Almost Sure Productivity

Alexandra Silva

joint work with Alejandro Aguirre, Gilles Barthe, and Justin Hsu

arXiV:1802.06283

Productivity

$$s : \mathbb{R}^{\omega}$$

Productivity

$$
\begin{tikzcd}
\boldsymbol{s} : \mathbb{R}^\omega \arrow[d, "<hd,tl>"', "\cong"] \\
\mathbb{R} \times \mathbb{R}^\omega
\end{tikzcd}
$$

Productivity

$$\begin{tikzcd} s : \mathbb{R}^\omega \arrow[d, "<hd{,}tl>"', "\cong"] \\ \mathbb{R} \times \mathbb{R}^\omega \end{tikzcd}$$

every finite part of the infinite structure can be evaluated in finite time

$$\boldsymbol{s} : \mathbb{R}^\omega$$

$s = 1 : s$

$1 : 1 : 1 : \cdots$

$$s = 1 : (ones + s)$$
$$ones = 1 : ones$$

$1 : 2 : 3 : 4 : \cdots$

$$s = 0 : zip(inv(even(s)), tail(s))$$

$$inv(0 : s) = 1 : inv(s) \quad inv(1 : s) = 0 : inv(s)$$

$$even(x : s) = x : odd(s) \quad odd(x : s) = even(s)$$

$$zip(1 : s, t) = 1 : zip(t, s)$$

$$s = 0 : zip(inv(even(s)), tail(s))$$

$$inv(0 : s) = 1 : inv(s) \quad inv(1 : s) = 0 : inv(s)$$

$$even(x : s) = x : odd(s) \quad odd(x : s) = even(s)$$

$$zip(1 : s, t) = 1 : zip(t, s)$$

$0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : \cdots \qquad Thue - Morse\ Sequence$

$$s = 0 : zip(inv(even(s)), tail(s))$$

$$inv(0 : s) = 1 : inv(s) \quad inv(1 : s) = 0 : inv(s)$$

$$even(x : s) = x : odd(s) \quad odd(x : s) = even(s)$$

$$zip(1 : s, t) = 1 : zip(t, s)$$

$0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : \cdots$ $\quad$ $Thue - Morse\ Sequence$

$$\{0 \to 01, 1 \to 10\}$$

$$
\begin{aligned}
s &= 0 : 1 : even(s) \\
even(x : s) &= x : odd(s) \\
odd(x : s) &= even(s)
\end{aligned}
$$

$$
\begin{aligned}
s &= 0 : 1 : even(s) \\
even(x : s) &= x : odd(s) \\
odd(x : s) &= even(s)
\end{aligned}
$$

$$0 : 1 : 0 : 0 : even^{\omega}$$

Strategies to check for productivity

- Syntactic restriction - cf Coq conductive format
- Term rewriting - productivity via termination

Probabilistic programs

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

productive?

Probabilistic programs

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

productive?

A probabilistic stream computation is **almost surely productive (ASP)** if it produces an infinite stream of outputs with probability 1.

Probabilistic programs

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

Prob of no output = $(1-p)^n$

A probabilistic stream computation is **almost surely productive (ASP)** if it produces an infinite stream of outputs with probability 1.

Probabilistic programs

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

Prob of no output = $(1-p)^n$

A probabilistic stream computation is **almost surely productive (ASP)** if it produces an infinite stream of outputs with probability 1.

Probabilistic programs

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

$$\bar{a} = a : a : a : a : \cdots$$

Probabilistic programs

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

~~ASP~~

$$\bar{a} = a : a : a : a : \cdots$$

Probabilistic programs

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

~~ASP~~

$$\bar{a} = a : a : a : a : \cdots$$

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

Probabilistic programs

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

~~ASP~~

$$\bar{a} = a : a : a : a : \cdots$$

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

$p \leq 1/2$ ASP ✓

$p > 1/2$ ~~ASP~~

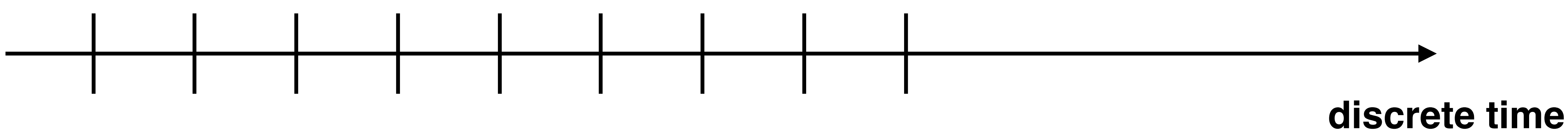
Almost surely productivity

- Formal definition that is language oblivious
- Can be generalised for different conductive types — streams, trees, etc
- Methods to verify ASP

Formal definition

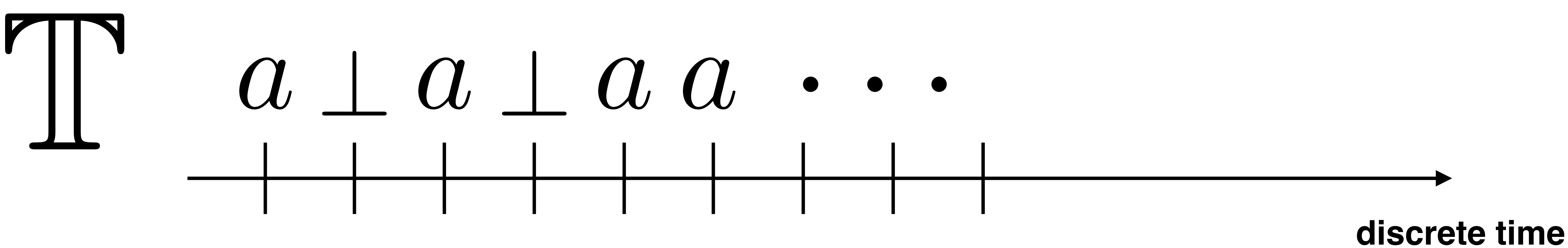
A probabilistic stream computation is **almost surely productive (ASP)** if it produces an infinite stream of outputs with probability 1.

$$\mathbb{T} \quad a \perp a \perp a \; a \; \cdots$$



Formal definition

A probabilistic stream computation is **almost surely productive (ASP)** if it produces an infinite stream of outputs with probability 1.



$$[\![-]\!] \colon \mathbb{T} \to \mathcal{D}((A_\perp)^\omega)$$

(deterministic) Coinductive definitions

A^ω is a final coalgebra $A^\omega \xrightarrow{\cong} A \times A^\omega$

$$
\begin{tikzcd}
\mathbb{P} \arrow[r, dashed, "[\![-]\!]"] \arrow[d, "st"] & A^\omega \arrow[d, "<hd,tl>"] \\
A \times \mathbb{P} \arrow[r, dashed] & A \times A^\omega
\end{tikzcd}
$$

Probabilistic Coinductive definitions

$\mathcal{D}(A^\omega)$ is a final coalgebra but … $Kl(\mathcal{D})$

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

Probabilistic Coinductive definitions

$\mathcal{D}(A^\omega)$ is a final coalgebra but … $Kl(\mathcal{D})$

$$[\![-]\!] : \mathbb{T} \to \mathcal{D}((A_\perp)^\omega)$$

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

Probabilistic Coinductive definitions

$\mathcal{D}(A^\omega)$ is a final coalgebra but … $Kl(\mathcal{D})$

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

Probabilistic Coinductive definitions

$\mathcal{D}(A^\omega)$ is a final coalgebra but … $Kl(\mathcal{D})$

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

Composition of — o —> arrows uses monad multiplication

ASP

A stream program $p \colon \mathbb{T}$ is **almost surely productive (ASP)** if

$$\Pr_{\sigma \sim [\![p]\!]}[\sigma \text{ has infinitely many concrete output elements } a \in A] = 1.$$

ASP

A stream program $p \colon \mathbb{T}$ is **almost surely productive (ASP)** if

$$\Pr_{\sigma \sim [\![p]\!]}[\sigma \text{ has infinitely many concrete output elements } a \in A] = 1.$$

definition parametrised on [[-]] *or* st : T —> A x T

Some remarks

- Definition of monad D involves basic measure theory
- Multiplication is given by integration
- All *events* need to be measurable

Trees

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

Trees

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

A tree program $p\colon \mathbb{T}$ is **almost surely productive (ASP)** if

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

Example: a language for streams

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

Example: a language for streams

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

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

Example: a language for streams

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

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

$$\mathsf{st}_e(\mathsf{tail}^k(a : e)) \triangleq \mathsf{st}_e(\mathsf{tail}^{k-1}(e))$$

$$\mathsf{st}_e(\mathsf{tail}^k(e_1 \oplus_p e_2)) \triangleq \mathsf{st}_e(\mathsf{tail}^k(e_1) \oplus_p \mathsf{tail}^k(e_2))$$

Example: a language for streams

$$e \in \mathbb{T} ::= \sigma \mid e \oplus_p e \mid a : e \;(a \in A) \mid \mathsf{tail}(e)$$

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

$$\mathsf{st}_e(\mathsf{tail}^k(a : e)) \triangleq \mathsf{st}_e(\mathsf{tail}^{k-1}(e))$$

$$\mathsf{st}_e(\mathsf{tail}^k(e_1 \oplus_p e_2)) \triangleq \mathsf{st}_e(\mathsf{tail}^k(e_1) \oplus_p \mathsf{tail}^k(e_2))$$

$$\mathsf{st}_e(a : e') \triangleq \delta(inl(a, e'))$$

$$\mathsf{st}_e(e') \triangleq \delta(inr(e'[e/\sigma])) \quad \text{otherwise}$$

ASP I: syntactic measure

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

Theorem

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

Example

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

$$\#(\sigma) = \; p \cdot 1 + (1-p) \cdot (-1) = 2p - 1$$

Example

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

$$\#(\sigma) = \; p \cdot 1 + (1 - p) \cdot (-1) = 2p - 1$$

$$\#(\sigma) > 0 \iff p > 1/2$$

ASP II : Probabilistic model checking

Idea: Associate with a program a pPDA and reduce ASP to model checking

ASP II : Probabilistic model checking

Idea: Associate with a program a pPDA and reduce ASP to model checking

$$\mathcal{A}_e = (\mathcal{S}_e, \{tl\}, \mathcal{T}_e)$$

$$\mathcal{T}_e((\sigma, a), (e, a)) = 1 \qquad \mathcal{T}_e((a' : e', \bot), (e', \varepsilon)) = 1$$

$$\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_1, a)) = p \qquad \mathcal{T}_e((a' : e', tl), (e', \varepsilon)) = 1$$

$$\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_2, a)) = 1 - p \qquad \mathcal{T}_e((\mathsf{tail}(e'), a), (e', tl \cdot a) = 1$$

ASP II : Probabilistic model checking

Theorem

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

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

In particular, e is ASP if and only if for almost all runs π starting in (e, ε), $\pi \models \Box\,\Diamond\,\mathcal{O}$.

ASP II : Probabilistic model checking

Theorem

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 for almost all runs π starting in (e, ε), $\pi \models \Box \Diamond \mathcal{O}$.

Model checking LTL formulas against pPDAs is decidable (Brazdil et al) => ASP is decidable for stream programs

Conclusions

- Simple definition — non trivial formalisation
- Methods for stream can be extended to trees
- Richer languages, higher-order, other conductive types
- Different notions of ASP - weaker
- (Non-)Dependency on the step relation