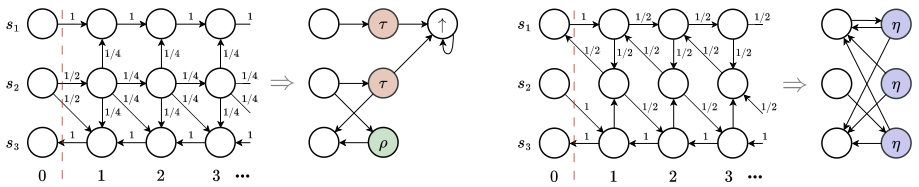


Fig. 3. The partitioned infinite-state MC induced by Prog. 1. Transition probabilities are omitted for clarity. ⊙ and ◎ mark the initial and terminal state, respectively. We artificially chain the states associated with $s_\perp$ to ensure that the MC has a unique exit state $(s_\perp, 0)$ (see Sect. 3).



(a) Transient and positive-recurrent states. (b) Null-recurrent states.

Fig. 4. An illustration of regular MC finitization using *transient* (τ), *positive-recurrent* (ρ), and *null-recurrent* (η) states. A transient state is an abstract state from which a random walk on the MC gets trapped in the regular part (signified by the artificial sink state ↑) with a positive probability. A positive-recurrent state is an abstract state from which the random walk crosses the boundary and enters the irregular part with probability 1 in a finite expected number of steps. In contrast, from a null-recurrent state, a random walk visits the irregular side with probability 1 but in an infinite expected number of steps.

$S \times \mathbb{Z}$; see Fig. 3 for the induced MC of Prog. 1. We then exploit the *regularity* of this Markov chain: Except for finitely many "middle" states around the initial counter value, the behavior of the Markov chain exhibits *periodicity*, i.e., the transitions form specific patterns ad infinitum. We can thus decompose the MC into a finite irregular component and two infinite regular components:[1]

$$M \quad = \quad M^-_{\text{reg}} \uplus M_{\text{irreg}} \uplus M^+_{\text{reg}} \,, \tag{†}$$

where $\uplus$ denotes a disjoint union. We identify the decomposition (†) by means of *guard analysis* over the PCTS. Such an analysis produces (i) the thresholds δ^- and δ^+ where to partition M; and (ii) the periods T^- and T^+ for M^-_{reg} and M^+_{reg}, respectively. Such information will be used to finitize M in the later step.

[1] The components may share states at the intersections for cross-boundary transitions.

3) Finitize the Partitioned MC Next, we finitize the infinite-state Markov chain M by abstracting its regular components M_{reg}^{-} and M_{reg}^{+} respectively into their *finite-state* counterparts $\dot{M}_{\text{reg}}^{-}$ and $\dot{M}_{\text{reg}}^{+}$, where every state is *labeled* with one of the three labels: *transient*, *positive-recurrent*, and *null-recurrent*; see Fig. 4 for an illustration. These labels are designed to preserve termination-relevant information about M whilst the rest ingredients in the regular components $M_{\text{reg}}^{+,-}$, e.g., concrete probabilities and specific states or transitions, are abstracted away from the finitized counterparts $\dot{M}_{\text{reg}}^{+,-}$. By integrating these components, we obtain a finite-state labeled Markov chain $\dot{M}$:

$$\dot{M} \quad = \quad \dot{M}_{\text{reg}}^{-} \uplus M_{\text{irreg}} \uplus \dot{M}_{\text{reg}}^{+} \,. \tag{‡}$$

Note that we drop all the explicit transition probabilities in $\dot{M}$ and keep track of only their *positivity*, i.e., zero or non-zero. This is because the (P)AST nature of finite-state MCs does not depend on the concrete probabilities thereof.

Our core labeling algorithm is inspired by the queueing theory [4] since the regular MCs $M_{\text{reg}}^{+,-}$ fall into the scope of quasi-birth-death processes (QBDs) [56]. The main technical challenge is that, in queueing theory, it is common to assume that a QBD is strongly connected (i.e., the underlying Markov chain is irreducible, cf. [8]), which is however not always the case for $M_{\text{reg}}^{+,-}$. We address this challenge by extending QBD analysis techniques with the identification of strongly connected components (SCCs) and reachability analysis between SCCs for infinite regular graphs (see Sect. 4). The resulting finite-state labeled Markov chain for Prog. 1 is depicted in Fig. 5.

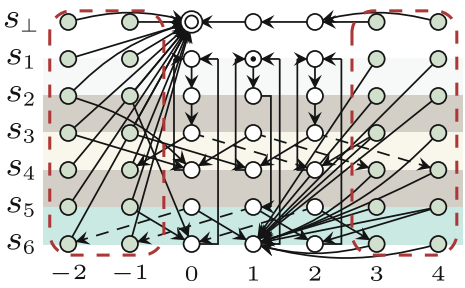


Fig. 5. Finitized labeled MC for Prog. 1.

4) Decide Termination via Reachability Analysis of the Finite-State Labeled MC. Our finitization technique guarantees that the finite-state labeled Markov chain $\dot{M}$ obtained in the previous step preserves the (positive) almost-sure termination nature of the original 1-d PCP in the following sense: The program is non-AST (and thus non-PAST) if there is a path from the initial state to a bottom SCC (distinct from the terminal state $(s_\perp, 0)$), i.e., an SCC that cannot be escaped once entered; the sink state $\uparrow$ is per se a bottom SCC. Otherwise, if any null-recurrent state is reachable from the initial state, the program is AST but non-PAST. The program is necessarily PAST if none of the above cases holds.

As per the above decision rules, we conclude that the parity random walk modeled by Prog. 1 is PAST (and thus AST), because no bottom SCC (distinct from the terminal state $(s_\perp, 0)$) nor null-recurrent state is reachable from the initial state $(s_1, 1)$; see Fig. 5. We note that, due to the *finiteness* and *non-probabilistic* nature of $\dot{M}$ (as concrete probabilities are abstracted away), the

underlying *qualitative* reachability analysis can be done effectively through standard graph-theoretic techniques.

$$
\begin{array}{rcl}
C & ::= & \texttt{skip} \\
& | & x \coloneqq x + c \\
& | & C \fatsemi C \\
& | & \{\, C \,\}\, [p]\, \{\, C \,\} \\
& | & \texttt{if}\ (\varphi)\ \{\, C \,\}\ \texttt{else}\ \{\, C \,\} \\
& | & \texttt{while}\, (\varphi)\, \{\, C \,\}
\end{array}
$$

(a) k-d PCPs

$$
\begin{array}{rcl}
\varphi & ::= & e < e \\
& | & \varphi \wedge \varphi \\
& | & \neg\varphi
\end{array}
$$

(b) Guards

$$
\begin{array}{rcll}
e & ::= & c & \\
& | & x & \\
& | & e \bmod c & (c \neq 0) \\
& | & e \div c & (c \neq 0) \\
& | & e \diamond e & (\diamond \in \{+, -, \cdot\})
\end{array}
$$

(c) Expressions

Fig. 6. Syntax of k-d PCPs and possibly nonlinear guards and expressions. Here, x is an integer *counter* variable taken from the finite set $\mathrm{Vars} = \{x_1, x_2, \ldots, x_k\}$ ranging over $\mathbb{Z}^k$, $p \in [0, 1]$ is a probability, and $c \in \mathbb{Z}$ is an integer constant. $x \bmod c$ denotes the unique number in $[0, c)$ congruent to x modulo c and $x \div c$ denotes $(x - x \bmod c)/c$. Moreover, we admit standard predicates in guards that are expressible as syntactic sugar, e.g., true, false, $\varphi \vee \varphi$, $e_1 = e_2$, $e_1 \leq e_2$, etc.

3 Problem Formulation

This section formulates our problem of certifying the (positive) almost-sure termination of probabilistic programs with $k \in \mathbb{N}$ counter variables.

Probabilistic Counter Programs. *Probabilistic counter programs* (PCPs) are a subclass of imperative programs described by the *probabilistic guarded command language* (pGCL) [44], where the assignments are restricted to a "counting" form. Specifically, the syntax of a k-d PCP C adheres to the grammar in Fig. 6. The semantics of most program constructs – including `skip`, sequential composition, conditional, and (possibly nested) loops – is standard. The *probabilistic choice* $\{\, C_1 \,\}\, [p]\, \{\, C_2 \,\}$ flips a coin with bias $p \in [0, 1]$ and executes C_1 in case the coin yields heads, and C_2 otherwise; Other forms of discrete random sampling can be mimicked by such coin flips [27]. Notice that PCPs admit *polynomial* and *modulo* operations in guards.[2] Moreover, from the termination point of view, constant assignments of the form $x \coloneqq c$ are syntactic sugar and can be encoded as certainly terminating loops $\texttt{while}\,(x > c)\,\{x \coloneqq x - 1\} \fatsemi \texttt{while}\,(x < c)\,\{x \coloneqq x + 1\}$.

We interpret the operational semantics of PCPs as *countably infinite-state Markov chains* [55]. To this end, we define *probabilistic counter transition systems* (PCTSs) as an intermediate representation:

Definition 1 (Probabilistic Counter Transition Systems). *A* PCTS *is a triple* $\langle \boldsymbol{x}, S, \mathcal{T} \rangle$*, where* $\boldsymbol{x} \in \mathbb{Z}^k$ *is a* k*-dimensional vector of counter variables,* S

[2] These operations are essential for reducing k-d PCPs to 1-d PCPs; see Sect. 5.

is a finite set of states including an initial state s_1 *(with constant initialization* $\boldsymbol{x} \coloneqq \boldsymbol{c}$*) and a terminal state*[3] $s_\perp$ *(without any outgoing transition), and* $\mathcal{T} \subset S \times \mathcal{P}(\boldsymbol{x}) \times [0,1] \times \mathbb{Z}^k \times S$ *is a set of transitions. For* $(s, g, p, \boldsymbol{u}, s') \in \mathcal{T}$*,*

- $s, s' \in S$ *are the source and target states, respectively (*s *and* s' *may coincide);*
- $g(\boldsymbol{x}) \in \mathcal{P}(\boldsymbol{x})$ *is a predicate over* $\boldsymbol{x}$*, which inherits the guard syntax of PCPs;*
- $p \in [0,1]$ *is the probability of the transition to be taken;*
- $\boldsymbol{u} \in \{-1, 0, 1\}^k$ *is an update vector, i.e., the counters are updated as* $\boldsymbol{x} \coloneqq \boldsymbol{x} + \boldsymbol{u}$ *upon taking the transition. Multi-step updates with* $\boldsymbol{u} \in \mathbb{Z}^k \setminus \{-1, 0, 1\}^k$ *are broken down to consecutive counting steps (e.g., dashed edges in* Fig. 2*).*

A transition is enabled *by its guard* $g(\boldsymbol{x})$ *if* $g(\boldsymbol{x}) = \mathsf{true}$ *for the current counter values. If multiple transitions from the same state are enabled by the same guard, then we assume that all the probabilities along these transitions sum up to 1.*

Translating PCPs to PCTSs can be done mechanically by recursively traversing the abstract syntax tree; similar procedures are standard in the literature [11].

Next, we convert a PCTS to a countable Markov chain [55] by entangling its state space with the infinitely many counter valuations: Given a PCTS $\langle \boldsymbol{x}, S, \mathcal{T} \rangle$, the induced Markov chain can be constructed as $M = \langle (S \times \mathbb{Z}^k), \mathcal{T}' \rangle$, where

- the initial state $(s_1, \boldsymbol{c}) \in S \times \mathbb{Z}^k$ of M is composed of the initial state s_1 of the PCTS and its initial counter valuations $\boldsymbol{c}$;
- the terminal state $(s_\perp, \mathbf{0}) \in S \times \mathbb{Z}^k$ of M is composed of the terminal state $s_\perp$ of the PCTS and special counter valuations $\mathbf{0}$;[4]
- for each PCTS transition $(s, g, p, \boldsymbol{u}, s') \in \mathcal{T}$, we construct MC transitions $((s, \boldsymbol{x}), p, (s, \boldsymbol{x} + \boldsymbol{u})) \in \mathcal{T}'$ for all $\boldsymbol{x}$ s.t. $g(\boldsymbol{x}) = \mathsf{true}$. Moreover, for $(s_\perp, \boldsymbol{x})$, we add transition $((s_\perp, \boldsymbol{x}), 1, (s_\perp, \boldsymbol{x} + 1))$ if $\boldsymbol{x} < \mathbf{0}$ or $((s_\perp, \boldsymbol{x}), 1, (s_\perp, \boldsymbol{x} - 1))$ if $\boldsymbol{x} > \mathbf{0}$, to ensure that $(s_\perp, \mathbf{0})$ is the unique exit of the M (cf. Fig. 3).

Remark that the terminal state $(s_\perp, \mathbf{0})$ is also the only *absorbing* state of M, i.e., a state that has no outgoing transitions. The conversion from PCPs to MCs via PCTSs has been illustrated in Sect. 2. Note that the induced MC may contain states unreachable from the initial state.

Given a possibly infinite-state Markov chain M induced by a PCP *prog*, for each path $\pi = \langle (s_1, \boldsymbol{c}), \ldots \rangle$ of M, let T_{prog} be the random variable such that $T_{prog}(\pi)$ represents the number of transitions until π reaching $(s_\perp, \mathbf{0})$; If π does not terminate by visiting $(s_\perp, \mathbf{0})$, then $T_{prog}(\pi) = \infty$. We call T_{prog} the *runtime* of *prog*. Then, the (P)AST property of a PCP can be defined as follows.

Definition 2 ((Positive) Almost-Sure Termination). *Let prog be a PCP and* $\mathbb{P}$ *be the probability measure generated by the corresponding Markov chain*

[3] For simplicity, we assume a *unique* terminal state for the PCTS (and thus a single exit for the PCP as well. For a program with multiple exits, one can append a ghost statement, e.g., `skip`, to the program end to obtain a unified exit.

[4] The concrete counter valuations do not matter. But, we assume, w.l.o.g., that both $\mathbf{0}$ and $\boldsymbol{c}$ will be partitioned into the irregular component of M in the finitization.

with initial state $(s_1, \boldsymbol{c})$ *(cf. [54, Appendix A.1]). Then, prog* terminates almost-surely *(AST) iff* $\mathbb{P}[T_{prog} < \infty] = 1$. *Moreover, prog* positive almost-surely terminates *(PAST) iff* $\mathbb{E}[T_{prog}] < \infty$.

Remark 1. Our notion of termination in Definition 2 is *non-universal* as it assumes implicitly a specific input $\boldsymbol{x} = \boldsymbol{c}$ to the program. We will show in Sect. 4.5 how our decidability result can be extended to the *universal* (positive) almost-sure termination (U(P)AST), i.e., (P)AST on *all* inputs. Nonetheless, we note that, in general, deciding AST for *one* input is as hard as ordinary termination for *all* inputs and deciding UPAST is even harder; see [35]. ◁

Example 2 (1-d Random Walk on $\mathbb{Z}$ [46]*).* Consider the following 1-d PCP:

$$x \coloneqq 1 \,;\, \texttt{while}\,(x > 0)\,\{\{\, x \coloneqq x - 1 \,\}\,[\,p\,]\,\{\, x \coloneqq x + 1 \,\}\}$$

which models a random walk on $\mathbb{Z}$ parameterized by the branching probability $p \in [0, 1]$. The program terminates as soon as the counter x becomes non-positive. It is known from the literature, e.g., [23,46], that the program is PAST if $p > 1/2$ (i.e., it terminates almost-surely with a finite expected runtime), AST but not PAST if $p = 1/2$ (i.e., it terminates almost-surely yet with an infinite expected runtime), and non-AST if $p < 1/2$ (i.e., it diverges with positive probability). The above termination behavior holds not only for the specific input $x = 1$, but also for all inputs satisfying the loop guard $x > 0$. ◁

The decision problem concerned in this paper reads as follows:

> **Problem Statement.** Given a k-d probabilistic counter program *prog* with input $\boldsymbol{x} = \boldsymbol{c}$, determine whether *prog* is PAST, AST, or neither.

We show that the problem is undecidable for k-d PCPs in general, yet decidable for 1-d PCPs as well as specific fragments of k-d PCPs. The same conclusion holds for the universal counterpart of the problem (i.e., (P)AST on *all* inputs).

4 Deciding (P)AST for 1-D PCPs

This section stablishes the decidability of (P)AST for 1-d PCPs and presents an efficient decision procedure leveraging the technique of Markov chain finitization.

Decision Procedure. Algorithm 1 outlines our procedure for deciding (P)AST for a 1-d PCP *prog* with input $x = c$. As has been exemplified in Sect. 2, our algorithm works in four main steps: (I) *Preprocessing*: It converts *prog* to the corresponding 1-d PCTS P, which induces a potentially infinite-state Markov chain M (Line 1); (II) *Decomposition*: It decomposes M – based on a guard analysis – into a finite component M_{irreg} and two infinite components $M^{-}_{\text{reg}}, M^{+}_{\text{reg}}$ featuring periodic behaviors (Lines 2 and 3); (III) *Finitization*: It abstracts M into a finite-state labeled Markov chain $\dot{M}$ which preserves the termination nature of *prog* (Lines 4 and 5); (IV) *Decision*: It determines the termination of *prog* by conducting a standard reachability analysis over $\dot{M}$ (Lines 6 to 10).

In the rest of this section, we elaborate Steps (II) to (IV) of the above decision procedure and justify its correctness and efficiency afterwards.

Algorithm 1: Deciding AST and PAST for 1-d PCPs

input: 1-d PCP *prog* (with input $x = c$).
output: PAST, AST (but non-PAST), or non-AST.

/* convert the program to a 1-d PCTS; see Section 3 */
1 $P \leftarrow$ PCP2PCTS(*prog*) ;

/* decompose the underlying infinite-state MC; see Section 4.1 */
2 $\langle \delta^-, \delta^+, T^-, T^+ \rangle \leftarrow$ **GuardAnalysis**(P) ; ▷ generate thresholds and periods
3 $\langle M_{\mathbf{irreg}}, M^-_{\mathbf{reg}}, M^+_{\mathbf{reg}} \rangle \leftarrow$ **Decompose**$(P, \delta^-, \delta^+, T^-, T^+)$;

/* obtain the finite-state labeled MC; see Section 4.2 */
4 $\dot{M}^-_{\mathbf{reg}} \leftarrow$ **Abstract**(**Label**$(M^-_{\mathbf{reg}})$) ; $\dot{M}^+_{\mathbf{reg}} \leftarrow$ **Abstract**(**Label**$(M^+_{\mathbf{reg}})$) ;
5 $\dot{M} \leftarrow \dot{M}^-_{\mathbf{reg}} \uplus M_{\mathbf{irreg}} \uplus \dot{M}^+_{\mathbf{reg}}$; ▷ combine the finite components

/* decide termination by reachability analysis of $\dot{M}$ (Section 4.3) */
6 **if** $\exists (s', c') \in \dot{M} \colon (s_1, c) \to^* (s', c')$ and $(s', c') \not\to^* (s_\perp, 0)$ **then**
7 **return** non-AST ; ▷ a run may get trapped in a bottom SCC
8 **if** $\exists (s', c') \in \dot{M} \colon \mathrm{NullRec}((s', c'))$ and $(s_1, c) \to^* (s', c')$ **then**
9 **return** AST (but non-PAST) ; ▷ a null-recurrent state is reachable
10 **return** PAST ; ▷ *prog* is necessarily PAST otherwise

4.1 Infinite-State MC Decomposition

Our decomposition of the countable Markov Chain M exploits its *regularity* over the infinite state space $S \times \mathbb{Z}$. Such regularity roots in the fact that *every (possibly nonlinear) guard of a* 1*-d probabilistic counter program is eventually periodic*:

Theorem 1 (Periodicity of Guards). *Let $P = \langle x, S, \mathcal{T} \rangle$ be the* 1*-d PCTS of prog and $\mathcal{G} = \{g \mid (s, g, p, u, s') \in \mathcal{T}\}$ be the set of guards in P. Then, every guard $g \in \mathcal{G}$ over the counter $x \in \mathbb{Z}$ is eventually periodic, i.e., there exist computable thresholds $\delta^-, \delta^+ \in \mathbb{Z}$ (with $\delta^- < \delta^+$) and periods $T^-, T^+ \in \mathbb{Z}_{>0}$ such that*

$$\forall x < \delta^- \colon\ g(x) \ = \ g(x - T^-) \quad \text{and} \quad \forall x > \delta^+ \colon\ g(x) \ = \ g(x + T^+) \ .$$

Proof (sketch). The proof is done by structural induction on the syntax tree of the guard g: For the base case with a pure polynomial predicate, we construct the thresholds $\delta^{+,-}$ using Cauchy's bound [32] associated with the periods $T^- = T^+ = 1$; For the induction step with nested operators $\div$ or mod, we iteratively replace the innermost $\div$/mod-expression with polynomial expressions while constructing $\delta^{+,-}$ and $T^{+,-}$. See complete proof in [54, Appendix B.1]. □

We refer to the construction in the proof of Theorem 1 as *guard analysis* (cf. Line 2 of Algorithm 1), which ultimately yields two thresholds $\delta^{+,-}$ and periods $T^{+,-}$ such that all guards are periodic with period T^- over $(-\infty, \delta^-)$ and T^+ over (δ^+, ∞), respectively. Recall the parity random walk in Prog. 1, our guard analysis yields $\delta^- = 0, \delta^+ = 2$ and $T^- = T^+ = 2$. Now, to describe the periodic parts beyond $[\delta^-, \delta^+]$, we introduce the model of *regular Markov chains*:

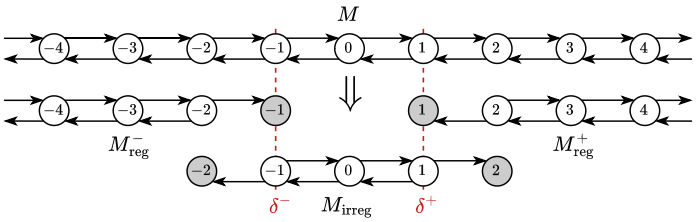


Fig. 7. The decomposition of the infinite-state MC M. Absorbing states (dashed circles) in $M^{+,-}_{\text{reg}}$ correspond to the non-absorbing states in M_{irreg} and vice versa.

Definition 3 (Regular Markov Chains). *Given a Markov chain M^+_{reg} over the state space $S \times \mathbb{Z}_{\geq \delta^+}$ with a transition probability function $\mu\colon (S \times \mathbb{Z}_{\geq \delta^+}) \times (S \times \mathbb{Z}_{\geq \delta^+}) \to [0,1]$. For $k \geq \delta^+$, we call the sets of the form $S \times \{k\}$* levels *and refer to them as level-k. M^+_{reg} is called a* regular Markov chain with period T^+ *(T^+-*periodic*, for short) if (i) there is no cross-level transition in M^+_{reg}, i.e., $\mu((s,k),(s',k')) = 0$ for any $|k'-k| > 1$; ii) all states at level-δ^+ are absorbing;*[5] *and (iii) $\mu((s,k),(s',k')) = \mu((s,k+T^+),(s',k'+T^+))$ for all $k > \delta^+, k' \geq \delta^+$.*[6]

A T^+-periodic regular Markov chain can be represented effectively as a T^+-tuple of matrices $\langle (A_i, B_i, C_i) \rangle_{i=1}^{T^+}$, where

$$\begin{aligned} A_i[s,s'] &= \mu((s,T^+ + i),(s',T^+ + i - 1)) \,, \\ B_i[s,s'] &= \mu((s,T^+ + i),(s',T^+ + i)) \,, \\ C_i[s,s'] &= \mu((s,T^+ + i),(s',T^+ + i + 1)) \,. \end{aligned} \tag{1}$$

A regular Markov chain M^-_{reg} over the state space $S \times \mathbb{Z}_{\leq \delta^-}$ with period T^- can be defined and represented analogously.

Given the thresholds $\delta^{+,-}$ and periods $T^{+,-}$ from guard analysis, a *decomposition* of infinite-state MC M (Line 3 of Algorithm 1) can be identified as

$$M \quad = \quad M^-_{\text{reg}} \uplus M_{\text{irreg}} \uplus M^+_{\text{reg}} \,, \tag{2}$$

where M_{irreg} is a finite-state MC over $S \times [\delta^- - 1, \delta^+ + 1]$ with absorbing states at levels of $\delta^- - 1$ and $\delta^+ + 1$; M^+_{reg} (resp. M^-_{reg}) is an infinite-state T^+-periodic (resp. T^--periodic) regular MC over $S \times \mathbb{Z}_{\geq \delta^+}$ (resp. $\mathbb{Z}_{\leq \delta^-}$) with absorbing states at level-δ^+ (resp. level-δ^-). An example of the decomposition is depicted in Fig. 7 and the detailed decomposition algorithm can be found in [54, Appendix C]. We justify in [54, Appendix B.2] that our decomposition as in (2) indeed yields regular Markov chains $M^{+,-}_{\text{reg}}$ in accordance with Definition 3.

[5] All outgoing transitions from level-δ^+ states are allocated to the irregular part of M.
[6] We rule out the case of $k = \delta^+$ to ensure that the level-δ^+ states are absorbing.

4.2 Regular MC Finitization

Next, our goal is to construct a *finite-state* Markov chain $\dot{M}$ via labeling and abstraction (Line 4 of Algorithm 1), which preserves termination nature of M:

$$\dot{M} \quad = \quad \dot{M}^{-}_{\text{reg}} \uplus M_{\text{irreg}} \uplus \dot{M}^{+}_{\text{reg}} \,, \tag{3}$$

where $\dot{M}^{-}_{\text{reg}}$ and $\dot{M}^{+}_{\text{reg}}$ are finitized MCs abstracting M^{-}_{reg} and M^{+}_{reg}, respectively. Due to the symmetry, we show below how to obtain $\dot{M}^{+}_{\text{reg}}$ by finitizing the T^{+}-periodic regular MC M^{+}_{reg} over $S \times \mathbb{Z}_{\geq \delta^{+}}$, whose transition probability function μ is described by the set of matrices $\langle (A_i, B_i, C_i) \rangle_{i=1}^{T^{+}}$ as per (1).

Without loss of generality, we assume $\delta^{+} = 0$ throughout the rest of this subsection; any other threshold can be aligned to 0 by shifting the MC accordingly. Moreover, we assume $T^{+} = 1$. This is because any T^{+}-periodic regular MC M^{+}_{reg} can be reduced equivalently to a 1-periodic regular MC; see [54, Appendix B.3].

Now, our task is to finitize a 1-periodic regular MC M^{+}_{reg} over $S \times \mathbb{Z}_{\geq 0}$, whose transition probability function μ is described by the matrices (A, B, C). Our approach to solving this task consists of two steps – *labeling* and *abstraction*. The former labels the level-1 states of M^{+}_{reg} whilst the latter replaces the regular part with a labeled Markov chain over the finite state space $S \times \{0, 1\}$.

The Labeling Procedure. Consider a random walk π starting from a level-1 state $(s, 1)$ of M^{+}_{reg}. The *absorbing time* of $(s, 1)$ is a random variable $T(\pi) \triangleq \min\{i \mid \pi(i) \in S \times \{0\}\}$. Let $\mathbb{P}_{(s,1)}$ be the probability measure generated M^{+}_{reg} starting from $(s, 1)$. The state $(s, 1)$ is *recurrent* if $\mathbb{P}_{(s,1)}[T < \infty] = 1$ and *transient* otherwise. In particular, we distinguish *positive-recurrence* with $\mathbb{E}_{(s,1)}[T] < \infty$ from *null-recurrence* with $\mathbb{E}_{(s,1)}[T] = \infty$. The only termination-relevant information from M^{+}_{reg} is which absorbing states at level-0 are reachable from each labeled level-1 state with positive probability. Such information can be obtained by applying [40, Theorem 7.2.3] on *quasi-birth-death processes* (QBDs) established in the queueing theory, which, however, relies on a crucial assumption that M^{+}_{reg} is *irreducible*, i.e., the underneath graph is strongly connected (see [54, Appendix A.2]). This assumption does unfortunately not hold for regular Markov chains induced by 1-d PCPs in general. Our approach drops this assumption by extending QBD analysis techniques with the identification of strongly connected components (SCCs) and reachability analysis between SCCs for infinite regular graphs. The key is to build the so-called *coupled model* – a finite-state Markov chain capturing all asymptotic properties of the infinite walks over M^{+}_{reg}.

To this end, we build a *coupled Markov chain* $\bar{M}$ over the finite state space $S \times \{-1, 0, 1\}$, where the first component represents the current state in M^{+}_{reg} and the second component indicates the *counter change* upon transiting to the current state: -1 for counter decreasing from level-k to level-$(k-1)$, 0 for counter remaining unchanged, and 1 for counter increasing from level-k to level-$(k+1)$. The transition probability function $\bar{\mu}$ of $\bar{M}$, for any $s, s' \in S$, is set as

$$\begin{array}{rcccccl}
\bar{\mu}((s,-1),(s',-1)) & = & \bar{\mu}((s,0),(s',-1)) & = & \bar{\mu}((s,1),(s',-1)) & = & A[s,s'] \,, \\
\bar{\mu}((s,-1),(s',0)) & = & \bar{\mu}((s,0),(s',0)) & = & \bar{\mu}((s,1),(s',0)) & = & B[s,s'] \,, \\
\bar{\mu}((s,-1),(s',1)) & = & \bar{\mu}((s,0),(s',1)) & = & \bar{\mu}((s,1),(s',1)) & = & C[s,s'] \,.
\end{array}$$

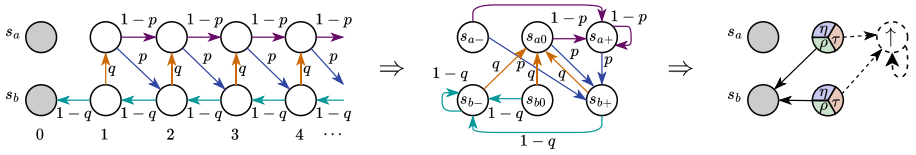


Fig. 8. Distilling from a regular MC M^+_{reg} over $S = \{s_a, s_b\} \times \mathbb{Z}_{\geq 0}$ (left) a coupled MC $\bar{M}$ over $S \times \{-1, 0, 1\} = \{s_{a-}, s_{a0}, s_{a+}, s_{b-}, s_{b0}, s_{b+}\}$ (middle). Based on the labeling of $\bar{M}$ and the abstraction procedure, we obtain the finite-state MC $\dot{M}^+_{\text{reg}}$ with labeled level-1 states (right). The concrete labels depend on the probability values p, q (see Example 3); We introduce an artificial sink state $\uparrow$ succeeding all transient (τ) states (in case of any) to signify that $\dot{M}^+_{\text{reg}}$ can get trapped.

See Fig. 8 for an example of constructing the coupled MC. The intuition behind the coupling is to *measure how much time a random walk over $\bar{M}$ spends in its "decreasing" component $S \times \{-1\}$ compared to that in the "increasing" component $S \times \{1\}$*. For the simple case where $\bar{M}$ is irreducible (assuming the irreducibility of M^+_{reg}), by the ergodic theorem for irreducible Markov chains (see [54, Appendix A.1, Theorem 5]), the ratio of the time spent in any state s converges to the stationary distribution $\bar{\gamma}(s)$. To measure the averaged imbalance between increasing- and decreasing-steps, we define the *imbalance value* $\bar{\nu}$ as

$$\bar{\nu} \quad \triangleq \quad \sum\nolimits_{s \in S \times \{1\}} \bar{\gamma}(s) \; - \; \sum\nolimits_{s \in S \times \{-1\}} \bar{\gamma}(s) \; . \tag{4}$$

Then, the sign of $\bar{\nu}$ classifies the coupled Markov chain $\bar{M}$ into three categories: transient ($\bar{\nu} > 0$), null-recurrent ($\bar{\nu} = 0$), and positive-recurrent ($\bar{\nu} < 0$). Note that this is a global property of $\bar{M}$, i.e., all its states are in the same category.

For the more involved case with *reducible* $\bar{M}$, the ergodic theorem does not apply. Nonetheless, every reducible $\bar{M}$ can be partitioned into *bottom SCCs* (BSCCs) and some *dangling states*. Each BSCC $B \subseteq S \times \{-1, 0, 1\}$ forms an irreducible Markov chain with a *local imbalance value* $\bar{\nu}_B$ obtained by restricting in (4) the state space to $B_+ = B \cap S \times 1$ and $B_- = B \cap S \times -1$.

The stationary distribution of B labels all states with $\bar{\nu}_B$. A dangling state s is labeled *transient* if any transient BSCC is reachable, *null-recurrent* if any null-recurrent BSCC is reachable, and *positive-recurrent* otherwise. The detailed labeling algorithm is in [54, Appendix C.3].

Example 3 (Labeling $\bar{M}$). Recall the *reducible* coupled Markov chain $\bar{M}$ in Fig. 8. $\bar{M}$ is composed by one BSCC $B = \{s_{a0}, s_{a+}, s_{b-}, s_{b+}\}$ and two dangling states s_{a-} and s_{b0}. For the MC induced by B, the stationary distribution is given by

$$[\bar{\gamma}(s_{a0}), \bar{\gamma}(s_{a+}), \bar{\gamma}(s_{b-}), \bar{\gamma}(s_{b+})] \;=\; \left[\frac{p \cdot q}{p+q}, \frac{(1-p) \cdot q}{p+q}, \frac{p \cdot (1-q)}{p+q}, \frac{p \cdot q}{p+q}\right] .$$

The local imbalance value $\bar{\nu}_B$ for B is calculated as

$$\bar{\nu}_B \;\triangleq\; \sum_{s \in B_+} \bar{\gamma}(s) - \sum_{s \in B_-} \bar{\gamma}(s) \;=\; \underbrace{\frac{(1-p)q}{p+q}}_{\bar{\gamma}(s_{a+})} + \underbrace{\frac{pq}{p+q}}_{\bar{\gamma}(s_{b+})} - \underbrace{\frac{p(1-q)}{p+q}}_{\bar{\gamma}(s_{b-})} \;=\; \frac{pq - p + q}{p+q} .$$

Therefore, the irreducible part B is labeled as transient (if $\bar{\nu}_B > 0$), null-recurrent (if $\bar{\nu}_B = 0$), and positive-recurrent (if $\bar{\nu}_B < 0$). Moreover, since B is the only BSCC that is reachable from the dangling states s_{a-} and s_{b0}, they two will be labeled with the same category as B (depending on the sign of $\bar{\nu}_B$). ◁

Intuitively, the coupled Markov chain $\bar{M}$ captures the asymptotic, long-term behavior of M^+_{reg}. In principle, we can label level-1 states of M^+_{reg} in accordance with the labels of $\bar{M}$.[7] However, two corner cases need to be taken into account: (I) A random walk over M^+_{reg} starting from level-1 can get trapped in a finite set of states, i.e., a finite disconnected region of the infinite graph with no way back to level-0. The coupled Markov chain would capture this behavior as null-recurrent, however, we must label such states as transient; (II) A random walk over M^+_{reg} starting from level-1 may visit level-0 immediately before exhibiting the asymptotic, long-term behavior as captured by $\bar{M}$.

We address corner case 4.2 by identifying the so-called *trappable states* at level-1 of M^+_{reg} and corner case 4.2 by allowing a "runway" of length $3 \cdot |S|$ for the random walk to enter the mode of asymptotic, long-term behaviors as captured by $\bar{M}$. Both solutions are implemented, again, via standard qualitative reachability analysis over M^+_{reg}. Due to limited space, we provide the detailed algorithm for labeling the level-1 states of M^+_{reg} (in the presence of the two corner cases) in Appendices B.4 and C.4 of [54].

The Abstraction Procedure Our abstraction procedure completes the finitization step by truncating M^+_{reg} – whose level-1 states are labeled – into a labeled finite-state MC $\dot{M}^+_{\text{reg}}$ over $S \times \{0, 1\}$ while abstracting away the infinite component beyond level-1. The key in this procedure is to identify *all possible exit states at level*-0 for a random walk starting from each state at level-1. Such random walks can be infinite, nevertheless, our "runway" tactic (formulated in [54, Appendix B.4, Lemma 2]) ensures that it suffices to consider only *finite* walks bounded by the first $3 \cdot |S|$ levels (see the detailed abstraction algorithm in [54, Appendix C.2]). An example of the finitized MC $\dot{M}^+_{\text{reg}}$ in depicted in (the right of) Fig. 8.

4.3 Deciding Termination via Reachability Analysis

The finitization step yields two finite-state MCs $\dot{M}^-_{\text{reg}}$ and $\dot{M}^+_{\text{reg}}$ with labels. By combining them with M_{irreg} à la (3), we obtain a finite-state labeled Markov chain $\dot{M}$. The following theorem establishes that the (P)AST nature of the original 1-d PCP can be determined by conducting a reachability analysis over $\dot{M}$ (suppose $s \to^* s'$ means s' is reachable from s in zero or finitely many steps):

Theorem 2 (Decision Rules). *Suppose a 1-d PCP prog induces a finite-state labeled Markov chain $\dot{M}$ over the state space $\dot{S} = S \times [\delta^- - 1, \delta^+ + 1]$ with initial state (s_1, c) and terminal state $(s_\perp, 0)$. Then,*

- *If $\exists (s', c') \in \dot{S}\colon (s_1, c) \to^* (s', c') \wedge (s', c') \not\to^* (s_\perp, 0)$, then prog is non-AST;*[8]

[7] All states in the same row of $\bar{M}$ share the same label due to the construction of $\bar{M}$.

[8] As a special case, if $(s_1, c) \to^* \uparrow$ (through a transient state), then *prog* is non-AST.

- *Otherwise, if there exists a null-recurrent state* (s', c') *such that* $(s_1, c) \to^* (s', c')$*, then prog is AST but non-PAST;*
- *Otherwise prog is PAST.*

Remark 2. The reachability analysis over $\dot{M}$ as conducted in Theorem 2 is *qualitative*: We drop all explicit transition probabilities in $\dot{M}$ and keep track of only their *positivity*, i.e., non-zero (a transition exists) or zero (no transition exists). This is because (positive) almost-sure termination of *finite-state* MCs is a topological property independent of the specific transition probabilities thereof. ◁

4.4 Correctness and Efficiency of Algorithm 1

The decidability of (P)AST problems for 1-d PCPs follows from the correctness (i.e., *soundness* and *completeness*) of our decision algorithm as established below.

Theorem 3 (Correctness). *For any* 1*-d PCP prog, Algorithm 1 terminates and returns the correct termination category of prog.*

The efficiency of our decision procedure is captured by the following theorem:

Theorem 4 (Time Complexity). *Algorithm 1 runs in time polynomial in the size of the transition system, the thresholds, and the periods, i.e.,*

$$\mathcal{O}\left(\mathrm{poly}(|S|, \delta^+ - \delta^-, T^+, T^-) + \ \mathit{StationaryDistribution}(\max(T^-, T^+) \cdot |S|)\right) \ ,$$

where StationaryDistribution(n) *denotes the complexity of an oracle for finding the stationary distribution of a Markov chain with* n *states, which can be expressed as a linear programming instance (see [54, Appendix A.1]).*

The proofs of Theorem 3 and Theorem 4 are provided in [54, Appendix B.7].

4.5 Extending the Decidability to U(P)AST

Now, we show how our decidability result can be extended to the U(P)AST problems, i.e., deciding (P)AST on *all* inputs. The key to the extension is that *the termination behavior of a* 1*-d PCP is eventually periodic w.r.t. the inputs*:

Lemma 1 (Periodicity of (P)AST). *Fix a* 1*-d PCP prog. Suppose* $PAST(k)$ *(*$AST(k)$*, resp.) is a predicate indicating whether prog is PAST (AST, resp.) on input* $x = k \in \mathbb{Z}$*. Then, there exist thresholds* $\delta_t^-, \delta_t^+ \in \mathbb{Z}$ *(with* $\delta_t^- < \delta_t^+$*) and periods* T_t^-*,* $T_t^+ \in \mathbb{Z}_{>0}$ *such that*

$$\forall x < \delta_t^- : \ PAST(x) = PAST(x - T_t^-) \quad \text{and} \quad AST(x) = AST(x - T_t^-)$$
$$\forall x > \delta_t^+ : \ PAST(x) = PAST(x + T_t^+) \quad \text{and} \quad AST(x) = AST(x + T_t^+) \ .$$

Moreover, the thresholds and periods are computable (cf. Theorem 1) as

$$\delta_t^- = \delta^- - 2^{T^- \cdot |S|}, \quad \delta_t^- = \delta^+ + 2^{T^+ \cdot |S|}, \quad T_t^- \leq 2^{T^- \cdot |S|}, \quad T_t^+ \leq 2^{T^+ \cdot |S|} \ .$$

The decidability of U(P)AST for 1-d PCPs then follows immediately from Lemma 1, since we can enumerate all possible inputs within $[\delta_t^- - T_t^-, \delta_t^+ + T_t^+]$.

5 Decidable Fragments for Multidimensional PCPs

Below, we identify four classes of k-d PCPs that can be reduced to 1-d PCPs:

(i) *All But One Counters are Bounded*: k-d PCPs with counter variables $\text{Vars} = \{x_1, x_2, \ldots, x_{k-1}, y\}$, for which there exists constant B such that $|x_i| < B$ for all $i = 1, \ldots, k-1$ in all reachable program states;

(ii) *Monotone Counters*: k-d PCPs with counter variables $\text{Vars} = \{x_1, \ldots, x_m, y_1, \ldots, y_n, z\}$, for which all counters x_i are always non-decreasing while all counters y_i are always non-increasing. An additional requirement is that all atomic propositions in guards must depend on only one variable;

(iii) *Conditionally Bounded Counters*: k-d PCPs with counter variables $\text{Vars} = \{x_1, x_2, \ldots, x_{k-1}, y\}$, for which there exist constants $A_i, B_i, C_i, D_i \in \mathbb{Z}$ such that $|A_i \cdot x_i - B_i \cdot y - C_i| \leq D_i$ for all $i = 1, \ldots, k-1$ in all reachable program states. In other words, all counters lie in a bounded neighborhood of a one-dimensional affine subspace of $\mathbb{Z}^k$;

(iv) *Constant Probability Programs*: The decidable fragment described in [25] can be rewritten as a 1-d PCP via a linear variable substitution.

We call the above classes and their mixtures *essentially 1-d PCPs*. The detailed techniques for reducing essentially 1-d PCPs to 1-d PCPs can be found in [54, Appendix D]. Below, we demonstrate the reduction using a real-world program; additional examples are provided in [54, Appendix E].

Example 4 (Zeroconf Protocol [5,23]). Consider the randomized IPv4 Zeroconf protocol for self-establishing IP connections via bounded retries:

$$
\begin{array}{l}
\text{start} \coloneqq 1 \,;\; \text{established} \coloneqq 0 \,;\; \text{probe} \coloneqq 0 \,; \\
\texttt{while}\,(\,\text{start} \leq 1 \wedge \text{established} \leq 0 \wedge \text{probe} < 4\,)\,\{ \\
\quad \texttt{if}\,(\,\text{start} = 1\,)\,\{\{\,\text{start} \coloneqq 0\,\}\,[0.5]\,\{\,\text{start} \coloneqq 0 \,;\, \text{established} \coloneqq 1\,\}\,\} \\
\quad \texttt{else}\,\{\,\{\,\text{probe} \coloneqq \text{probe} + 1\,\}\,[0.001]\,\{\,\text{start} \coloneqq 1 \,;\, \text{probe} \coloneqq 0\,\}\,\}\,\}\,.
\end{array}
$$

Observe that all the three variables are bounded and thus – as a special case of Class (i) – the program is an essentially 1-d PCP: By introducing a new variable $z = 16 \cdot \text{probe} + (\text{start} + 2) + 4 \cdot (\text{established} + 2)$ and applying the substitutions

$$
\text{start} \;\mapsto\; (z \bmod 4) - 2, \quad \text{established} \;\mapsto\; (z \div 4) \bmod 4 - 2, \quad \text{probe} \;\mapsto\; z \div 16,
$$

we obtain the reduced 1-d PCP:

$$
\begin{array}{l}
z \coloneqq 10 \,; \\
\texttt{while}\,(\,(z \bmod 4) - 2 \leq 1 \wedge (z \div 4) \bmod 4 - 2 \leq 0 \wedge z \div 16 < 4\,)\,\{ \\
\quad \texttt{if}\,(\,(z \bmod 4) - 2 = 1\,)\,\{\{\,z \coloneqq z - 1\,\}\,[0.5]\,\{\,z \coloneqq z - 1 \,;\, z \coloneqq z + 4\,\}\,\} \\
\quad \texttt{else}\,\{\,\{\,z \coloneqq z + 16\,\}\,[0.001]\,\{\,z \coloneqq z + 1 \,;\, \texttt{while}\,(\,z \div 16 > 0\,)\,\{\,z \coloneqq z - 16\,\}\,\}\,\}\,.
\end{array}
$$

Hence, termination behaviors of the protocol can be inferred automatically. ◁

6 Implementation and Experimental Results

We have implemented our techniques in Python as a prototypical tool called Pastry[9] – the Positive Almost-Sure Termination pRototYpe. By interfacing with `Probably` [49] for parsing probabilistic programs, `Sympy` for solving systems of linear equations, `NetworkX` [28] for graph analysis, and `SciPy` [63] for representing and manipulating sparse graphs as COO-encoded matrices, Pastry decides whether a given essentially 1-d PCP with possibly infinite states is PAST, AST (but non-PAST), or non-AST. All experiments are conducted on a 3.22 GHz Apple M1 Pro processor with 16 GB RAM running macOS Sequoia.

Remark 3. Can one automatically check if a given k-d PCP belongs to one of the four classes of essentially 1-d PCPs as identified in Sect. 5? For Classes (ii) and (iv), the check can be done automatically in a purely *syntactic* way. For Classes (i) and (iii), the check can be automated by leveraging techniques for *synthesizing invariants* of the form $|x_i| < B$ or $|A_i \cdot x_i - B_i \cdot y - C_i| \leq D_i$. For instance, for linear programs, the check can be automated by a reduction to nonlinear constraint solving via Farkas' lemma [15] (though the procedure may not be complete due to integer-valued (counter) variables x_i and y). Our current implementation of Pastry automates the check for Classes (ii) and (iv).

Baselines and Benchmarks. We compare Pastry in terms of applicability and efficiency against four state-of-the-art tools for deciding termination of probabilistic programs: Amber [52] for analyzing *prob-solvable loops*, KoAT1 [25] for analyzing *constant probability programs* (as a subset of essentially 1-d PCPs), and KoAT2 [48] and Absynth [53] – both are for computing *upper bounds on expected costs* and thus are limited to certifying PAST. Our benchmark suite is compiled from seven sources in the literature (see Table 1). To facilitate the comparison, we initialize programs with open inputs (i.e., uninitialized variables and/or parameters) using randomly drawn inputs.

Experimental Results. As reported in Table 1, Pastry suffices to determine the termination category of *all but 2 benchmarks*, including probabilistic counter programs featuring complex control flows, such as nested loops (e.g., `two_endpoints`) and conditional branches within loops (e.g., `generalized_rw` and `1d_poly_rw`). These program features pose significant challenges for other decision procedures like Amber and KoAT1. Such benchmarks can be handled by techniques based on stochastic invariants, e.g., [12], which admit, however, only a relatively complete template-based approach for algorithmic synthesis. Moreover, compared to KoAT2 and Absynth which can certify PAST only, Pastry is capable of deciding both (non-)PAST and (non-)AST within a unified framework. We note that `geometric_gauss` and `polynomial_nast` are out of reach by Pastry (yet can be handled by Amber) as they feature continuous sampling and nonlinear updates, respectively (thus beyond PCPs).

[9] Available at https://github.com/FICTION-ZJU/Pastry.

Table 1. Experimental results (–: inapplicability; TO: timeout in 90 s).

Benchmark	Src	Dim	PAST	AST	Time (s)[1]				
					Pastry	Amber	KoAT1	KoAT2[2]	Absynth
symmetric_rw1	[52]	1	✗	✓	0.171	0.028	0.003	–	–
symmetric_rw2		1	✗	✓	0.137	0.027	0.003	–	–
biased_rw1		1	✗	✗	0.078	0.027	0.003	–	–
biased_rw2		1	✗	✗	0.419	0.073	0.004	–	–
biased_rw3		1	✗	✗	0.430	0.079	0.004	–	–
biased_rw4		1	✓	✓	0.081	0.023	0.032	0.888	0.011
binomial1		1	✗	✗	0.076	0.028	0.004	–	–
binomial2		1	✓	✓	0.080	0.022	0.031	0.877	0.012
geometric		1	✓	✓	0.074	0.018	0.029	0.661	0.009
2d_bounded_rw		2	✓	✓	3.390	0.031	–	–	–
geometric_gauss		1	✓	✓	–	0.022	–	–	–
polynomial_nast		2	✗	✗	–	1.590	–	–	–
asymmetric_rw1	[25]	1	✓	✓	0.075	0.024	0.031	0.980	0.011
complex_roots		1	✓	✓	0.137	0.025	0.056	9.459	0.028
high_multiplicity		1	✓	✓	0.132	0.028	0.052	7.137	0.027
neg_binomial		1	✓	✓	0.073	0.024	0.030	0.924	0.012
nast_prog		1	✗	✗	0.160	0.046	0.003	–	–
npast_prog		1	✗	✓	0.109	0.040	0.004	–	–
dir_term		1	✓	✓	0.078	–	0.034	–	0.016
irr_runtime		1	✓	✓	0.081	0.024	0.036	0.942	0.012
tortoise_hare_un		1	✓	✓	0.467	0.024	0.130	TO	0.133
tortoise_hare		2	✓	✓	2.450	–	0.159	TO	0.773
tortoise_hare_dt		2	✓	✓	0.185	–	0.032	1.047	–
generalized_rw	[12]	1	✗	✗	0.236	–	–	–	–
two_endpoints		1	✗	✗	0.209	–	–	–	–
infinite_loop		1	✗	✗	0.030	–	–	–	–
two_loops		1	✗	✗	0.151	–	–	–	–
ast_loop		1	✓	✓	0.078	0.022	0.032	1.193	0.013
ast_rw		1	✓	✓	0.174	0.021	0.042	3.114	0.020
biased_rw5		1	✗	✗	0.178	–	–	–	–
skewed_rw		1	✗	✗	0.229	–	–	–	–
asymmetric_rw2		1	✓	✓	0.088	0.031	0.039	0.929	0.011
1d_poly_rw		1	✗	✗	3.720	–	–	–	–
catmouse	[1]	1	✓	✓	0.103	–	–	1.079	0.020
speedpldi4		1	✓	✓	0.112	–	–	1.448	0.024
insertsort		2	✓	✓	0.303	–	–	3.462	0.030
speedpldi2		2	✓	✓	0.135	–	–	2.972	0.020
speedpldi3		2	✓	✓	0.226	–	–	2.235	0.026
counterex1b		2	✓	✓	0.361	–	–	6.758	0.051
Knuth-Yao_dice	[36]	2	✓	✓	0.499	–	–	12.712	53.015
brp_protocol	[30]	3	✓	✓	0.311	–	–	2.135	0.025
zeroconf	[5]	3	✓	✓	0.264	–	–	3.727	0.036

[1] Timings for Pastry, Amber, and KoAT1 are the total time for deciding both PAST and AST, whilst timings for KoAT2 and Absynth are for certifying PAST only.
[2] KoAT2 takes inputs in the form of *integer transition systems* (ITS), where semantically equivalent programs may induce ITS with different structures. Thus, for some program that KoAT2 fails to certify PAST, e.g., `dir_term`, KoAT2 may succeed on its transformed version (by, e.g., eliminating loopy structures).

In terms of efficiency, PASTRY performs on par with its competitors for most 1-d programs yet less efficient for certain multi-dimensional programs. This is primarily due to an unoptimized procedure for constructing potentially large graphs for multi-dimensional programs.

We further note that the efficiency of PASTRY depends on the *size of counter increments* as it affects the size of the underlying Markov chain to be constructed. To investigate the dependency in a quantitative aspect, we examine a simple program with parameters α, β, and p:

$$x \coloneqq 10 \,;\, \texttt{while}\,(x > \alpha)\,\{\{\, x \coloneqq x + \beta \,\}\,[\,p\,]\,\{\, x \coloneqq x - \beta \,\}\}\ .$$

The expected theoretical time complexity for analyzing this program is $O(\beta^2) + O(|\alpha \cdot \beta|)$. Our empirical results (by varying the parameters α, β, and p) agree with this theoretical estimation; see detailed analysis in [54, Appendix G].

7 Related Work

Program Termination Landscape. As a cornerstone problem in computability, the results concerning termination can be broadly split into four categories: (i) establishing the *undecidability* of the termination problem in a model of computation. Historically the first results were obtained for Turing machines [62] and lambda calculus [14]. Later on, much simpler yet undecidable models were discovered. The most relevant to our work is the counter-machine model. These machines operate on integer registers, and the instructions generally involve incrementing, decrementing, copying, and testing the values in these registers. Different variations of counter machines have been proposed by, e.g., Hermes [31], Ershov [19], Péter [59], Minsky [18,37,50,51], Lambek [39], Shepherdson and Sturgis [60]. Some key differences in these models include: the presence of an accumulator (a special register for arithmetic operations), the use of direct vs. indirect addressing, the availability of instructions for incrementing, decrementing, and comparing the values in the registers, and whether or not the machine allows for unconditional jumps, conditional jumps, or both; (ii) establishing the *hardness* of the termination problem w.r.t. a known hypothesis, such as leveraging the Skolem problem [22,29] to loop termination analysis [57]; (iii) establishing *decidability* results in special cases, such as the decidability of termination for probabilistic pushdown automata (pPDA) [20] or programs in a special restrictive syntax such a single while loop with affine assignments [6,7,24,58,61]; (iv) finding an easily verifiable termination *certificate* (also called a proof). Within this category, we can identify two sub-categories: (a) when the certificate always exists, but the search procedure might be undecidable in general, as for the intersection types [16,17] or recent works on the supermartingale rules [10,42,43,46]; (b) when there is no guarantee of existing certificates, but if exists, it can be found efficiently as for special cases of ranking supermartingales (SMs), e.g., polynomial SMs [11], linear lexicographic ranking SMs [1], repulsing SMs [13]; or for the case of resource-aware programming languages [33].

PCPs in the Landscape. We place our results on k-d PCPs to Categories 7 and 7, especially in the probabilistic setting. For the undecidability category, the most relevant results are on the undecidability of counter machines (with two counters, decrement and jump instructions) [18,37,51] as our proof is based on encoding an arbitrary 2-counter machine inside a 2-d PCP program. For the decidability category, the most relevant works are on probabilistic pushdown automata (pPDA) [20], one-counter automata (pOC) [9], recurrent Markov Chains (RMCs) [21], and one-counter Markov decision procedures (OC-MDPs) [8]. Our work is orthogonal to these models as we allow a wide range of guard expressions that cannot be simulated (or at least we are not aware of the simulation) by pPDA, pOC, RMCs, or OC-MDPs. The non-trivial guards play a crucial role in rewriting the special cases of k-d PCPs into 1-d PCPs. For example, it allows for encoding constant probability programs [25] into 1-d PCPs using a simple transformation. Moreover, we provide the first open-source decision-procedure implementation for PCPs that could be useful for the mentioned models.

8 Conclusion

We have investigated the (positive) almost-sure termination problem of probabilistic counter programs inducing possibly infinite-state Markov chains. Our work establishes the decidability of U(P)AST problems for (essentially) 1-d PCPs – a significant contribution as obtaining stronger decidability results for new classes of programs is inherently challenging. Conversely, we show that these problems are undecidable for k-d PCPs in general. For the decidable class of essentially 1-d PCPs, we developed an efficient decision procedure that relies on decomposing and finitizing the underlying Markov chains. Experimental results demonstrate that our procedure effectively determines (P)AST for non-trivial probabilistic programs, including cases beyond the reach of existing tools.

Future directions include the extension of our results to (i) reasoning about *quantitative aspects* of probabilistic programs beyond almost-sure termination, e.g., termination probabilities and expected runtimes; and (ii) dealing with programs featuring nondeterminism [44] and/or conditioning [55].

Acknowledgments. This work has been partially funded by the ZJNSF Major Program (No. LD24F020013), by the Fundamental Research Funds for the Central Universities of China (No. 226-2024-00140), and by the ZJU Education Foundation's Qizhen Talent program.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Agrawal, S., Chatterjee, K., Novotný, P.: Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. Proc. ACM Program. Lang. **2**(POPL), 34:1–34:32 (2018)
2. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
3. Barthe, G., Katoen, J., Silva, A. (eds.): Foundations of Probabilistic Programming. Cambridge University Press (2020)
4. Bhat, U.N.: An Introduction to Queueing Theory: Modeling and Analysis in Applications. Statistics for Industry and Technology, Birkhäuser, Boston, MA (2015)
5. Bohnenkamp, H.C., van der Stok, P., Hermanns, H., Vaandrager, F.W.: Cost-optimization of the IPV4 Zeroconf protocol. In: DSN, pp. 531–540. IEEE Computer Society (2003)
6. Bozga, M., Iosif, R., Konecný, F.: Deciding conditional termination. Log. Methods Comput. Sci. **10**(3), 252–266 (2014)
7. Braverman, M.: Termination of integer linear programs. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 372–385. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_34
8. Brázdil, T., Brozek, V., Etessami, K., Kucera, A., Wojtczak, D.: One-counter Markov decision processes. In: SODA, pp. 863–874. SIAM (2010)
9. Brázdil, T., Kiefer, S., Kučera, A.: Efficient analysis of probabilistic programs with an unbounded counter. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 208–224. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_18
10. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 511–526. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_34
11. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through Positivstellensatz's. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_1
12. Chatterjee, K., Goharshady, A.K., Meggendorfer, T., Zikelic, D.: Sound and complete certificates for quantitative termination analysis of probabilistic programs. In: CAV (1). LNCS, vol. 13371, pp. 55–78. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_4
13. Chatterjee, K., Novotný, P., Zikelic, D.: Stochastic invariants for probabilistic termination. In: POPL, pp. 145–160. ACM (2017)
14. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
15. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_39
16. Coppo, M., Dezani-Ciancaglini, M.: A new type assignment for λ-terms. Arch. Math. Log. **19**(1), 139–156 (1978)
17. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Functional characters of solvable terms. Math. Log. Q. **27**(2–6), 45–58 (1981)

18. Dudenhefner, A.: Certified decision procedures for two-counter machines. In: FSCD. LIPIcs, vol. 228, pp. 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
19. Ershov, A.P.: On operator algorithms. Doklady Akademii Nauk SSSR **122**, 967–970 (1958). English translation in Automat. Express **1**, 20–23 (1959)
20. Esparza, J., Kucera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: LICS, pp. 12–21. IEEE Computer Society (2004)
21. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. J. ACM **56**(1), 1:1–1:66 (2009)
22. Everest, G., van der Poorten, A.J., Shparlinski, I.E., Ward, T.: Recurrence Sequences, Mathematical surveys and monographs, vol. 104. American Mathematical Society (2003)
23. Feng, S., Chen, M., Su, H., Kaminski, B.L., Katoen, J., Zhan, N.: Lower bounds for possibly divergent probabilistic programs. Proc. ACM Program. Lang. **7**(OOPSLA1), 696–726 (2023)
24. Frohn, F., Giesl, J.: Termination of triangular integer loops is decidable. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 426–444. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_24
25. Giesl, J., Giesl, P., Hark, M.: Computing expected runtimes for constant probability programs. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 269–286. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_16
26. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE, pp. 167–181. ACM (2014)
27. Gryszka, K.: From biased coin to any discrete distribution. Period. Math. Hung. **83**(1), 71–80 (2021)
28. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkX. Tech. rep, Los Alamos National Laboratory (LANL), Los Alamos, NM, USA (2008)
29. Halava, V., Harju, T., Hirvensalo, M.: Positivity of second order linear recurrent sequences. Discret. Appl. Math. **154**(3), 447–451 (2006)
30. Helmink, L., Sellink, M., Vaandrager, F.W.: Proof-checking a data link protocol. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 127–165. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58085-9_75
31. Hermes, H.: Die universalität programmgesteuerter rechenmaschinen. Mathematisch-Physikalische Semesterberichte **4**, 42–53 (1954)
32. Hirst, H.P., Macey, W.T.: Bounding the roots of polynomials. Coll. Math. J. **28**(4), 292–295 (1997)
33. Hoffmann, J., Aehlig, K., Hofmann, M.: Resource aware ML. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 781–786. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_64
34. Holtzen, S., den Broeck, G.V., Millstein, T.D.: Scaling exact inference for discrete probabilistic programs. Proc. ACM Program. Lang. **4**(OOPSLA), 140:1–140:31 (2020)
35. Kaminski, B.L., Katoen, J., Matheja, C.: On the hardness of analyzing probabilistic programs. Acta Informatica **56**(3), 255–285 (2019)
36. Knuth, D.E., Yao, A.C.: The complexity of nonuniform random number generation. In: Algorithms and Complexity: New Directions and Recent Results. Academic Press (1976)
37. Korec, I.: Small universal register machines. Theor. Comput. Sci. **168**(2), 267–301 (1996)

38. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
39. Lambek, J.: How to program an infinite abacus. Math. Bull. **4**(3), 295–302 (1961)
40. Latouche, G., Ramaswami, V.: Introduction to Matrix Analytic Methods in Stochastic Modeling. SIAM, ASA-SIAM Series on Statistics and Applied Mathematics (1999)
41. Lommen, N., Meyer, É., Giesl, J.: Control-flow refinement for complexity analysis of probabilistic programs in Koat (short paper). In: IJCAR (1). LNCS, vol. 14739, pp. 233–243. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_14
42. Majumdar, R., Sathiyanarayana, V.R.: Positive almost-sure termination: complexity and proof rules. Proc. ACM Program. Lang. **8**(POPL), 1089–1117 (2024)
43. Majumdar, R., Sathiyanarayana, V.R.: Sound and complete proof rules for probabilistic termination. Proc. ACM Program. Lang. **9**(POPL), 1871–1902 (2025)
44. McIver, A., Morgan, C.: Abstraction, refinement and proof for probabilistic systems. Monographs in Computer Science, Springer (2005). https://doi.org/10.1007/b138392
45. McIver, A., Morgan, C.: Introduction to PGCL: its logic and its model. In: Refinement Techniques in Software Engineering. Springer (2005). https://doi.org/10.1007/0-387-27006-X_1
46. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.: A new proof rule for almost-sure termination. Proc. ACM Program. Lang. **2**(POPL), 33:1–33:28 (2018)
47. van de Meent, J.W., Paige, B., Yang, H., Wood, F.: An introduction to probabilistic programming (2021). https://arxiv.org/abs/1809.10756
48. Meyer, F., Hark, M., Giesl, J.: Inferring expected runtimes of probabilistic integer programs using expected sizes. In: TACAS 2021. LNCS, vol. 12651, pp. 250–269. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_14
49. Meyer, P.J.: Probably: probabilistic guarded command language (PGCL) documentation (2023). https://philipp15b.github.io/probably/pgcl.html. Accessed 25 Oct 2023
50. Minsky, M.: Recursive unsolvability of post's problem. Tech. Rep. 54G-0023, Massachusetts Institute of Technology, Lincoln Laboratory (1954)
51. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs, NJ, USA (1967)
52. Moosbrugger, M., Bartocci, E., Katoen, J.-P., Kovács, L.: The probabilistic termination tool amber. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 667–675. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_36
53. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: resource analysis for probabilistic programs. In: PLDI, pp. 496–512. ACM (2018)
54. Novozhilov, S., Yang, M., Chen, M., Li, Z., Yin, J.: On the almost-sure termination of probabilistic counter programs (2025). https://hal.science/hal-05082395, hal preprint hal-05082395
55. Olmedo, F., Gretz, F., Jansen, N., Kaminski, B.L., Katoen, J., McIver, A.: Conditioning in probabilistic programming. ACM Trans. Program. Lang. Syst. **40**(1), 4:1–4:50 (2018)
56. Ost, A.: Quasi-birth-and-death processes. In: Performance of Communication Systems: A Model-Based Approach with Matrix-Geometric Methods, pp. 51–102. Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04421-6_4
57. Ouaknine, J., Worrell, J.: On linear recurrence sequences and loop termination. ACM SIGLOG News **2**(2), 4–13 (2015)

58. Ouaknine, J., Sousa-Pinto, J., Worrell, J.: On termination of integer linear loops 2015 (2014)
59. Péter, R.: Graphschemata und rekursive funktionen. Dialectica **12**, 373 (1958)
60. Shepherdson, J.C., Sturgis, H.E.: Computability of recursive functions. J. ACM **10**(2), 217–255 (1963)
61. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_6
62. Turing, A.M.: On computable numbers, with an application to the entscheidungs problem. Proc. London Math. Soc. **s2-42**(1), 230–265 (1937)
63. Virtanen, P., et al.: SciPy 1.0 Contributors: SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nat. Methods **17**, 261–272 (2020)