Lecture 3:

Pseudodeterministic Constructions and rK^t

Igor Carboni Oliveira

University of Warwick

CIRM - Randomness, Information & Complexity

February/2024

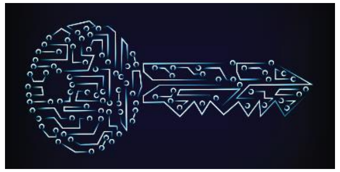
Plan for the Week

Lecture 1 (Monday)

Probabilistic Notions of (Time-Bounded) Kolmogorov Complexity

“**Unconditional** results & applications to average-case complexity”

Lecture 2 (Tuesday)



OWF P vs NP

Connections to Cryptography and Complexity Theory

“Major questions in complexity are **equivalent** to statements about Kolmogorov Complexity”

Lecture 3 (Thursday)

Connections to Algorithms (explicit constructions, generating primes, etc.)

“Existence of large primes with efficient short descriptions”

Primes

An integer is a **prime** if it is only divisible by **1** and **itself**.

n-bit prime: $[2^{n-1}, 2^n - 1]$, i.e., binary representation of the form 1xxxxxxx1

Two fundamental computational problems about **primes**:

Primality Testing: check whether a given n-bit integer is prime

AKS primality test: solves this problem in **deterministic** $\mathrm{poly}(n)$ time

Prime Generation: find an n-bit prime

Focus of this talk

Challenge

- Generating **prime** numbers:
  - **Input**: n
  - **Output**: A **fixed** n-bit prime p_n. (*i.e.*, in $[2^{n-1}, 2^n\text{-}1]$)

- Can we solve this problem **deterministically** in time $\text{poly}(n)$?

A simple approach: **Cramér**

```
Algorithm Cramér:
For i ← 2^(n−1) to 2^n − 1
  If i is prime,
    Output i and halt
```

Uses [AKS04] for checking primality!

Cramér's conjecture: Let p_k denote the k-th prime, then $p_{k+1} - p_k = O\left((\log p_k)^2\right)$

Under Cramér's conjecture, this algorithm inspects $O(n^2)$ numbers, so it runs in $\mathrm{poly}(n)$ time.

State-of-the-art:

$$p_{k+1} - p_k = O\left((p_k)^{0.525}\right)$$

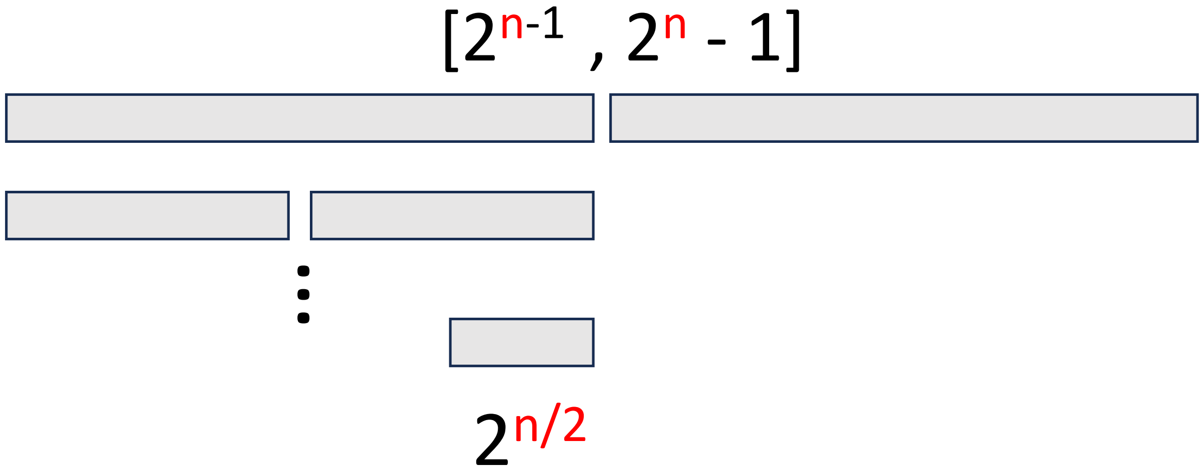
Although Algorithm **Cramér** is conjectured to run in $\mathrm{poly}(n)$ time, the provable guarantee is only $O^*(2^{0.525n})$ time [BHP01]

State of the art

Best known algorithm is due to **[Lagarias-Odlyzko'87]**.

[LO87] employs techniques from analytic number theory to approx. count primes in an interval [a,b]. It has running time guarantee $2^{n/2+o(n)}$.

Idea:



Approx. # primes in each interval in time $2^{n/2}$

Infinitely Often: Mersenne

Infinitely-Often Algorithms

On infinitely many n, the algorithm outputs a prime of length n.

(already a non-trivial notion!)

<u>Conjecture:</u> There are infinitely many *Mersenne primes* (primes of the form $2^n - 1$).

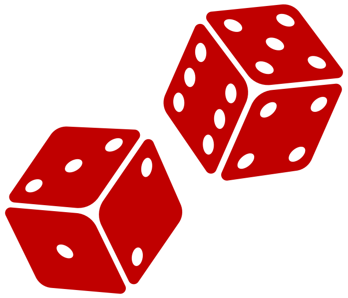
```
Algorithm Mersenne:

Output string is a
sequence of n ones
```

Under this conjecture, **<u>Mersenne</u>** is an infinitely-often polynomial-time algorithm for generating primes.

Generalization: Dense Properties

- A property $Q \subseteq \{0,1\}^*$ is dense, if for every input length n, $|Q \cap \{0,1\}^n| \geq 2^n/\mathrm{poly}(n)$
- PRIMES is dense:
  - **Prime Number Theorem:** there are $\sim N/\ln N$ primes in [1, N]
  - $|\mathrm{PRIMES} \cap \{0,1\}^n| \geq 2^n/100n$
- Explicit construction problem: For a dense property Q, find a length-n string in Q in $\mathrm{poly}(n)$ time.



```
Algorithm Random:
   Sample x ← {0,1}^n
         until x ∈ Q
Output x
```

Easy with **randomness**!

Deterministic algorithms are open

Complexity Theory and Pseudorandomness

$G_{\mathrm{IW}} \subseteq \{0,1\}^n$: The generator from [IW97]

```
Algorithm IW:
For x in G_IW
  If x is a prime
    Output x and halt
```

<u>Algorithm **IW**</u> is **conjectured** to find a prime, but we seem very far from proving this hypothesis

<u>Circuit Lower Bound Hypothesis:</u>
$\mathbf{E}$ requires $2^{\Omega(n)}$-size Boolean circuits

Assuming hypothesis, G_{IW} hits every **<u>dense</u>** property that is **<u>easy</u>** to decide

In particular, G_{IW} **contains a prime!**

State-of-the-art:
$\mathbf{E}$ requires $3.1n - o(n)$ size circuits

Summary

Cramer, Mersenne

Almost-everywhere / infinitely often poly-time (**under conjectures**)

Algorithm IW

Assuming **E requires exponential-size circuits**, **Algorithm IW** runs in $\mathrm{poly}(n)$ time.

State of the art

Time Complexity $O(2^{0.5n})$

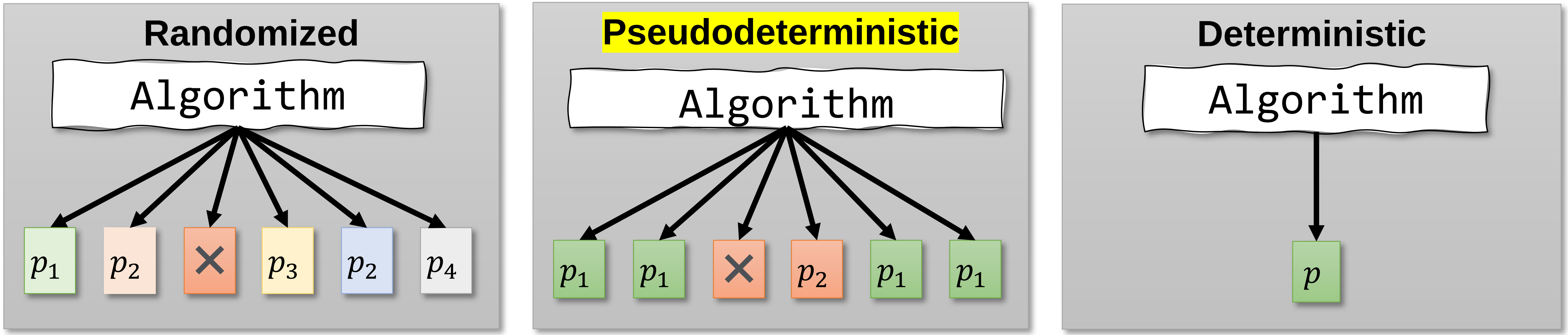
Can we find a prime in $\mathbf{poly}(n)$ time provably?

Polymath 4: Attempted to use **number-theoretic techniques** but did not obtain an **unconditional** improvement.

Relaxing our goal: Pseudodeterminism

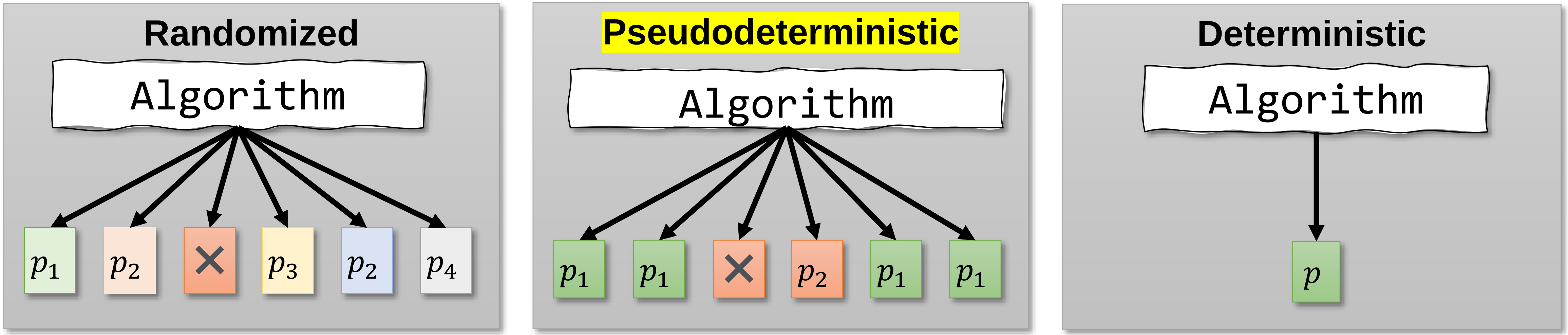
Algorithm **Random**:
Sample $x \leftarrow \{0,1\}^n$
    until $x \in Q$
Output x

Drawback of **Random**:
different primes on **different** executions



A randomized algorithm is **pseudodeterministic** if on most of its computational branches it outputs the **same** answer.

Any bounded observer thinks the algorithm is deterministic.



Literature

Pseudodeterminism was first defined and studied in:

Eran Gat and Shafi Goldwasser **[GG11]**:

“*Probabilistic search algorithms with unique answers and their cryptographic applications*”.

Query complexity [GGR13], [GIPS21, CDM23]

Streaming algorithms [GGMW20], [BKKS23]

Parallel computation [GG17], [GG21]

Learning algorithms [OS18]

Kolmogorov complexity [O19, LOS21]

Space complexity [GL19]

Proof systems [GGH18], [GGH19]

Computational algebra [Gro15]

Approximation algorithms [DPV18]

and more [BB18], [Gol19], [DPWV22], [WDP+22], [CPW23], …

Gat-Goldwasser (2011):

Is there an efficient **pseudodeterministic** algorithm
for generating prime numbers?

More generally,

Is it the case that the generation problem
for every **dense** and **easy** property **Q** can be solved
pseudodeterministically in polynomial time?

Relevant previous work

[Oliveira-Santhanam'17]

There is a **sub-exponential** algorithm A such that, for **infinitely many** $n \in \mathbb{N}$, there is a **prime** $p_n \in [2^{n-1}, 2^n)$ such that $A(1^n)$ outputs p_n with probability at least $1 - 2^{-n}$ over its internal randomness.

Poly-time pseudodeterministic constructions

Theorem (Joint work L. Chen, Z. Lu, H. Ren, and R. Santhanam)

There is a **polynomial-time** algorithm A such that, for **infinitely many** $n \in \mathbb{N}$, there is a **prime** $p_n \in [2^{n-1}, 2^n)$ such that $A(1^n)$ outputs p_n with probability at least $1 - 2^{-n}$ over its internal randomness.

Consequence in Kolmogorov Complexity

Corollary. Primes with **succinct** and **efficient** descriptions:

For every integer **m** there is **n > m** and an **n-bit prime p** with $rK^{poly}(\mathbf{p}) = \log \mathbf{n} + O(1)$

Proof: An efficient pseudodeterministic algorithm A and its input 1^n serve as an encoding of the canonical *n*-bit prime **p** such that $\mathbf{p} = A(1^n)$.

What **properties** of primes are used in the Theorem?

- **Density**: A $1/\mathrm{poly}(n)$ fraction of n-bit strings are prime numbers.
- **Easiness**: There is a $\mathrm{poly}(n)$-time deterministic algorithm that checks if a given integer is prime.

Theorem (Main Result)

Let $Q = \{Q_n \subseteq \{0,1\}^n\}_{n \in \mathbb{N}}$ be a property such that:

- (**Dense**) There is a polynomial q such that for all $n \in \mathbb{N}$, $|Q_n| \geq \frac{1}{q(n)} \cdot 2^n$;
- (**Easy**) There is a deterministic poly-time algorithm deciding Q.

Then, there is a **polynomial-time** algorithm A such that, for **infinitely many** $n \in \mathbb{N}$, there is a **canonical solution** $x_n \in Q_n$ such that $A(1^n)$ outputs x_n with probability at least $1 - 2^{-n}$ over its internal randomness.

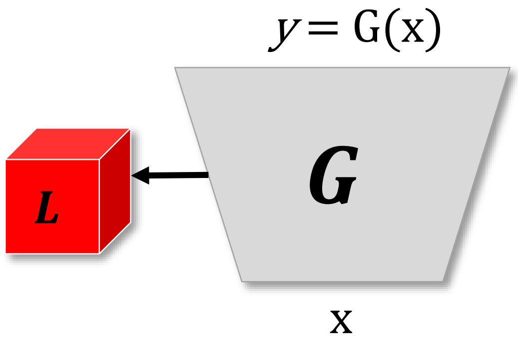
(Previous work **[OS17]** also works for all easy and dense properties)

Warm-up: Sub-exponential time construction [OS17]

There is a pseudodeterministic algorithm that outputs an n-bit prime in $2^{n^{o(1)}}$ time (infinitely often).

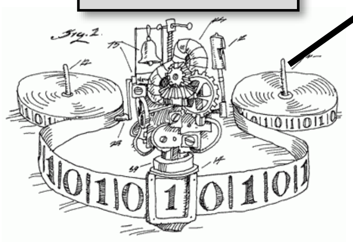
- **Idea I:** **Uniform** hardness vs randomness
- **Idea II:** **Win-win** argument

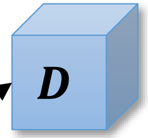
Uniform Hardness vs Randomness



$G = G^L$ is a PRG

Recon





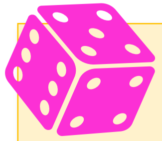
For any D that breaks G, $\mathbf{Recon}^D$ computes L

Trevisan-Vadhan 07

A language L_{TV} with special properties that is **PSPACE-complete**

Corollary

If $G^{L_{\mathrm{TV}}}$ doesn't fool **PRIMES**, then $\mathbf{Recon}^{\mathrm{PRIMES}}$ computes L_{TV}.



Impagliazzo-Wigderson 01

(uniform) **hardness vs randomness**

If L has special properties, given a distinguisher D of G, we can compute L with a uniform reconstruction oracle algorithm $\mathbf{Recon}^D$.

Impagliazzo-Wigderson 97

(non-uniform) **hardness vs randomness**

Given a **distinguisher** D of G, we can compute L with a nonuniform reconstruction oracle algorithm $\mathbf{Recon}^D/_{\mathbf{advice}}$.

Review of the previous approach [OS17]

$m = n^C$ for a large constant C

Candidate HSG

$H^{L_n^{TV}} : \{0,1\}^{O(n)} \to \{0,1\}^m$

L^{TV} on n-bit inputs
(space-n computation)

AKS: $\{0,1\}^m \to \{0,1\}$
accepting a $1/m$ fraction of inputs

Reconstruction Algorithm

$R^{\text{AKS}} : \{0,1\}^n \to \{0,1\}$

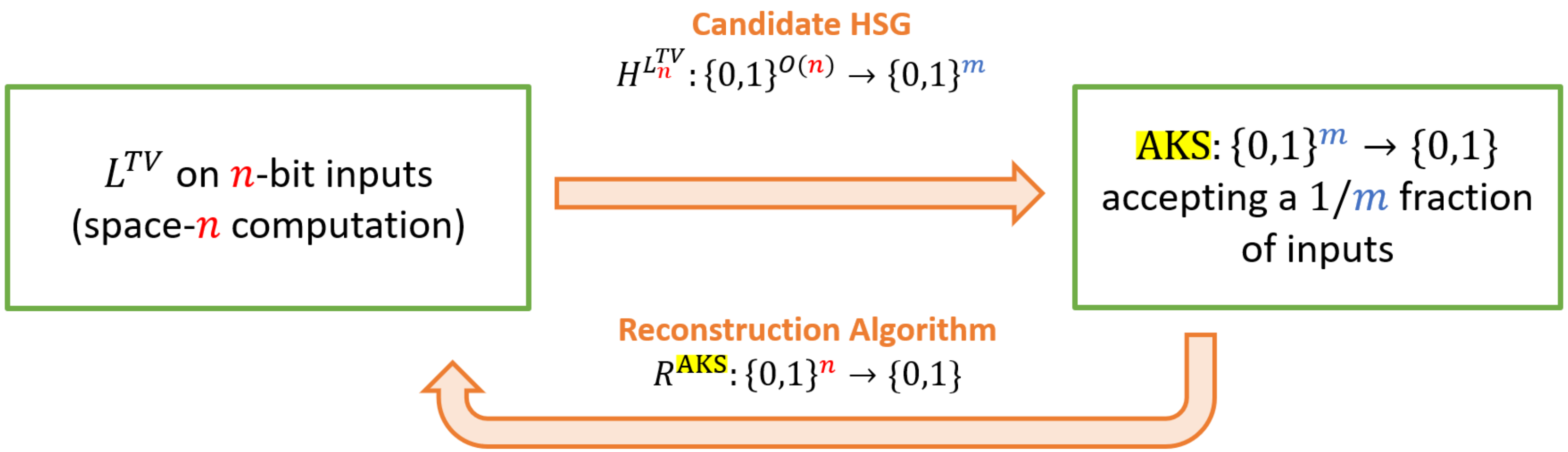
$H^{L_n^{TV}}$ **does not** hit AKS $\Rightarrow$ R^{AKS} **computes** L_n^{TV}

Review of the previous approach [OS17]

$m = n^C$ for a large constant C

Key Idea:

win-win argument



Case HIT: $H^{L_n^{TV}}$ **hits** AKS? We have a hitting set generator!

In $2^{O(n)}$ time, **enumerate** all **outputs** of $H^{L_n^{TV}}$ and find the **first one accepted** by AKS.

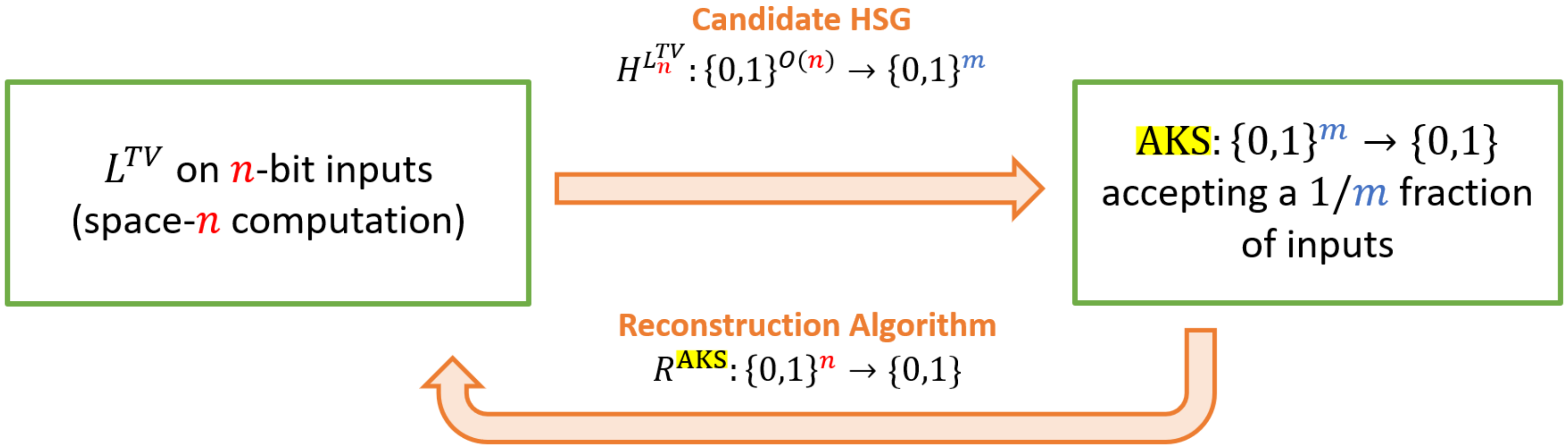
$2^{O(n)} = 2^{m^{1/C}}$-time construction of **a fixed m-bit prime**.

Review of the previous approach [OS17]

$m = n^C$ for a large constant C

Key Idea:

win-win argument



Case AVOID: $H^{L_n^{TV}}$ **does not hit** AKS? We can now compute L^{TV} very **FAST**!

R^{AKS} is a $\text{poly}(m) = \text{poly}(n)$ time **randomized algorithm** for L_n^{TV}

L^{TV} ***covers all space-n computations*** *(naively it takes* $\mathbf{2^n}$ ***time*** *to compute)*

In $O(n)$ space, one can find the lexicographically first n-bit prime

$\Rightarrow \text{poly}(n)$-time **randomized algorithm** that outputs the **lexicographically** first n-bit prime w.h.p.

Digest:

From a <u>special language</u> $L_n^{TV}: \{0,1\}^n \to \{0,1\}$, build $H_n: \{0,1\}^{O(n)} \to \{0,1\}^m$ attempting to hit m-bit primes.

- If it **HITS**, we get a $2^{O(n)}$-time construction of an m-bit prime!
- If it **does not hit** (**AVOIDS**), L_n^{TV} itself is in $\text{poly}(n)$ time, and we use that to get a $\text{poly}(n)$ time construction of an n-bit prime!

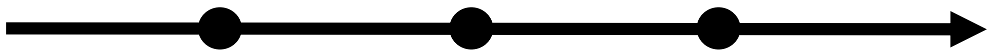
Polynomial time?

AVOID case is FAST,
but **HIT** case is SLOW

Idea:

HIT case still makes non-trivial progress

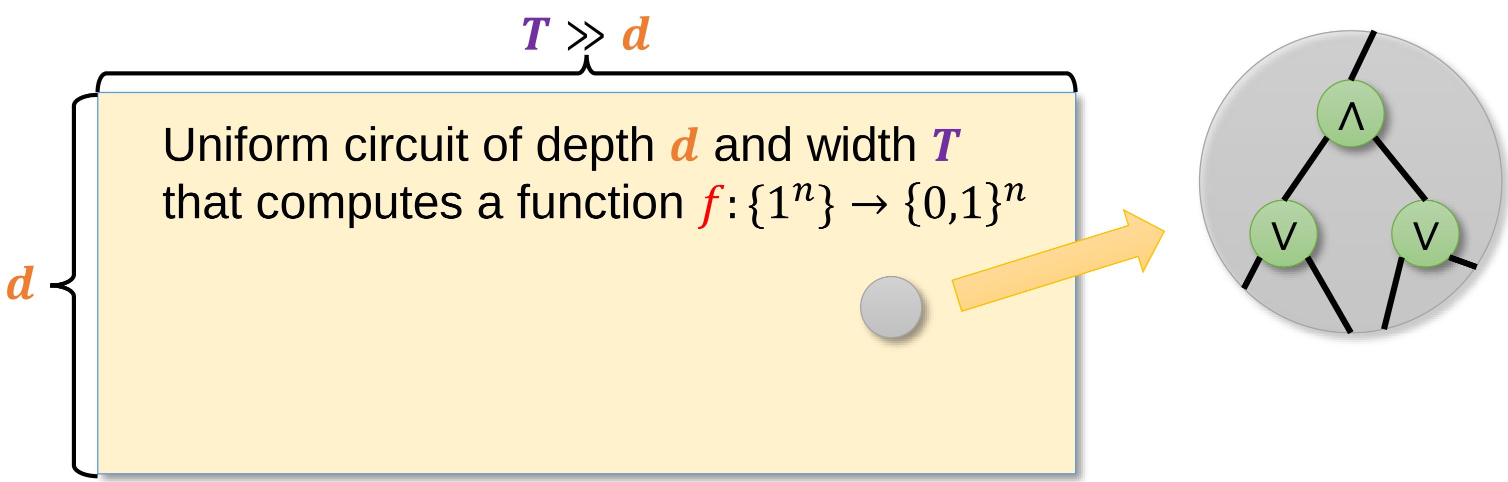
"Iterate"

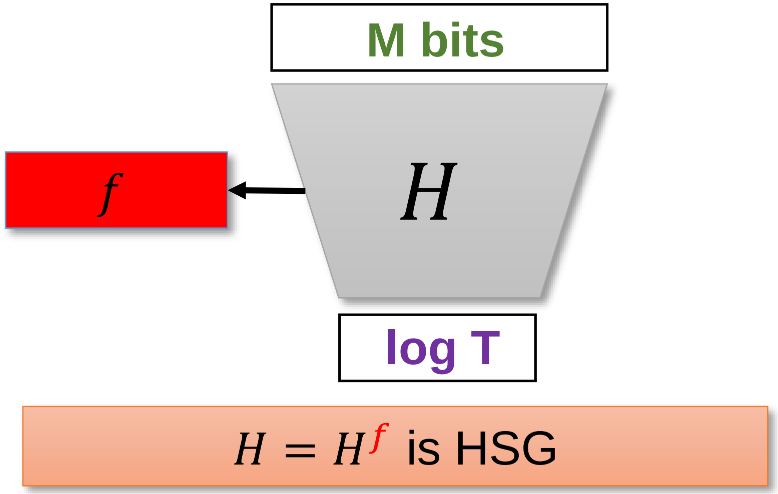


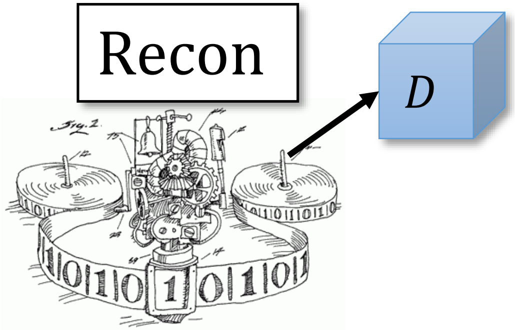
The (ideal) Chen-Tell generator (2021)

Uniform $f: \{1^n\} \to \{0,1\}^n$

(Circuit of depth d and width T)







For any dense D that avoids H, $\mathrm{Recon}^D(1^n)$ computes $f(1^n)$ in randomized time $\mathrm{poly}(d, M)$

Chen-Tell: For any integer M such that log T < M < T:

HSG H: $\{0,1\}^{\log T} \to \{0,1\}^M$ computable in poly(T) time.

Pseudodeterministic constructions from [CT21]

$T = 2^{O(n)}$

$d = \mathrm{poly}(n)$

$\mathbf{BF_0}$: Brute Force
Enumerates all strings of length n, and outputs the first n-bit prime

Fix $M = \mathrm{poly}(n)$

$\mathrm{BF_0}$ ← H

$\log T = O(n)$

Plug in $f = \mathrm{BF_0}$ as the "hard function"

Reconstruction guarantee:
If $\mathbf{AKS}_M$ avoids $H^{\mathrm{BF_0}}$, then one can speed-up the computation of $\mathrm{BF_0}$ in $\mathrm{poly}(d, M) = \mathrm{poly}(n)$ time.

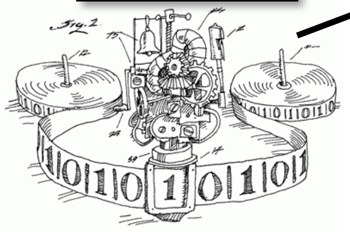
Does $\mathbf{AKS}_M$ avoid $H^{\mathrm{BF_0}}$?

YES: Compute the first length-n prime in randomized (i.e. pseudodet) $\mathrm{poly}(n)$ time

NO: Hitting set $H^{\mathrm{BF_0}}$ that hits $\mathbf{AKS}_M$ and is computable in $2^{O(n)}$ time.

Iterate!

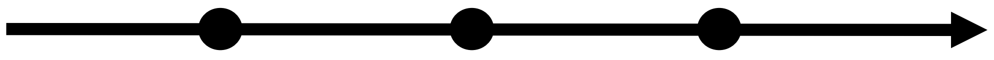
Recon



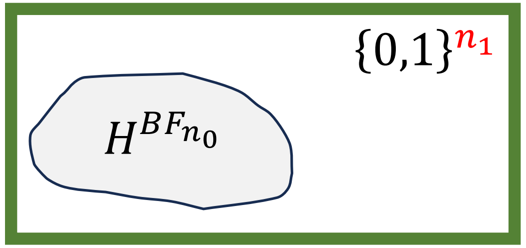
→ AKS_M

Use $\mathbf{AKS}_M$ as distinguisher

The iterated win-win argument



$n_i = n_{i-1}^{\beta}$



$H^{BF_{n_0}}: \{0,1\}^{O(\log T_0)} \to \{0,1\}^{n_1}$

$BF_{n_0}: \{1^{n_0}\} \to \{0,1\}^{n_0}$ outputs the **smallest** n_0-bit prime in $T_0 = 2^{O(n_0)}$ time

Case HIT
$H^{BF_{n_0}}$ **HITS** AKS_{n_1} →

$H^{BF_{n_1}}: \{0,1\}^{O(\log T_1)} \to \{0,1\}^{n_2}$

$BF_{n_1}: \{1^{n_1}\} \to \{0,1\}^{n_1}$ outputs a fixed n_1-bit prime in $T_1 = \mathrm{poly}(T_0)$time

Case HIT
$H^{BF_{n_1}}$ **HITS** AKS_{n_2} →

$H^{BF_{n_2}}: \{0,1\}^{O(\log T_2)} \to \{0,1\}^{n_3}$

$BF_{n_2}: \{1^{n_2}\} \to \{0,1\}^{n_2}$ outputs a fixed n_2-bit prime in $T_2 = \mathrm{poly}(T_1)$ time →

Case AVOID:
$H^{BF_{n_0}}$ **DOES NOT HIT** AKS_{n_1} ↓

poly(n_0)-time construction of fixed n_0-bit prime!

Case AVOID:
$H^{BF_{n_1}}$ **DOES NOT HIT** AKS_{n_2} ↓

poly(n_1)-time construction of fixed n_1-bit prime!

Case AVOID: $H^{BF_{n_2}}$
DOES NOT HIT AKS_{n_3} ↓

poly(n_2)-time construction of fixed n_2-bit prime!

A closer look at each iteration

- n_i = length of prime that we want to find
- H_i = HSG containing an n_i-bit prime
- T_i = size of H_i

$T_0 = 2^{O(n_0)}$

T_i

d_i

$\mathbf{BF}_i$

Enumerate all strings in H_i and output the first prime

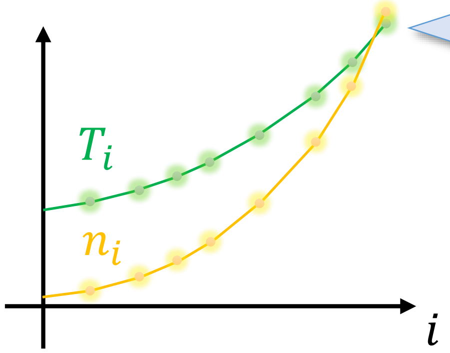
Next hitting set H_{i+1} is H^{BF_i}

Each iteration:

$n_i \leftarrow (n_{i-1})^{\beta}$ for some β that we set

$T_i \leftarrow (T_{i-1})^{\alpha}$ for some α depending on Chen-Tell

Hope: If we set β large enough, $\{n_i\}$ grows faster than $\{T_i\}$



Does $\mathrm{AKS}_{n_{i+1}}$ avoid H^{BF_i}?

YES

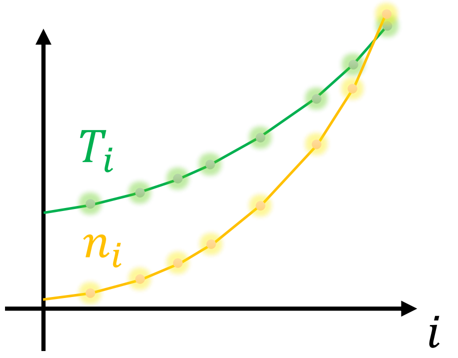
Compute the first prime in H_i in randomized (i.e. pseudodet) $\mathrm{poly}(n_{i+1})$ time

NO

A smaller hitting set $H_{i+1} \coloneqq H^{\mathrm{BF}_i}$ that hits $\mathrm{AKS}_{n_{i+1}}$

$\{n_i\}$ vs $\{T_i\}$

Hope: If we set β large enough, $\{n_i\}$ will grow faster than $\{T_i\}$!



- $n_i = (n_{i-1})^{\beta} \Rightarrow n_i = (n_0)^{\beta^i}$
- $T_i = (T_{i-1})^{\alpha} \Rightarrow T_i = (2^{n_0})^{\alpha^i}$, for some α.
- Want to find t such that T_t = poly(n_t). We set $\beta = 2\alpha$.
- When $t = O(\log n_0)$, a simple computation shows that T_t will be comparable to n_t.

The algorithm and its correctness

```
Algorithm CLORS23:
Let's say n = n_i for some i

If i = t (recall that T_t ≤ poly(n_t))
  Find the first prime in H_t by brute force
Else
  Use Recon^{AKS_{n_{i+1}}} to output a candidate n_i-bit prime
```

If H_t contains a prime

We can find this prime using brute force in polynomial time!

If H_t doesn't contain a prime

- But H_0 does…
- There is some i s.t. H_i contains a prime but $H_{i+1} = H_{\mathrm{BF}_i}$ does not.
- $\mathrm{AKS}_{n_{i+1}}$ avoids H_{BF_i}, so $\mathrm{Recon}^{\mathrm{AKS}_{n_{i+1}}}$ computes BF_i correctly!

Omitted Technical Details

- The **HSG** of **[CT21] doesn't** apply to **all uniform computations**: only to **low-depth uniform circuits**. Luckily, the algorithms BF_{n_i} we constructed can be implemented by **low-depth uniform circuits**.

- The original **[CT21]** paper gives a **HSG** with $O(\frac{\log^2 T}{\log M})$ seed length instead of $O(\log T)$. This only gives a **quasi-poly** time construction instead of **poly-time**.

- We improve **Chen-Tell** by combining it with the **Shaltiel-Umans PRG [SU05]**.
  This requires extra work (the original **SU** reconstruction algorithm is **not** uniform).

Open Problems

Main Challenge: Make the result work on **all input lengths** (or reduce gap)?

[OS17] achieves **zero-error** (it outputs the **canonical prime** or ``**FAILURE**'').

Can we get a zero-error polynomial-time infinitely-often algorithm?

[OS17] works for every dense property in **BPP**. We require the property **Q** to be in **P**.

Main Reference for Lecture 3:

Paper: **“Polynomial-time pseudodeterministic construction of primes”** (2023)

(Joint work with L. Chen, Z. Lu, H. Ren, and R. Santhanam)

Thank you