Almost Sure Productivity[*]

Alejandro Aguirre[1], Gilles Barthe[1], Justin Hsu[2], and Alexandra Silva[2]

1 **IMDEA Software, Madrid, Spain**
2 **University College London, London, UK**

Abstract

We define *Almost Sure Productivity (ASP)*, a probabilistic generalization of the productivity condition for coinductively defined structures. Intuitively, a probabilistic coinductive stream or tree is ASP if it produces infinitely many outputs with probability 1. Formally, we define almost sure productivity using a final coalgebra semantics of programs inspired from Kerstan and König. Then, we introduce a core language for probabilistic streams and trees, and provide two approaches to verify ASP: a sufficient syntactic criterion, and a reduction to model-checking pCTL* formulas on probabilistic pushdown automata. The reduction shows that ASP is decidable for our core language.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages, D.3.1 Formal Definitions and Theory

Keywords and phrases Coinduction, Probabilistic Programming, Productivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.

1 Introduction

The study of probabilistic programs has a long history, especially in connection with semantics [25] and verification [26, 20, 32]. Over the last decade, the field of probabilistic programming has undergone impressive developments, with the emergence of practical probabilistic programming languages and novel applications in machine learning, privacy-preserving data mining, and modeling of complex systems. On the more theoretical side, many semantical and syntactic tools have been developed for modeling and verifying probabilistic programs. In particular, significant attention has been devoted to termination of probabilistic programs, focusing on the complexity of the different termination classes [22], and on practical methods for proving that a program terminates [18, 29, 2, 31]. The latter class of works generally focuses on *almost sure termination*, which guarantees that a program terminates with probability 1.

Coinductive probabilistic programming is an emerging computational paradigm that extends probabilistic programming to infinite objects such as streams and infinite trees and provides a natural setting for programming and reasoning about probabilistic infinite processes such as Markov chains or Markov decision processes. Rather surprisingly, the study of coinductive probabilistic programming was initiated only very recently [3], and little is known about generalizations of coinductive concepts and methods to the probabilistic setting. In this paper, we focus on the notion of *productivity*, which informally ensures that one can compute arbitrarily precise finite approximations of infinite objects in finite

[*] This is the full version of the paper.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Don Sannella;
Article No. ; pp. :1–:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

time. Productivity has been studied extensively for standard, non-probabilistic coinductive languages [21, 16, 1, 13, 8], and the probabilistic setting introduces new challenges.

Contributions

Our first contribution is definitional. We introduce *almost sure productivity* (ASP), a probabilistic counterpart to productivity. In the simplest setting of streams, a probabilistic computation is almost surely productive if it produces an infinite stream of outputs with probability 1. For instance, consider the stream defined by the equation

$$\sigma = (a : \sigma) \oplus_p \sigma$$

Viewed as a program, this stream repeatedly flips a coin with bias $p \in (0, 1)$, producing the value a if the coin comes up heads and retrying if the coin comes up tails. This computation is almost surely productive since the probability it fails to produce outputs for n steps is $(1 - p)^n$, which tends to zero as n increases. In contrast, consider the stream defined by the equation

$$\sigma = \bar{a} \oplus_p \epsilon$$

This computation flips a single biased coin and returns an infinite stream of a's if the coin comes up heads, and the empty stream ϵ if the coin comes up tails. This process is *not* almost surely productive since its probability of outputting an infinite stream is only p, which is strictly less than 1.

We define almost sure productivity for an abstract language that can be equipped with a final coalgebra semantics, in the style of Kerstan and König [23], and give a semantic characterization. Although intuitive, the definition involves measure-theoretic technicalities (Section 3). We instantiate our semantics on a concrete, core probabilistic language for computing over streams and trees (Section 4). Then, we propose two methods for proving almost sure productivity.

1. We begin with a syntactic method that assigns to each expression e a measure in $\mathbb{R}$ (Section 5). Intuitively, the measure represents the expected difference between the number of outputs produced and consumed per evaluation step of the expression. For instance, the computation that repeatedly flips a fair coin and outputs a value if the coin is heads has measure $\frac{1}{2}$—with probability $1/2$ it produces an output, with probability 0 it produces no outputs. More complex terms in our language can also consume outputs internally, leading to possibly negative values for the productivity mesaure.

   We show that every expression whose measure is strictly positive is almost surely productive. The proof of soundness of the method uses concentration results from martingale theory. However, the method is incomplete—it does not yield any information for expressions with non-positive measure.

2. To give a more sophisticated analysis, we give an encoding into probabilistic model-checking for both streams and trees (Section 6). We define an interpretation of expressions as probabilistic pushdown automata and show that almost sure productivity of the expression can be characterized by a logical formula in the qualitative fragment of pCTL*. This fragment is known to be decidable [28], giving a decision procedure for the almost sure productivity property.

We consider more advanced generalizations and extensions in Section 7, survey related work in Section 8, and conclude in Section 9.

2 Mathematical Preliminaries

This section reviews basic notation and definitions from measure theory and category theory (though we assume some familiarity with these areas).

Given a set A we will denote by $A_\perp$ the coproduct of A with a one-element set containing a distinguished element $\perp$, i.e., $A_\perp = A + \{\perp\}$.

Streams, Trees, Coalgebra

We will denote by O^ω the set of infinite streams of elements of O (alternatively characterized as functions $\mathbb{N} \to O$). We have functions $\mathsf{head}\colon O^\omega \to O$ and $\mathsf{tail}\colon O^\omega \to O^\omega$ that enable observation of the elements of the stream. In fact, they provide O^ω with a one-step structure that is canonical: given any two functions $h\colon S \to O$ and $t\colon S \to S$ there exists a *unique* stream function associating semantics to elements of S:

$$\begin{tikzcd}
S \arrow[r, dashed, "\llbracket - \rrbracket"] \arrow[d, "<h{,}t>"'] & O^\omega \arrow[d, "<\mathsf{head}{,}\mathsf{tail}>"] \\
O \times S \arrow[r, dashed, "id \times \llbracket - \rrbracket"] & O \times O^\omega
\end{tikzcd}$$

Formally, this uniqueness property is known as *finality*: O^ω is the *final coalgebra* of the functor $F(X) = O \times X$ and the above diagram gives rise to a coinductive definition principle. A similar principle can be obtained for infinite binary trees and other algebraic datatypes. The above diagrams are in the category of sets and functions, but infinite streams and trees have a very rich algebraic structure and they are also the carrier of final coalgebras in other categories. For the purpose of this paper, we will be particularly interested in a category where the maps are probabilistic—the Kleisli category of the distribution (or *Giry*) monad.

Probability Distributions, σ-algebras, Measurable Spaces

Given an arbitrary set X we call a set Σ of subsets of X a *σ-algebra* if it contains the empty set and is closed under complement and countable union. A *measurable space* is a pair (X, Σ). A *probability measure* or distribution μ over such a space is a function $\mu : \Sigma \to [0, 1]$ that assigns probabilities $\mu(A) \in [0, 1]$ to the *measurable sets* $A \in \Sigma$, and satisfies the following conditions:

- $\mu(X) = 1$
- $\mu(\bigcup_{i \in I} A_i) = \sum_{i \in I} \mu(A_i)$ whenever $\{A_i\}_{i \in I}$ is a countable collection of disjoint measurable sets.

The collection $\mathcal{D}(X)$ of probability distributions over a measurable space X forms the so-called Giry monad. The monad unit $\eta\colon X \to \mathcal{D}(X)$ maps $a \in X$ to the point mass (or Dirac measure) δ_a on a. The monad multiplication $m\colon \mathcal{D}\mathcal{D}(X) \to \mathcal{D}(X)$ is given by integration:

$$m(P)(S) = \int ev_S dP, \text{ where } ev_S(\mu) = \mu(S)$$

Given measurable spaces (X, Σ_X) and (Y, Σ_Y), a *Markov kernel* is a function $P : X \times \Sigma_Y \to [0, 1]$ (equivalently, $X \to \Sigma_Y \to [0, 1]$) that maps each source state $x \in X$ to a distribution over target states $P(x, -) : \Sigma_Y \to [0, 1]$.

Markov kernels form the arrows in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$ of the $\mathcal{D}$ monad; we denote such arrows by $\begin{tikzcd} X \arrow[r, "P", "\circ" marking] & Y \end{tikzcd}$. Composition in the Kleisli category is given by integration:

$$\begin{tikzcd} X \arrow[r, "P", "\circ" marking] & Y \arrow[r, "Q", "\circ" marking] & Z \end{tikzcd} \qquad (P \circ Q)(x, A) = \int_{y \in Y} P(x, dy) \cdot Q(y, A)$$

Associativity of composition is essentially Fubini's theorem.

3 Defining Almost Sure Productivity

We will consider programs that denote probability distributions over coinductive types, such as infinite streams or trees. In this section, we focus on the definitions for programs producing streams and binary trees for simplicity, but all the results below can be generalized to arbitrary polynomial functors (see Section 7).

First, we introduce the semantics of programs. Rather than fix a concrete programming language at this point, we let $\mathbb{T}$ denote the space of programs and suppose we have a discrete global clock. At each time step, with some probability, we will either observe a concrete output $a\colon A$ or nothing $\bot$. Intuitively, a program $p \in \mathbb{T}$ is ASP if its probability of producing unboundedly many outputs is 1. Formally, we will endow programs in $\mathbb{T}$ with a denotational semantics $[\![-]\!]\colon \mathbb{T} \to \mathcal{D}((A_\bot)^\omega)$. We will define this *global* semantics coinductively, starting from a given one-step semantics function that maps each term to an output in $A_\bot$ and the resulting term. The subtlety in the definition is that since the step function will be probabilistic, we need to work in the Kleisli category for the distribution monad; final coalgebras in this category are more difficult to compute. We take the work on probabilistic streams by Kerstan and König [23] as our starting point, and then generalize to probabilistic trees.

▶ **Theorem 1** (Finality for streams [23]). *Given a set of programs $\mathbb{T}$ endowed with a probabilistic step function $\mathsf{st}\colon \mathbb{T} \to \mathcal{D}(A_\bot \times \mathbb{T})$, there is a unique semantics function $[\![-]\!]$ assigning to each program a probability distribution of output streams such that the following diagram commutes in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$.*

$$\begin{tikzcd}
\mathbb{T} \arrow[r, "{[\![-]\!]}"] \arrow[d, "\mathsf{st}"'] & (A_\bot)^\omega \arrow[d, "<\mathsf{head},\mathsf{tail}>"] \\
A_\bot \times \mathbb{T} \arrow[r, "id\times[\![-]\!]"] & A_\bot \times (A_\bot)^\omega
\end{tikzcd}$$

▶ **Definition 2** (ASP for streams). A stream program $p \in \mathbb{T}$ is *almost surely productive* (ASP) if

$$\Pr_{\sigma\sim[\![p]\!]}[\sigma \text{ has finitely many concrete output elements } a \in A] = 0.$$

For this to be a sensible definition, the event in the probability must be a measurable set in some σ-algebra on $(A_\bot)^\omega$. Following Kerstan and König, we take the σ-algebra generated by *cones*, sets of the form $uA^\omega = \{v \in (A_\bot)^\omega \mid u \text{ prefix of } v, u \in (A_\bot)^*\}$. We stress that our definition of ASP is independent of the way we defined $[\![-]\!]\colon \mathbb{T} \to \mathcal{D}(A_\bot)^\omega$. Working with a coinductive definition principle will be useful later for showing soundness when verifying ASP, but our definition is sensible for any semantic function $[\![-]\!]$.

▶ **Example 3.** As an example let us consider the following program defining a stream σ recursively, in which each recursion step is determined by a coin flip with bias p. If the coin flip results in heads we add an element a, otherwise we don't output and we compute the tail.

$$\sigma = (a : \sigma) \oplus_p \mathsf{tail}(\sigma)$$

The above program will output infinite many a's when $p > 1/2$ (that is, the probability of heads is greater than tails). Hence, the above program is ASP for $p > 1/2$. In the sequel we will show two methods to prove this fact.

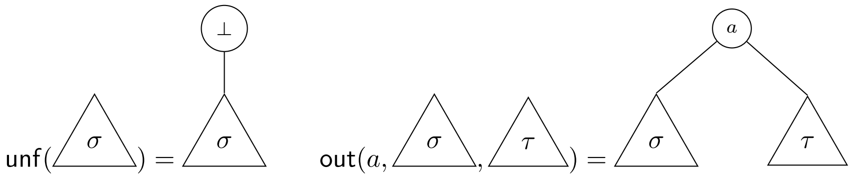
For the purpose of generalizing the above to other datatypes and also for developing methods to check ASP, it will be convenient to use a different representation for the functor $F(X) = A_\perp \times X \cong A \times X + X$. In the rest of this paper we will often use the latter representation and refer to the final coalgebra as *observation streams* $\mathsf{OS} = (A_\perp)^\omega$ with structure $\mathsf{OS} \xleftarrow[\cong]{<\mathsf{out},\mathsf{unf}>} A \times \mathsf{OS} + \mathsf{OS}$ given by $\mathsf{out}(a, \sigma) = a : \sigma$ and $\mathsf{unf}(\sigma) = \perp : \sigma$.

Now, we can generalize to infinite trees by using the functor $F(X) = A \times X \times X + X$.

▶ **Theorem 4** (Finality for trees). *Given a set of programs* $\mathbb{T}$ *endowed with a probabilistic step function* $\mathsf{st}\colon \mathbb{T} \to \mathcal{D}(A \times \mathbb{T} \times \mathbb{T} + \mathbb{T})$, *there is a* unique *semantics function* $[\![-]\!]$ *assigning to each program a probability distribution of output trees such that the following diagram commutes in the Kleisli category* $\mathcal{K}\ell(\mathcal{D})$.

$$\begin{tikzcd}
\mathbb{T} \arrow[r, "{[\![-]\!]}"] \arrow[d, "\mathsf{st}"'] & \mathsf{Trees}(A_\perp) \arrow[d, "{<\mathsf{out},\mathsf{unf}>^{-1}}"] \\
A \times \mathbb{T} \times \mathbb{T} + \mathbb{T} \arrow[r, "{id \times [\![-]\!] \times [\![-]\!] + [\![-]\!]}"] & A \times \mathsf{Trees}(A_\perp) \times \mathsf{Trees}(A_\perp) + \mathsf{Trees}(A_\perp)
\end{tikzcd}$$

$\mathsf{Trees}(A_\perp)$ are infinite trees where the nodes are either elements of A or $\perp$. An a-node has two children whereas a $\perp$-node only has one child. Formally, we can construct these trees with the two maps out and unf:



Defining ASP for trees is a bit more subtle than for streams. Due to measurability issues, our definition will only take the probability along one path at a time in the tree. A bit more formally, let $w \in \{L, R\}^\omega$ be an infinite word on alphabet $\{L, R\}$. Given any tree $t \in \mathsf{Trees}(A_\perp)$, w induces a single path t_w in the tree—starting from the root, the path follows the left or right child of a-nodes as indicated by w, and the single child of $\perp$-nodes. Then, we can define the following notion of ASP.

▶ **Definition 5** (ASP for trees). A tree program $p\colon \mathbb{T}$ is *almost surely productive* (ASP) if

$$\forall w \in \{L, R\}^\omega.\ \Pr_{t \sim [\![p]\!]}[t_w \text{ has infinitely many concrete output nodes } a \in A] = 1.$$

We have omitted the σ-algebra structure on $\mathsf{Trees}(A_\perp)$ for lack of space, but it is quite similar to the one for streams: it is generated by the cones $u\mathsf{Trees}(A_\perp) = \{t \in \mathsf{Trees}(A_\perp) \mid t \text{ is an extension of the finite tree } u\}$. For every $w \in \{L, R\}^\omega$, the above event is measurable in this σ-algebra.

▶ **Example 6.** Consider the probabilistic tree defined by the following equation:

$$\tau = \mathsf{mk}(a, \tau, \tau) \oplus_p \mathsf{left}(\tau)$$

The $\mathsf{mk}(a, t_1, t_2)$ constructor produces a tree with the root labeled by a and children t_1 and t_2, while the $\mathsf{left}(t)$ destructor consumes the output at the root of t and steps to the left child of t. While this example is more difficult to work out informally, it has similar ASP behavior as the previous example we saw for streams. When $p > 1/2$ this program is ASP, since it has strictly higher probability of constructing a node (and producing an output) than destructing a node (and consuming an output). In the following two sections, we will present two methods to verify this property.

4 A Calculus for Probabilistic Streams and Trees

Now that we have introduced almost sure productivity, we consider how to verify this property. We work with a simple calculus for probabilistic coinductive programming, separated into languages for streams and for trees. We suppose that outputs are drawn from some finite alphabet A. The language for streams considers terms of the following form:

$$e \in \mathbb{T} ::= \sigma \mid e \oplus_p e \mid u : e \ (u \in \mathbb{P}) \mid \mathsf{tail}(e)$$

The distinguished variable σ represents a recursive occurrence of the stream so that streams can be defined via equations $\sigma = e$. The operation $e_1 \oplus_p e_2$ selects e_1 with probability p and e_2 with probability $1 - p$. We consider a set $\mathbb{P}$ of primitive streams, represented by sequences in A^ω. The constructor $u : e$ builds a stream with head given by the first element in u and tail e. The first element of u is consumed, so that further occurrences of u in e draw from the tail of u. The destructor $\mathsf{tail}(e)$ computes the tail of a stream.

The language for trees is similar, with terms of the following form:

$$e \in \mathbb{T} ::= \tau \mid e \oplus_p e \mid \mathsf{mk}(u, e, e) \ \ (u \in \mathbb{P}) \mid \mathsf{left}(e) \mid \mathsf{right}(e)$$

The distinguished variable τ represents a recursive occurrence of the tree, so that trees can be defined as $\tau = e$. The set $\mathbb{P}$ of primitive trees now contains infinite binary trees labeled by A; the constructor $\mathsf{mk}(u, e_1, e_2)$ builds a tree with root labeled by the root of u and children e_1 and e_2, where e_1 and e_2 draw from the left and right children of u. The destructors $\mathsf{left}(e)$ and $\mathsf{right}(e)$ extract the left and right children of e, respectively.

We interpret these terms coalgebraically by first giving a step function from $\mathsf{st}_e : \mathbb{T} \to \mathcal{D}(F(\mathbb{T}))$ for an appropriate functor, and then taking the semantics as the map to the final coalgebra as defined in the previous section. In the case of streams, we take the functor $F(X) = A \times X + X$: a term steps to a distribution over either an output in A and a resulting term, or just a resulting term (with no output). To describe how the recursive occurrence σ steps, we parametrize the step function st_e by the top level stream term e; this term remains fixed throughout the evaluation. We also assume each primitive stream term u is associated with an output $a_u \in A$ and primitive stream $u' \in \mathbb{P}$, representing the rest of the stream.

In the step relation, probabilistic choice terms are reduced by scaling the result of stepping e and the result of stepping e' by p and $1 - p$ respectively, and then combining the distributions:

$$\mathsf{st}_e(e_1 \oplus_p e_2) \triangleq p \cdot \mathsf{st}_e(e_1) + (1 - p) \cdot \mathsf{st}_e(e_2)$$

The next cases push destructors into terms:

$$\begin{aligned} \mathsf{st}_e(\mathsf{tail}^k(u : e)) &\triangleq \mathsf{st}_e(\mathsf{tail}^{k-1}(e[u'/u])) \\ \mathsf{st}_e(\mathsf{tail}^k(e_1 \oplus_p e_2)) &\triangleq \mathsf{st}_e(\mathsf{tail}^k(e_1) \oplus_p \mathsf{tail}^k(e_2)) \end{aligned}$$

Observe that after the output in the constructor is consumed in the first case, occurences of u in the tail term e are replaced by a new primitive stream u'. The remaining cases return a point distribution. If we have reached a constructor then we produce a single output. Otherwise, we replace σ by the top level stream term, unfolding a recursive occurrence.

$$\begin{aligned} \mathsf{st}_e(u : e') &\triangleq \delta(\mathit{inl}(a_u, e'[u'/u])) \\ \mathsf{st}_e(e') &\triangleq \delta(\mathit{inr}(e'[e/\sigma])) \quad \text{otherwise} \quad (e' = \sigma, e' = \mathsf{tail}^k(\sigma)) \end{aligned}$$

Note that a stream may produce multiple outputs in a single step but only first output is recorded during the step. However, the remaining outputs are preserved and recorded in later steps.

The semantics is similar for trees. We take the functor $F(X) = (A \times X \times X) + X$: a term reduces to a distribution over either an output in A and two child terms, or a resulting term and no output. We also assume each primitive tree term u is associated with an output $a_u \in A$ and two primitive terms u_l, u_r.

The main changes are in the constructors and destructors. The constructor $\mathsf{mk}(u, e_1, e_2)$ reduces to $\delta(\mathit{inl}(a_u, e_1[u_l/u], e_2[u_r/u]))$, representing an output a_u this step. Destructors are handled similar to tail for streams, where $\mathsf{left}(\mathsf{mk}(u, e_1, e_2))$ reduces to $e_1[u_l/u]$ and $\mathsf{right}(\mathsf{mk}(u, e_1, e_2))$ reduces to $e_2[u_r/u]$, and $\mathsf{tail}^k(-)$ is generalized to any finite combination of $\mathsf{left}(-)$ and $\mathsf{right}(-)$.

Concretely, let $C[e']$ be any (possibly empty) combination of left and right applied to e'. We have the following step rules:

$$\begin{aligned} \mathsf{st}_e(C[\mathsf{left}(\mathsf{mk}(u, e_l, e_r))]) &\triangleq \mathsf{st}_e(C[e_l]) \\ \mathsf{st}_e(C[\mathsf{right}(\mathsf{mk}(u, e_l, e_r))]) &\triangleq \mathsf{st}_e(C[e_r]) \\ \mathsf{st}_e(C[e_1 \oplus_p e_2]) &\triangleq p \cdot \mathsf{st}_e(C[e_1]) + (1-p) \cdot \mathsf{st}_e(C[e_2]) \\ \mathsf{st}_e(\mathsf{mk}(u, e_l, e_r)) &\triangleq \delta(\mathit{inl}(a_u, e_l[u_l/u], e_r[u_r/u])) \\ \mathsf{st}_e(C[\tau]) &\triangleq \delta(\mathit{inr}(C[e])) \end{aligned}$$

5 Syntactic Conditions for ASP

With the language and semantics in hand, we now turn to proving ASP. While it is theoretically possible to reason directly on the semantics using our definitions from Section 3, in practice it is much easier to work with the language. In this section we present a syntactic sufficient condition for ASP. Intuitively, the idea is to approximate the expected number of outputs every step. If this measure is strictly positive, then the program is ASP.

5.1 A Syntactic Measure

We define a syntactic measure $\#(-) : \mathbb{T} \to \mathbb{R}$ by induction on stream terms:

$$\begin{aligned} \#(\sigma) &\triangleq 0 \\ \#(e_1 \oplus_p e_2) &\triangleq p \cdot \#(e_1) + (1-p) \cdot \#(e_2) \\ \#(u : e) &\triangleq \#(e) + 1 \\ \#(\mathsf{tail}(e)) &\triangleq \#(e) - 1 \end{aligned}$$

The measure $\#$ describes the expected difference between the number of outputs produced (by constructors) and the number of outputs consumed (by destructors) in each unfolding of the term. Likewise, we can define a measure for tree terms.

$$\begin{aligned}
\#(\tau) &\triangleq 0 \\
\#(e_1 \oplus_p e_2) &\triangleq p \cdot \#(e_1) + (1-p) \cdot \#(e_2) \\
\#(\mathsf{mk}(u, e_1, e_2)) &\triangleq \min(\#(e_1), \#(e_2)) + 1 \\
\#(\mathsf{left}(e)) = \#(\mathsf{right}(e)) &\triangleq \#(e) - 1
\end{aligned}$$

We can now state conditions for ASP for streams.

▶ **Theorem 7.** *Let e be a stream term with $\gamma = \#(e)$. If $\gamma > 0$, e is ASP.*

Note that we cannot conclude anything positive or negative when $\gamma = 0$. Likewise, we can give similar sufficient conditions for tree terms.

▶ **Theorem 8.** *Let e be a tree term with $\gamma = \#(e)$. If $\gamma > 0$, e is ASP.*

5.2 Soundness

The main idea behind the proof for streams is that by construction of the step relation, each step either produces an output or unfolds a fixed point (if there is no output). In unfolding steps, the expected measure of the term plus the number of outputs increases by γ. By defining an appropriate martingale and applying the Azuma-Hoeffding inequality, the sum of the measure and the number of outputs must increase linearly as the term steps when $\gamma > 0$. Since the measure is bounded above—when the measure is large the stream outputs instead of unfolding—the number of outputs must increase linearly and the stream is ASP.

We will need a few standard constructions and results from probability theory.

▶ **Definition 9** (See, e.g., [14]). A *filtration* $\{\mathcal{F}_i\}_{i\in\mathbb{N}}$ of a σ-algebra $\mathcal{F}$ on a measurable space A is an sequence of σ-algebras such that $\mathcal{F}_i \subseteq \mathcal{F}_{i+1}$ and $\mathcal{F}_i \subseteq \mathcal{F}$, for all $i \in \mathbb{N}$. A *stochastic process* is a sequence of random variables $\{X_i : A \to B\}_{i\in\mathbb{N}}$ for B some measurable space, and the process is *adapted to the filtration* if every X_i is $\mathcal{F}_i$-measurable.

Intuitively, a filtration gives each event a time i at which the event starts to have a well-defined probability. A stochastic process is adapted to the filtration if its value at time i only depends on events that are well-defined at time i or before (and not events at future times).

An important class of stochastic processes are martingales.

▶ **Definition 10** (See, e.g., [14]). Let $\{X_i : A \to \mathbb{R}\}$ be a real-valued stochastic process adapted to some filtration on A, and let μ be a measure on A. Suppose that $\mathbb{E}_\mu[X_i] < \infty$ for all X_i. The sequence is a *martingale* if for all $i \in \mathbb{N}$, we have

$$\mathbb{E}_\mu[X_{i+1} \mid \mathcal{F}_i] = X_i.$$

The conditional expectation turns X_{i+1} from an $\mathcal{F}_{i+1}$-measurable map to an $\mathcal{F}_i$-measurable map; equivalently, the martingale condition can be stated as

$$\mathbb{E}_\mu[(X_{i+1} - X_i)\chi_F] = 0,$$

for every event $F \in \mathcal{F}$, where χ_F is the indicator function. If the equalities are replaced by $\geq$ (resp., $\leq$), then the sequence is a sub- (resp., super-) martingale.

Martingale processes determine a sequence of random variables that may not be independent, but where the expected value of the process at some time step depends only on its value at the previous time step. Martingales satisfy concentration inequalities.

▶ **Theorem 11** (Azuma-Hoeffding inequality [5]). *Let $\{X_i\}_i$ be sequence such that $|X_{i+1} - X_i| \leq c$ for all $i \in \mathbb{N}$. If $\{X_i\}_i$ is a sub-martingale, then for every $n \in \mathbb{N}$ and $B \geq 0$, we have*

$$\Pr[X_n - X_0 \leq -B] \leq \exp(-B^2/2nc).$$

If $\{X_i\}_i$ is a super-martingale, then for every $n \in \mathbb{N}$ and $B \geq 0$, we have

$$\Pr[X_n - X_0 \geq B] \leq \exp(-B^2/2nc).$$

If $\{X_i\}_i$ is both a martingale, then combining the above results gives

$$\Pr[|X_n - X_0| \geq B] \leq 2\exp(-B^2/2nc).$$

Proof of Theorem 7. While the semantics constructed in Section 3 is sufficient to describe ASP, for showing soundness it is more convenient to work with an instrumented semantics that tracks the term in the observation stream. We can give a step function of type $\mathsf{st}'_e : \mathbb{T} \to \mathcal{D}(F'(\mathbb{T}))$, where $F'(X) = (A \times X + X) \times \mathbb{T}$ by recording the input term in the output. For instance:

$$\begin{aligned} \mathsf{st}'_e(u : e') &\triangleq \delta(\mathit{inl}(a_u, e'[u'/u]), u : e') \\ \mathsf{st}'_e(\sigma) &\triangleq \delta(\mathit{inr}(e), \sigma) \end{aligned}$$

and so forth. Using essentially the same construction as in Section 3, we get an instrumented semantics $[\![-]\!]' : \mathbb{T} \to \mathcal{D}(\mathsf{OS}')$, where OS' are infinite streams with constructors $\mathsf{out}' : (A \times \mathsf{OS}') \times \mathbb{T} \to \mathsf{OS}'$ and $\mathsf{unf}' : \mathsf{OS}' \times \mathbb{T} \to \mathsf{OS}'$, representing output and unfold steps respectively. Letting the map $u : \mathsf{OS}' \to \mathsf{OS}$ simply drop the instrumented terms, the map $\mathcal{D}(u) \circ [\![-]\!]' : \mathbb{T} \to \mathcal{D}(\mathsf{OS})$ coincides with the semantics $[\![-]\!]$ defined in Section 3 by finality.

Now, we define a few stochastic processes. Let $\{T_i : \mathsf{OS}' \to \mathbb{T}\}_i$ be the sequence of instrumented terms with $T_0 = e$, $\{O_i : \mathsf{OS}' \to \{0,1\}\}_i$ be 1 if the ith node is an output node and 0 if not, $\{U_i : \mathsf{OS}' \to \{0,1\}\}_i = \{1 - O_i\}_i$. It is straightforward to show that T_i is $\mathcal{F}_{i-1}$-measurable (and hence $\mathcal{F}_i$-measurable), and O_i, U_i are $\mathcal{F}_i$-measurable—all three processes are defined by the events in the first i steps.

Now for any stream term $t \in \mathbb{T}$, we claim that

$$\mathbb{E}_{\mathsf{st}'_e(t)}[(\mathit{inl}(-, t'), -) \to 1 + \#(t') \text{ else } (\mathit{inr}(t'), -) \to \#(t') - \gamma] = \#(t).$$

This follows by induction on terms using the definition of st'_e. We can lift the equality to the semantics, giving

$$\mathbb{E}_{[\![T_0]\!]'}[O_{i+1} - \gamma U_{i+1} + \#(T_{i+2}) \mid \mathcal{F}_i] = \#(T_{i+1})$$

noting that T_{i+1} is $\mathcal{F}_i$-measurable and recalling that $T_0 = e$ is the initial term. We now define another stochastic process via

$$X_i \triangleq \sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1}).$$

Note that X_i is $\mathcal{F}_i$-measurable. As we will show, this process tends towards zero, the second term decreases, and the third term remains bounded. Hence, the first term—the cumulative number of outputs—must tend towards infinity. Evidently each X_i is bounded and we are working with probability measures, so each X_i is integrable. We can directly check that $\{X_i\}_i$ is a martingale:

$$\begin{aligned}\mathbb{E}_{[\![T_0]\!]'}[X_{i+1} \mid \mathcal{F}_i] &= \mathbb{E}_{[\![T_0]\!]'}\left[\sum_{j=0}^{i} O_j + O_{i+1} - \gamma \sum_{j=0}^{i} U_j - \gamma U_{i+1} + \#(T_{i+2}) \mid \mathcal{F}_i\right] \\ &= \mathbb{E}_{[\![T_0]\!]'}\left[\sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1}) \mid \mathcal{F}_i\right] \\ &= \sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1}) = X_i.\end{aligned}$$

We now claim that $\#(T_i) \leq c'$ where c' is one more than the number of constructors in the original term T_0. This follows by observing that (i) the step function increases the measure by at most the number of constructors or 1 (if the step function unfolds a primitive term) every unfolding step, and (ii) the step function only unfolds if a term reduces to a term with non-positive measure. Similarly,

$$\sum_{j=0}^{i} U_j \geq \lfloor i/c' \rfloor$$

since each unfolding step leads to at most c' output (non-unfolding) steps.

Since O_i and U_i are both in $\{0, 1\}$, this implies that $|X_{i+1} - X_i|$ is bounded by some constant $c = c' + 2$, depending only on the initial term. We can now apply the Azuma-Hoeffding inequality (Theorem 11). For every $n \in \mathbb{N}$ and $B \geq 0$, we have

$$\Pr_{[\![T_0]\!]'}[X_n - X_0 \geq -B] \geq 1 - \exp(-B^2/2nc).$$

Taking $B = n^{2/3}$, we have

$$\Pr_{[\![T_0]\!]'}[X_n \geq X_0 - n^{2/3}] \geq 1 - \exp(1/2n^{1/3}c).$$

We also know that the total number of outputs is at least

$$\sum_{j=0}^{n} O_j = X_n + \gamma \sum_{j=0}^{n} U_j - \#(T_{n+1}) \geq X_n + \gamma \lfloor n/c' \rfloor.$$

So if $\gamma > 0$, the stream has zero probability of producing at most M outputs for any finite M. This is because for X_n is at least $-n^{2/3}$ with probability arbitrarily close to 1 (for large enough n), and $\gamma \lfloor n/c' \rfloor$ grows linearly in n for γ positive. Hence, the term is ASP. ◀

The proof for trees is similar, showing that on any path through the observation tree there are infinitely many output steps with probability 1.

Proof of Theorem 8. We again work with an instrumented semantics based on the step function $\mathsf{st}'_e : \mathbb{T} \to \mathcal{D}(F'(\mathbb{T}))$, where $F'(X) = (A \times X \times X + X) \times \mathbb{T}$ by recording the input term in the output. For instance:

$$\begin{aligned}\mathsf{st}'_e(\mathsf{mk}(u, e_1, e_2)) &\triangleq \delta(\mathit{inl}(a_u, e_1[u_l/u], e_2[u_r/u]), \mathsf{mk}(u, e_1, e_2)) \\ \mathsf{st}'_e(\tau) &\triangleq \delta(\mathit{inr}(e), \tau)\end{aligned}$$

and so forth. Using essentially the same construction as in Section 3, we get an instrumented semantics $[\![-]\!]' : \mathbb{T} \to \mathcal{D}(\mathsf{OT}')$, where OT' are infinite trees with constructors $\mathsf{out}' : (A \times \mathsf{OT}' \times \mathsf{OT}') \times \mathbb{T} \to \mathsf{OT}'$ and $\mathsf{unf}' : \mathsf{OT}' \times \mathbb{T} \to \mathsf{OT}'$, representing output and unfold steps respectively. Letting the map $u : \mathsf{OT}' \to \mathsf{OT}$ simply drop the instrumented terms, the map $\mathcal{D}(u) \circ [\![-]\!]' : \mathbb{T} \to \mathcal{D}(\mathsf{OT})$ coincides with the semantics $[\![-]\!]$ defined in Section 3 by finality.

Let $w \in \{L, R\}^\omega$ be any infinite word, describing whether to follow the left or right child of a tree. Each word determines a path through an observation tree: on unfold nodes we simply follow the child, while on output nodes we follow the child indicated by w. We aim to show that if $\gamma > 0$, then there are infinitely many output nodes along this path with probability 1. If this holds for all w, then the tree term must be ASP.

To model the path, we define a sequence $\{P_i : \mathsf{OT}' \to A \times \mathbb{T} \times \mathbb{T} + \mathbb{T}\}_i$ inductively. P_0 is simply the root of the output tree OT'. Given $P_0, \dots, P_i$, we define P_{i+1} to be a child of P_i as follows. If P_i is an unfold node it only has one child, so we take P_{i+1} to be this child. Otherwise we take P_{i+1} to be the child of P_i indicated by w_{j+1}, where j is the number of output nodes in $P_0, \dots, P_i$. The process $\{P_i\}_i$ is adapted to the filtration on OT'. (Note that all indices start at 0.)

Now, we can define similar processes as in the stream case with respect to the path. Let $\{T_i : \mathsf{OS}' \to \mathbb{T}\}_i$ be the sequence of instrumented terms along the path with $T_0 = e$, $\{O_i : \mathsf{OS}' \to \{0, 1\}\}_i$ be 1 if P_i is an output node and 0 if not, $\{U_i : \mathsf{OS}' \to \{0, 1\}\}_i = \{1 - O_i\}_i$. It is straightforward to show that T_i is $\mathcal{F}_{i-1}$-measurable (and hence $\mathcal{F}_i$-measurable), and O_i, U_i are $\mathcal{F}_i$-measurable—all three processes are defined by the events in the first i steps.

Now for any tree term $t \in \mathbb{T}$, we have

$$\mathbb{E}_{\mathsf{st}'_e(t)}[(\mathit{inl}(-, t'), -) \to 1 + \#(t') \text{ else } (\mathit{inr}(t'), -) \to \#(t') - \gamma] \geq \#(t)$$

by induction on terms using the definition of st'_e. The inequality arises from applying a destructor to a constructor—we may end up with a child term that has larger measure than the parent term, since the measure of a constructor takes the smaller measure of its children. We can lift the inequality to the semantics, giving

$$\mathbb{E}_{[\![T_0]\!]'}[O_{i+1} - \gamma U_{i+1} + \#(T_{i+2}) \mid \mathcal{F}_i] \geq \#(T_{i+1})$$

noting that T_{i+1} is $\mathcal{F}_i$-measurable and letting $T_0 = e$ be the initial term. We can now our invariant process

$$X_i \triangleq \sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1})$$

which is a sub-martingale:

$$\begin{aligned} \mathbb{E}_{[\![T_0]\!]'}[X_{i+1} \mid \mathcal{F}_i] &= \mathbb{E}_{[\![T_0]\!]'}\left[\sum_{j=0}^{i} O_j + O_{i+1} - \gamma \sum_{j=0}^{i} U_j - \gamma U_{i+1} + \#(T_{i+2}) \mid \mathcal{F}_i\right] \\ &\geq \mathbb{E}_{[\![T_0]\!]'}\left[\sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1}) \mid \mathcal{F}_i\right] \\ &= \sum_{j=0}^{i} O_j - \gamma \sum_{j=0}^{i} U_j + \#(T_{i+1}) = X_i. \end{aligned}$$

The remainder of the proof is now quite similar to the stream case. $\#(T_i) \leq c'$ where c' is one more than the number of constructors in the original term T_0. This follows by observing that (i) the step function increases the measure by at most the number of constructors or 1 (if the step function unfolds a primitive term) every unfolding step, and (ii) the step function only unfolds if a term reduces to a term with non-positive measure. Similarly,

$$\sum_{j=0}^{i} U_j \geq \lfloor i/c' \rfloor$$

since each unfolding step leads to at most c' output (non-unfolding) steps.

Since O_i and U_i are both in $\{0, 1\}$, this implies that $|X_{i+1} - X_i|$ is bounded by some constant $c = c' + 2$, depending only on the initial term. We can now apply the Azuma-Hoeffding inequality (Theorem 11). For every $n \in \mathbb{N}$ and $B \geq 0$, we have

$$\Pr_{[\![T_0]\!]'}[X_n - X_0 \geq -B] \geq 1 - \exp(-B^2/2nc).$$

Taking $B = n^{2/3}$, we have

$$\Pr_{[\![T_0]\!]'}[X_n \geq X_0 - n^{2/3}] \geq 1 - \exp(1/2n^{1/3}c).$$

We also know that the total number of outputs along the path w is at least

$$\sum_{j=0}^{n} O_j = X_n + \gamma \sum_{j=0}^{n} U_j - \#(T_{n+1}) \geq X_n + \gamma \lfloor n/c' \rfloor.$$

So if $\gamma > 0$, the stream has zero probability of producing at most M outputs along w for any finite M. This is because for X_n is at least $-n^{2/3}$ with probability arbitrarily close to 1 (for large enough n), and $\gamma \lfloor n/c' \rfloor$ is growing linearly in n for γ positive. Since the tree term produces at least M outputs along path w with probability 1 for every M and every w, it is ASP. ◀

5.3 Examples

We consider a few examples of our analysis. Let the alphabet $A = \mathbb{N}$.

▶ **Example 12.** Consider the stream definition $\sigma = (\mathit{nats} : \sigma) \oplus_p \mathsf{tail}(\sigma)$, where the primitive term nats represents the stream of natural numbers $0, 1, \ldots$. Each element in nats is produced with probability p and dropped with probability $1 - p$. The # measure of the stream term is $p \cdot 1 + (1 - p) \cdot (-1) = 2p - 1$. By Theorem 7, the stream is ASP when $p > 1/2$.

The measure does not give useful information when # is not positive.

▶ **Example 13.** Consider the stream definition $\sigma = (\mathit{nats} : \sigma) \oplus_{1/2} \mathsf{tail}(\sigma)$; the # measure of the term is 0. The number of outputs can be modeled by a simple random walk on a line, where the maximum position is the number of outputs produced by the stream. Since a simple random walk has probability 1 of reaching every $n \in \mathbb{N}$ [30], the stream term is ASP.

In contrast, the term $\#(\sigma) = 0$ but the stream definition $\sigma = \sigma$ is clearly non-productive.

We can give similar examples for tree terms.

▶ **Example 14.** Let ones be a primitive tree term that always produces the output 1. Consider the tree definitions $\tau = e_i$, where

- $e_1 \triangleq \mathsf{left}(\tau) \oplus_{1/4} \mathsf{mk}(\mathit{ones}, \tau, \tau)$
- $e_2 \triangleq \mathsf{left}(\tau) \oplus_{1/4} \mathsf{mk}(\mathit{ones}, \tau, \mathsf{left}(\tau))$.

We apply Theorem 8 to deduce ASP. We have $\#(e_1) = (1/4) \cdot (-1) + (3/4) \cdot (+1) = 1/2$, so the first term is ASP. For the second term, $\#(e_2) = (1/4) \cdot (-1) + (3/4) \cdot 0 = -1/4$, so our analysis does not give any information.

6 Probabilistic Model-Checking for ASP

The syntactic analysis for ASP is simple, but it is not complete—no information is given if $\#(e) \leq 0$. In this section we give a more sophisticated, complete analysis by first modeling the operational semantics of a term by a Probabilistic Pushdown Automaton (pPDA), then deciding ASP by reduction to model-checking.

6.1 Probabilistic Pushdown Automata and pCTL*

A pPDA is a tuple $\mathcal{A} = (S, \Gamma, \mathcal{T})$ where S is a finite set of states and Γ is a finite stack alphabet. The transition function $\mathcal{T} : S \times (\Gamma \cup \{\bot\}) \times S \times \Gamma^* \to [0, 1]$ in addition to jumping between states, looks on each step at the symbol on top of the stack (which might be empty, denoted $\bot$), consumes it, and pushes a (possibly empty, denoted ε) string of symbols onto the stack. A configuration of $\mathcal{A}$ is an element of $\mathcal{C} = S \times \Gamma^*$, and represents the state of the pPDA and the contents of its stack (with the top on the left) at some point of its execution. Given a configuration, the transition function $\mathcal{T}$ specifies a distribution over configurations in the next step. Given an initial state s and an initial stack $\gamma \in \Gamma^*$, $\mathcal{T}$ induces a distribution $\mathrm{Paths}(s, \gamma)$ over the infinite sequence of configurations starting in (s, γ).

Probabilistic Computation Tree Logic (pCTL*) [19] is a branching-time temporal logic that describes states of a probabilistic transition system, which in a pPDA actually correspond to its configurations. Propositions in pCTL* are defined by the syntax

$$\Phi, \Psi ::= \top \mid Q \mid \Phi \wedge \Psi \mid \neg\Phi \mid \mathcal{P}(\phi) \bowtie p \qquad\qquad \phi, \psi ::= \Phi \mid \neg\phi \mid \mathcal{X}\phi \mid \phi\,\mathcal{U}\psi \mid \Diamond\phi \mid \Box\phi$$

where Φ, Ψ are *state formulas*, which describe the paths starting on a given state, and ϕ, ψ are *path formulas*, which describe a particular path, Q is a set of atomic propositions, $\bowtie$ ranges over predicates $\{=, \leq, \ldots\}$ and $p \in [0, 1]$. The *qualitative fragment* of pCTL* is the set of formulas where p is restricted to $\{0, 1\}$.

Given a pPDA $\mathcal{A}$, we define the semantics of a pCTL* formula as follows:

$$\begin{aligned}
(s,\gamma) &\models \top \\
(s,\gamma) &\models q \Leftrightarrow (s,\gamma) \in [\![q]\!] \quad (q \in Q) \\
(s,\gamma) &\models \Phi \wedge \Psi \Leftrightarrow (s,\gamma) \models \Phi \wedge (s,\gamma) \models \Psi \\
(s,\gamma) &\models \neg\Phi \Leftrightarrow (s,\gamma) \not\models \Phi \\
(s,\gamma) &\models \mathcal{P}(\phi) \bowtie p \Leftrightarrow \Pr_{\pi \sim \mathrm{Paths}(s,\gamma)}[\pi \models \phi] \bowtie p
\end{aligned}
\qquad
\begin{aligned}
\pi &\models \Phi \Leftrightarrow \pi[0] \models \Phi \\
\pi &\models \neg\phi \Leftrightarrow \pi \not\models \phi \\
\pi &\models \mathcal{X}\phi \Leftrightarrow \pi_1 \models \phi \\
\pi &\models \phi\,\mathcal{U}\psi \Leftrightarrow \exists i.\pi_i \models \psi \wedge \forall j < i.\pi_j \models \phi \\
\pi &\models \Diamond\phi \Leftrightarrow \exists i.\pi_i \models \phi \\
\pi &\models \Box\phi \Leftrightarrow \forall i.\pi_i \models \phi
\end{aligned}$$

where atomic propositions q are interpreted as $[\![q]\!] \subseteq \mathcal{C}$. As expected, state formulas are interpreted in pPDA configurations $(s, \gamma) \in \mathcal{C}$, while path formulas are interpreted in traces of configurations $\pi \in \mathcal{C}^\omega$; $\pi[i]$ is the ith element in the path π, and π_i is the suffix of π from $\pi[i]$. Path formulas describe measurable events in the σ-algebra of cones of $\mathcal{C}^\omega$, so the semantics is indeed well-defined [19].

Given a pPDA $\mathcal{A}$, a configuration $(s, \gamma) \in \mathcal{C}$ and a pCTL* formula Φ, the model-checking problem is to decide whether $(s, \gamma) \models \Phi$. The following is known.

▶ **Theorem 15** (Brázdil, Brozek and Forejt [7]). *The model-checking problem for pPDAs is decidable for the qualitative fragment of pCTL*.*[1]

[1] Formally, this is proven for atomic propositions that are sets of configurations whose stack can be recognized by a finite automaton. Since our propositions will be configurations with an empty stack, this is enough for us.

Almost sure productivity states that an event—producing an output—occurs infinitely often with probability 1. Such properties belong to the qualititative fragment of pCTL*.

▶ **Lemma 16.** *Let $(s, \gamma) \in \mathcal{C}$ be an initial configuration and $\mathcal{B} \subseteq \mathcal{C}$ be a set of configurations. Then $\Pr_{\pi \in \mathrm{Paths}(s,\gamma)}[\pi$ visits $\mathcal{B}$ infinitely often$] = 1$ iff $(s, \gamma) \models \mathcal{P}(\Box \Diamond \mathcal{B}) = 1$.*

Proof. See [6], Lemma 10.46. ◀

We will encode language terms as pPDAs and cast almost sure productivity as a qualitative pCTL* property stating that configurations representing output steps are reached infinitely often with probability 1. Theorem 15 then gives a decision procedure for ASP.

6.2 Modeling streams with pPDAs

The idea behind our encoding from terms to pPDAs is simple to describe. The states of the pPDA will represent subterms of the original term, and transitions will model steps. In the original step relation, the only way a subterm can step to a non-subterm is by accumulating destructors. We use a single-letter stack alphabet to track the number of destructors so that a term like $\mathsf{tail}^k(e)$ can be modeled by the state corresponding to e. More formally, given a stream term e we define a pPDA $\mathcal{A}_e = (\mathcal{S}_e, \{tl\}, \mathcal{T}_e)$, where $\mathcal{S}_e$ is the set of syntactic subterms of e and $\mathcal{T}_e$ is the following transition function:

$$\begin{aligned}
\mathcal{T}_e((\sigma, a), (e, a)) &= 1 & \mathcal{T}_e((u : e', \bot), (e', \varepsilon)) &= 1 \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_1, a)) &= p & \mathcal{T}_e((u : e', tl), (e', \varepsilon)) &= 1 \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_2, a)) &= 1 - p & \mathcal{T}_e((\mathsf{tail}(e'), a), (e', tl \cdot a) &= 1
\end{aligned}$$

where $\cdot$ concatenates strings, and we implicitly treat a as alphabet symbol or a singleton string. All non-specified transitions have zero probability. We define the set of *outputting configurations* as $\mathcal{O} \triangleq \{s \in \mathcal{C} \mid \exists u, e'. \, s = (u : e', \bot)\}$, that is, configurations where the current term is a constructor and there are no destructors left to apply. Our main result states that this set is visited infinitely often with probability 1 if and only if e is ASP. In fact, we prove something stronger:

▶ **Theorem 17.** *Let e be a stream term and let $\mathcal{A}_e$ be the corresponding pPDA. Then,*

$$\Pr_{t \sim [\![e]\!]}[t \textit{ has infinitely many output nodes}] = \Pr_{\pi \sim \mathrm{Paths}(e,\varepsilon)}[\pi \models \Box\Diamond\, \mathcal{O}].$$

In particular, e is ASP if and only if $(e, \varepsilon) \models \mathcal{P}(\Box \Diamond \mathcal{O}) = 1$.

Proof. The first part of the proof consists on simplifying the automaton $\mathcal{A}_e$ into a new automaton with a transition function $\mathcal{T}'$ that is synchronized to the step function for streams considered in Section 5, while preserving the validity of $\pi \models \Box\Diamond\, \mathcal{O}$ for every path π. The simplified transition function needs to skip over all states of the form $\mathsf{tail}(e')$ or $e_1 \oplus e_2$ until it reaches a state of the form $u : e'$ or σ. We proceed in two steps.

1. For every state of the form $\mathsf{tail}^{k+1}(e)$ such that e does not have a tl on top, we add a new transition from $\mathsf{tail}^{k+1}(e)$ to e so that $\mathcal{T}'((\mathsf{tail}^{k+1}(e), a), (e, tl^{k+1} \cdot a)) = 1$ and we remove the transition from $\mathsf{tail}^{k+1}(e)$ to $\mathsf{tail}^{k}(e)$. Then we remove unreachable states.
2. States of the form $e_1 \oplus_p e_2$ are removed, and for every transition from some e to $e_1 \oplus_p e_2$ such that $\mathcal{T}((e, a), (e_1 \oplus_p e_2, \gamma)) = q > 0$, we add new transitions from e to e_1 and e_2 so that $\mathcal{T}'((e, a), (e_1, \gamma)) = pq$ and $\mathcal{T}'((e, a), (e_2, \gamma)) = (1 - p)q$.

Notice that this step can be removed if we construct a reduced pPDA from the beginning, the choice of the given construction is motivated by clarity.

Now, the transition function induces a map $\overline{\mathcal{T}} : \mathcal{C} \to \mathcal{D}(A \times \mathcal{C} + \mathcal{C})$ from configurations to output distributions over configurations and outputs from one step of the pPDA:

$$\begin{aligned}
\overline{\mathcal{T}}(\sigma, \gamma) &\triangleq \delta(\mathit{inr}(e, \gamma)) \\
\overline{\mathcal{T}}(e_1 \oplus_p e_2, \gamma) &\triangleq p \cdot \overline{\mathcal{T}}(e_1, \gamma) + (1-p) \cdot \overline{\mathcal{T}}(e_2, \gamma) \\
\overline{\mathcal{T}}(u : e', \varepsilon) &\triangleq \delta(\mathit{inl}(u, (e', \varepsilon))) \\
\overline{\mathcal{T}}(u : e', \mathit{tl} \cdot \gamma) &\triangleq \overline{\mathcal{T}}(e', \gamma) \\
\overline{\mathcal{T}}(\mathsf{tail}(e'), \gamma) &\triangleq \overline{\mathcal{T}}(e', \mathit{tl} \cdot \gamma).
\end{aligned}$$

Hence, $(\mathcal{C}, \overline{\mathcal{T}})$ and $(\mathbb{T}, \mathsf{st})$ are coalgebras of the same functor. We can now build a map $f : \mathbb{T} \to \mathcal{C}$ from terms to configurations:

$$\begin{aligned}
f(\sigma) &\triangleq (\sigma, \varepsilon) \\
f(\mathsf{tail}^k(\sigma)) &\triangleq (\sigma, \mathit{tl}^k) \\
f(e_1 \oplus_p e_2) &\triangleq (e_1 \oplus_p e_2, \varepsilon) \\
f(\mathsf{tail}^k(e_1 \oplus_p e_2)) &\triangleq (e_1 \oplus_p e_2, \mathit{tl}^k) \\
f(\hat{u} : e') &\triangleq (\hat{u} : e', \varepsilon) \\
f(\mathsf{tail}^k(\hat{u} : e')) &\triangleq (\hat{u} : e', \mathit{tl}^k).
\end{aligned}$$

We assume that every primitive term has been replaced by the same constant stream $\hat{u}$, which changes expressivity but not productivity. For every term e, we have

$$\overline{\mathcal{T}}(f(e)) = \mathrm{case}(\mathsf{st}(e), \mathit{inl}(a, e_1) \mapsto \mathit{inl}(a, f(e_1)), \mathit{inr}(e_2) \mapsto \mathit{inr}(f(e_2)))$$

Therefore, f is a coalgebra homomorphism. By finality, this means that for all $e \in \mathbb{T}$, $[\![e]\!] = \mathrm{Paths}(f(e))$, and so we conclude

$$\Pr_{t \sim [\![e]\!]}[t \text{ has infinitely many output nodes}] = \Pr_{\pi \sim \mathrm{Paths}(f(e))}[\pi \models \Box\Diamond\, \mathcal{O}].$$

◀

By Theorem 15, ASP is decidable for stream terms. In fact, it is also possible to decide whether a stream term is almost surely *not* productive, i.e., the probability of producing infinitely many outputs is zero. Notice that since model-checking pCTL* for pPDAs is **2-EXPTIME**-hard [7], the syntactic criterion performs far better than the pPDA reduction whenever it can be applied.

6.3 Extending to trees

Now, we extend our approach to trees. The main difficulty is that since the pPDA can only simulate one path in the coinductive structure. The problem can be seen in the constructors. For streams, we can encode the term $u : e$ by proceeding to the tail e. For trees, however, how can we encode $\mathsf{mk}(a, e_1, e_2)$? The pPDA cannot simulate both e_1 and e_2. Since the failure of ASP may occur down either path, we cannot directly translate the ASP property on trees to pCTL*—ASP is a property of *all* paths. Instead, on constructors our pPDA encoding will choose a path at random to simulate. As we will show, if the probability of choosing a path that outputs infinitely often is 1, then every path will output infinitely often. Notice that in general, properties that happen with probability 1 do not necessarily happen for every path, but the structure of our problem allows us to make this generalization.

More formally, the stack alphabet will now be $\{rt, lt\}$, and on constructors we transition to each child with probability 1/2:

$$\begin{aligned}
\mathcal{T}_e((\tau, a), (e, a)) &= 1 \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_1, a)) &= p \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_2, a)) &= 1 - p \\
\mathcal{T}_e((\mathsf{mk}(u, e_l, e_r), \bot), (e_l, \varepsilon)) &= 1/2 \\
\mathcal{T}_e((\mathsf{mk}(u, e_l, e_r), \bot), (e_r, \varepsilon)) &= 1/2
\end{aligned}
\qquad
\begin{aligned}
\mathcal{T}_e((\mathsf{mk}(u, e_l, e_r), lt), (e_l, \varepsilon)) &= 1 \\
\mathcal{T}_e((\mathsf{mk}(u, e_l, e_r), rt), (e_r, \varepsilon)) &= 1 \\
\mathcal{T}_e((\mathsf{left}(e'), a), (e', lt \cdot a)) &= 1 \\
\mathcal{T}_e((\mathsf{right}(e'), a), (e', rt \cdot a)) &= 1
\end{aligned}$$

We define $\mathcal{O} \triangleq \{s \in \mathcal{C} \mid \exists u, e_l, e_r.\, s = (\mathsf{mk}(u, e_l, e_r), \varepsilon)\}$ to be the set of outputting configurations. We can characterize ASP with the following theorem:

▶ **Theorem 18.** *Let e be a tree term and $\mathcal{A}_e$ be the corresponding probabilistic PDA. Then $\Pr_{\pi \sim \mathrm{Paths}(e,\bot)}[\pi \models \Box\Diamond\, \mathcal{O}] = 1$ if and only if for every $w \in \{L, R\}^\omega$,*

$$\Pr_{t \sim [\![e]\!]}[t \textit{ has infinitely many output nodes along } w] = 1.$$

In particular, e is ASP if and only if $(e, \varepsilon) \models \mathcal{P}(\Box\, \Diamond\, \mathcal{O}) = 1$.

Proof. The main result we need to prove is that given a distribution μ over OT and the distribution μ' over OS induced by μ,

$$\Pr_{\pi \sim \mu'}[\pi \models \Box\Diamond\mathcal{O}] = 1 \iff \forall w \in \{L, R\}^\omega.\ \Pr_{t \sim \mu}[\pi \text{ has infinitely many output nodes along } w] = 1.$$

After this, all that remains is to check that the distribution over the runs of $\mathcal{A}_e$ starting on (e, ε) is exactly μ', which is done using similar techniques as in the proof of 17.

We start by showing how to compute this induced distribution. Let $F : X \mapsto A \times X + X$ and $G : X \mapsto A \times X \times X + X$ be the functors that generate OS and OT respectively. We define a natural transformation $G \overset{\rho}{\Rightarrow} \mathcal{D}F$, which will allow us to transform G-coalgebras into $\mathcal{D}F$-coalgebras, and in particular OT into $\mathcal{D}(\mathsf{OS})$. We assign to every object X a morphism $\rho_X : GX \to \mathcal{D}FX$ as follows:

$$\begin{aligned}
\rho_X &: A \times X \times X + X \to \mathcal{D}(A \times X + X) \\
\rho_X(\mathit{inr}(x)) &= \delta(\mathit{inr}(x)) \\
\rho_X(\mathit{inl}(a, x, y)) &= 1/2 \cdot \delta(\mathit{inl}(a, x)) + 1/2 \cdot \delta(\mathit{inl}(a, y))
\end{aligned}$$

This gives us a map f from OT to $\mathcal{D}(\mathsf{OS})$ as the unique coalgebra homomorphism closing the following commutative diagram (in the Kleisli category):

$$\begin{tikzcd}
\mathsf{OT} \arrow[r, dashed, "f" marking, "\circ" description] \arrow[d, "\circ" description, "\rho_{\mathsf{OT}} \circ \langle \mathsf{out}, \mathsf{unf} \rangle^{-1}" right] & \mathsf{OS} \arrow[d, "\circ" description, "\langle \mathsf{head}, \mathsf{tail} \rangle" right] \\
F(\mathsf{OT}) \arrow[r, dashed, "Ff", "\circ" description] & F(\mathsf{OS})
\end{tikzcd}$$

We can see that, by uniqueness:

$$\begin{aligned} f : \mathsf{OT} &\to \mathcal{D}(\mathsf{OS}) \\ f\ \mathsf{unf}(x) &= \mathcal{D}(\mathsf{unf}(\bullet))(f(x)) \\ f\ \mathsf{out}(a,x,y) &= 1/2 \cdot \mathcal{D}(a : \bullet)(f(x)) + 1/2 \cdot \mathcal{D}(a : \bullet)(f(y)). \end{aligned}$$

Since $\mathcal{D}$ is a monad, we can extend ρ to a natural transformation $\mathcal{D}G \overset{\bar{\rho}}{\Rightarrow} \mathcal{D}F$ assigning to every X a morphism $\bar{\rho}_X = m_{FX} \circ \mathcal{D}\rho_X$, where m is the product of the $\mathcal{D}$ monad. Given a final $\mathcal{D}G$-coalgebra $(\mathcal{D}(\mathsf{OT}), h)$, we have a $\mathcal{D}F$-algebra $(\mathcal{D}(\mathsf{OT}), \rho_{\mathcal{D}(\mathsf{OT})} \circ h)$ so there is a unique $\mathcal{D}F$-homomorphism $\hat{f}$ to the final $\mathcal{D}F$-coalgebra $(\mathcal{D}(\mathsf{OS}), g)$. This allows us to give semantics in $D(OS)$ to a tree term:

$$\begin{tikzcd}
\mathsf{OT} \arrow[d, "h"] \arrow[rrr, dashed, bend left, "\hat{f}"] & \mathbb{T} \arrow[l, dashed, "\llbracket - \rrbracket_{\mathsf{OT}}"'] \arrow[r, equal] \arrow[d, "\mathsf{st}"] & \mathbb{T} \arrow[r, dashed, "\llbracket - \rrbracket_{\mathsf{OS}}"] \arrow[d, "\bar{\rho}_{\mathbb{T}} \circ \mathsf{st}"] & \mathsf{OS} \arrow[d, "g"] \\
G(\mathsf{OT}) \arrow[d, "\bar{\rho}_{(\mathsf{OT})}"] & G(\mathbb{T}) \arrow[l, dashed] \arrow[r, "\bar{\rho}_{\mathbb{T}}"] & F(\mathbb{T}) \arrow[r, dashed] & F(\mathsf{OS}) \\
F(\mathsf{OT}) \arrow[urrr, dashed, bend right, "F\hat{f}"] & & &
\end{tikzcd}$$

Notice that, by uniqueness:

$$\hat{f}(M)(E) = (m_{\mathsf{OS}} \circ (\mathcal{D}f))(M)(E) = \int_{t \in \mathsf{OT}} \left(\int_{\pi \in \mathsf{OS}} \chi_E(\pi) df(t) \right) dM$$

where χ_E is the characteristic function of $E \subseteq \mathsf{OS}$. Now, let

$$\begin{aligned} S &= \{\pi \mid \pi \models \Box\Diamond\, \mathcal{O}\} \\ P_\pi &= \{t \mid t_\pi \in S\} \end{aligned}$$

where t_π for $\pi \in \{L, R\}^\omega$ is the path in t corresponding to the choices π:

$$\begin{aligned} \mathsf{unf}(x)_{L:w} &= \mathsf{unf}(x_{L:w}) \\ \mathsf{unf}(x)_{R:w} &= \mathsf{unf}(x_{L:w}) \\ \mathsf{out}(a,x,y)_{L:w} &= a : x_w \\ \mathsf{out}(a,x,y)_{R:w} &= a : y_w \end{aligned}$$

Then

$$\Pr_{\pi \sim \mu}[\pi \models \Box\Diamond\, \mathcal{O}] = \int_{\pi \in \mathsf{OS}} \chi_S(\pi) d\mu$$

and

$$\Pr_{t \sim \mu}[t_\pi \text{ has infinitely many output nodes}] = \int_{t \in \mathsf{OT}} \chi_{P_\pi}(t) d\mu.$$

The rest of the proof proceeds in three steps. In the following, let U be the distribution on $\{L, R\}^\omega$ assigning probability $(1/2)^k$ to every cone generated by a prefix of length k. First we need the following lemma stating that the distribution induced by f on OS is the same as the distribution induced by taking paths sampled from U. Intuitively, we are just pre-sampling the randomness in f from U:

▶ **Lemma 19.** *Let $t \in \mathsf{OT}$. Then*

$$\int_{w\in\{L,R\}^\omega} \chi_{P_w}(t)dU = \int_{\pi\in\mathsf{OS}} \chi_S(\pi)df(t).$$

Proof of Lemma 19. We show that for every measurable $B \subseteq \mathsf{OS}$,

$$\int_{w\in\{L,R\}^\omega} \chi_B(t_w)dU = \int_{\pi\in\mathsf{OS}} \chi_B(\pi)df(t).$$

For any distribution $N \in \mathcal{D}(\{L,R\}^\omega)$, there is an induced distribution by t and N on $\mathcal{D}(\mathsf{OS})$:

$$\begin{aligned} g : \mathsf{OT} &\to \mathcal{D}(\{L,R\}^\omega) \to \mathcal{D}(\mathsf{OS}) \\ g\,\mathsf{unf}(x)\;N &= \mathcal{D}(\mathsf{unf}(\bullet))(g\;x\;N) \\ g\,\mathsf{out}(n,x,y)\;N &= \Pr_{w\sim N}[w[0]=L]\cdot\mathcal{D}(n:\bullet)(g\,x\,(N\mid^L_{\mathsf{tail}})) + \Pr_{w\sim N}[w[0]=R]\cdot\mathcal{D}(n:\bullet)(g\,y\,(N\mid^R_{\mathsf{tail}})) \end{aligned}$$

where $N\mid^X_{\mathsf{tail}}$ is the distribution on the tails of N conditioned to the head being X. (if it is empty, we just make that side of the sum 0) What we are doing is taking from the first position of a w sampled from N the randomness for deciding which branch of the tree to take. In particular, it is easy to see that $g\;t\;U = f(t)$. Therefore,

$$\int_{w\in\{L,R\}^\omega} \chi_B(t_w)dU = \int_{\pi\in\mathsf{OS}} \chi_B(\pi)d(g\;t\;U) = \int_{\pi\in\mathsf{OS}} \chi_B(\pi)df(t).$$

◼

Then following lemma shows that a tree produces paths with infinitely many outputs along every w in $\{L,R\}^\omega$ with probability 1 if and only if it produces paths with infinitely many outputs along a w sampled from $\{L,R\}^\omega$ with probability 1. In other words, it provides a connection between the universal quantification in the definition of ASP and the probabilistic nature of pCTL*.

▶ **Lemma 20.** *Let $M \in \mathcal{D}(\mathsf{OT})$. Then, the following are equivalent:*

1. *For every $w \in \{L,R\}^\omega$, $\int_{\mathsf{OT}} \chi_{P_w} dM = 1$.*
2. *$\int_{w\in\{L,R\}^\omega} \int_{\mathsf{OT}} \chi_{P_w} dMdU = 1$.*

Proof of Lemma 20. ($1 \Rightarrow 2$) is immediate. To prove ($2 \Rightarrow 1$), we suppose that there is a $w \in \{L,R\}^\omega$ such that $\int_{\mathsf{OT}} \chi_{P_w} dM < 1$, and we show that this must also be true for a $W \subseteq \{L,R\}^\omega$ such that $\int_{v\in\{L,R\}^\omega} \chi_W dU > 0$, and therefore $\int_{v\in\{L,R\}^\omega} \int_{\mathsf{OT}} \chi_{P_v} dMdU < 1$.

To do this, we consider the set $\{w_n\}_{n\in\mathbb{N}}$ of prefixes of w of length n, and the cones $\{C_n\}_{n\in\mathbb{N}}$ generated by them. For each of those prefixes w_i, we can compute the set of observation trees T_i such that every $t \in T_i$ has i output nodes along w_i. This set is measurable (the union of cones of all finite trees of height i satisfying the conditions), $T_1 \supseteq T_2 \supseteq T_3 \supseteq \ldots$, and therefore:

$$1 \geq \int_{T_1} dM \geq \int_{T_2} dM \geq \int_{T_3} dM \ldots$$

But since $P_w = \cap_n T_n$, the limit of this sequence is exactly $\int_{\mathsf{OT}} \chi_{P_w} dM < 1$. Therefore $\int_{T_k} dM < 1$ for some k, and so

$$\begin{aligned} \int_{v\in C_k}\int_{\mathsf{OT}} \chi_{P_v} dMdU &= \int_{v\in C_k}\int_{T_k} \chi_{P_v} dMdU + \int_{v\in C_k}\int_{T_k^c} \chi_{P_v} dMdU = \\ &= \int_{v\in C_k}\int_{T_k} \chi_{P_v} dMdU + 0 < \int_{v\in C_k} dU. \end{aligned}$$

Hence, we conclude

$$\int_{v\in\{L,R\}^\omega}\int_{\mathsf{OT}}\chi_{P_v}dMdU = \int_{v\in C_k}\int_{\mathsf{OT}}\chi_{P_v}dMdU + \int_{v\in C_k^c}\int_{\mathsf{OT}}\chi_{P_v}dMdU$$
$$< \int_{v\in C_k} dU + \int_{v\in C_k^c} dU = 1.$$

■

Finally we show that given $M \in \mathcal{D}(\mathsf{OT})$, $\hat{f}(M)$ produces paths with infinitely many outputs with probability 1 if and only if $\mathcal{D}(\mathsf{OT})$ is ASP.

▶ **Lemma 21.** *Let $M \in \mathcal{D}(\mathsf{OT})$. Then,*

$$\int_{\mathsf{OT}}\int_{\mathsf{OS}}\chi_S df(t)dM = 1 \iff \forall w \in \{L,R\}^\omega.\ \int_{\mathsf{OT}}\chi_{P_w}dM = 1.$$

This is immediate by Lemma 20, Fubini's theorem and Lemma 19.

◀

7 Possible Generalizations and Extensions

Our definition of ASP and our verification approaches suggest several natural directions for future investigation. We believe that some extensions can be handled without too much trouble; however, other generalizations may require new ideas.

Handling Richer Languages The most concrete direction is to consider richer languages for coinductive probabilistic programming. Starting from our core language, one might consider allowing more operations on coinductive terms, mutually recursive definitions, or conditional tests of some kind. It should also be possible to develop languages for more complex coinductive types associated with general polynomial functors (see, e.g., Kozen [27]). Note that adding more operations, e.g. point-wise + of streams would increase the expressivity of the language but have additional challenges from the perspective of the semantics – we would have to add extra structure to the base category and re-check that the finality proof still works.

Developing new languages for coinductive probabilistic programming—perhaps an imperative language or a higher-order language—would also be interesting. From the semantics side, our development in Section 3 should support any language equipped with a small-step semantics producing output values, allowing ASP to be defined for many kinds of languages. The verification side appears more challenging; our techniques are specialized to our core language. Natural extensions, like a pointwise addition operation, already seem to pose challenges for the analyses. As of now, we know of no general method to reasoning about ASP. This stands in sharp contrast to almost sure termination, which can be established by where flexible criteria like decreasing probabilistic variants [20]. Considering counterparts of these methods for ASP is an interesting avenue of research.

Exploring Other Definitions Our definition of ASP is natural, but other definitions are possible. For trees (and possibly more complex coinductive structures), we could instead require that there exists a path producing infinitely many outputs, rather than requiring that all paths produce infinitely many outputs. This weaker notion of ASP can be defined in our semantics, but it is currently unclear how to verify this kind of ASP.

Our notion of ASP also describes just the probability of generating infinitely many outputs, and does not impose any requirement on the generation rate. Quantitative strengthenings of ASP—say, requiring bounds on the expected number of steps between outputs—could give more useful information.

Understanding Dependence on Step Relation Our coalgebraic semantics supporting our verification methods are based on a small-step semantics for programs. A natural question is whether this dependence is necessary, or if one could verify ASP with a less step-dependent semantics. Again drawing an analogy, it appears that fixing a reduction strategy is important in order to give a well-defined notion of almost sure termination for probabilistic higher-order languages (see, e.g., [29]). The situation for almost sure productivity is less clear.

8 Related Work

Our work draws inspiration from two previously independent lines of research: probabilistic termination and productivity of coalgebraic definitions.

Probabilistic Termination There are a broad range of techniques for proving termination of probabilistic programs. Many of the most powerful criteria use advanced tools from probability theory [31], especially martingale theory [9, 18, 10, 11, 12]. Other works adopt more pragmatic approaches, generally with the goal of achieving automation. Arons, Pnueli and Zuck [4] reduce almost sure termination of a program P to termination of a non-deterministic program Q, using a planner that must be produced by the verifier. Subsequent work by Esparza, Gaiser and Kiefer [17] give a CEGAR-like approach for building patterns—which play a role similar to planners—and proving that their approach is complete for a natural class of programs.

Productivity of Corecursive Definitions There has been a significant amount of work on verifying productivity of corecursive definitions, without probabilistic choice. Endrullis and collaborators [16] give a procedure for deciding productivity of an expressive class of stream definitions. In a companion work [15], they study the strength of data oblivious criteria, i.e., criteria that do not depend on values. More recently, Komendantskaya and collaborators [24] define the notion of observational productivity and give a semi-decision procedure for logic programs.

9 Conclusion

We introduce almost sure productivity, a counterpart to almost sure termination for probabilistic coinductive programs. In addition, we propose two methods for proving ASP for a core language for streams and infinite trees. Our results demonstrate that verification of ASP is feasible and can even be decidable for simple languages. Our work can be seen as an initial exploration into productivity and probabilistic coalgebraic definitions, with many avenues for extensions to more complex languages and generalizations to other datatypes.

References

1 Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In Roberto Giacobazzi and Radhia Cousot, editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL),*

Rome, Italy, pages 27–38, 2013. URL: `http://doi.acm.org/10.1145/2429069.2429075`, `doi:10.1145/2429069.2429075`.

2 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 2(POPL):34:1–34:32, 2018. URL: `http://doi.acm.org/10.1145/3158122`, `doi:10.1145/3158122`.

3 Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Ales Bizjak, Marco Gaboardi, and Deepak Garg. Relational reasoning for Markov chains in a probabilistic guarded lambda calculus. In *European Symposium on Programming (ESOP), Thessaloniki, Greece*, Lecture Notes in Computer Science. Springer-Verlag, 2018.

4 Tamarah Arons, Amir Pnueli, and Lenore D. Zuck. Parameterized verification by probabilistic abstraction. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2620 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 2003. URL: `https://doi.org/10.1007/3-540-36576-1_6`, `doi:10.1007/3-540-36576-1_6`.

5 Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19(3):357–367, 1967.

6 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

7 Tomás Brázdil, Václav Brozek, and Vojtech Forejt. Branching-time model-checking of probabilistic pushdown automata. *Electronic Notes in Theoretical Computer Science*, 239:73–83, 2009. URL: `https://doi.org/10.1016/j.entcs.2009.05.031`, `doi:10.1016/j.entcs.2009.05.031`.

8 Venanzio Capretta and Jonathan Fowler. The continuity of monadic stream functions. In *IEEE Symposium on Logic in Computer Science (LICS), Reykjavik, Iceland*, pages 1–12, 2017. URL: `https://doi.org/10.1109/LICS.2017.8005119`, `doi:10.1109/LICS.2017.8005119`.

9 Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In *International Conference on Computer Aided Verification (CAV), Saint Petersburg, Russia*, pages 511–526, 2013. URL: `https://www.cs.colorado.edu/~srirams/papers/cav2013-martingales.pdf`.

10 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through Positivstellensatz's. In *International Conference on Computer Aided Verification (CAV), Toronto, Ontario*, volume 9779 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2016. URL: `https://doi.org/10.1007/978-3-319-41528-4_1`, `doi:10.1007/978-3-319-41528-4_1`.

11 Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Saint Petersburg, Florida*, pages 327–342, 2016. URL: `https://doi.acm.org/10.1145/2837614.2837639`, `doi:10.1145/2837614.2837639`.

12 Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*, pages 145–160, 2017. URL: `https://doi.acm.org/10.1145/3009837.3009873`, `doi:10.1145/3009837.3009873`.

13 Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. The guarded lambda-calculus: Programming and reasoning with guarded recursion for coinductive types. *Logical Methods in Computer Science*, 12(3), 2016. URL: `https://doi.org/10.2168/LMCS-12(3:7)2016`, `doi:10.2168/LMCS-12(3:7)2016`.

14 Rick Durrett. *Probability: Theory and Examples*. Cambridge University Press, 2010.

15 Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks. Data-oblivious stream productivity. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Doha, Qatar*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer-Verlag, 2008. URL: https://doi.org/10.1007/978-3-540-89439-1_6, doi:10.1007/978-3-540-89439-1_6.

16 Jörg Endrullis, Clemens Grabmayer, Dimitri Hendriks, Ariya Isihara, and Jan Willem Klop. Productivity of stream definitions. *Theoretical Computer Science*, 411(4-5):765–782, 2010. URL: https://doi.org/10.1016/j.tcs.2009.10.014, doi:10.1016/j.tcs.2009.10.014.

17 Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving termination of probabilistic programs using patterns. In P. Madhusudan and Sanjit A. Seshia, editors, *International Conference on Computer Aided Verification (CAV), Berkeley, California*, volume 7358 of *Lecture Notes in Computer Science*, pages 123–138. Springer-Verlag, 2012. URL: https://doi.org/10.1007/978-3-642-31424-7_14, doi:10.1007/978-3-642-31424-7_14.

18 Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In Sriram K. Rajamani and David Walker, editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India*, pages 489–501, 2015. URL: http://doi.acm.org/10.1145/2676726.2677001, doi:10.1145/2676726.2677001.

19 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994. URL: https://doi.org/10.1007/BF01211866, doi:10.1007/BF01211866.

20 Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983. URL: http://doi.acm.org/10.1145/2166.357214, doi:10.1145/2166.357214.

21 John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), St. Petersburg Beach, Florida*, pages 410–423, 1996. URL: http://doi.acm.org/10.1145/237721.240882, doi:10.1145/237721.240882.

22 Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS), Milan, Italy*, volume 9234 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2015. URL: https://doi.org/10.1007/978-3-662-48057-1_24, doi:10.1007/978-3-662-48057-1_24.

23 Henning Kerstan and Barbara König. Coalgebraic trace semantics for continuous probabilistic transition systems. *Logical Methods in Computer Science*, 9(4), 2013. URL: https://doi.org/10.2168/LMCS-9(4:16)2013, doi:10.2168/LMCS-9(4:16)2013.

24 Ekaterina Komendantskaya, Patricia Johann, and Martin Schmidt. A productivity checker for logic programming. In Manuel V. Hermenegildo and Pedro López-García, editors, *International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR), Edinburgh, Scotland*, volume 10184 of *Lecture Notes in Computer Science*, pages 168–186. Springer-Verlag, 2016. URL: https://doi.org/10.1007/978-3-319-63139-4_10, doi:10.1007/978-3-319-63139-4_10.

25 Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981. URL: https://doi.org/10.1016/0022-0000(81)90036-2, doi:10.1016/0022-0000(81)90036-2.

26 Dexter Kozen. A probabilistic PDL. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *ACM SIGACT Symposium on Theory of Computing (STOC), Boston, Massachusetts*, pages 291–297, 1983. URL: `http://doi.acm.org/10.1145/800061.808758`, `doi:10.1145/800061.808758`.

27 Dexter Kozen. Realization of coinductive types. *Electronic Notes in Theoretical Computer Science*, 276:237–246, 2011. URL: `https://doi.org/10.1016/j.entcs.2011.09.024`, `doi:10.1016/j.entcs.2011.09.024`.

28 Antonín Kucera, Javier Esparza, and Richard Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006. URL: `https://doi.org/10.2168/LMCS-2(1:2)2006`, `doi:10.2168/LMCS-2(1:2)2006`.

29 Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. In Hongseok Yang, editor, *European Symposium on Programming (ESOP), Uppsala, Sweden*, volume 10201 of *Lecture Notes in Computer Science*, pages 393–419. Springer-Verlag, 2017. URL: `https://doi.org/10.1007/978-3-662-54434-1_15`, `doi:10.1007/978-3-662-54434-1_15`.

30 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009. URL: `https://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf`.

31 Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages*, 2(POPL):33:1–33:28, 2018. URL: `http://doi.acm.org/10.1145/3158121`, `doi:10.1145/3158121`.

32 Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996. URL: `http://doi.acm.org/10.1145/229542.229547`, `doi:10.1145/229542.229547`.