Computational Complexity and Entropy

David Zuckerman
University of Texas at Austin

Computational Complexity

- What makes some problems hard to compute and others easy?
- Trade-offs between various resources
  - time
  - space
  - randomness

Randomness and Computation

Randomization extremely useful:

- Algorithms (e.g. primality testing, factoring polynomials, approximation algorithms, min cut, load balancing)
- Monte Carlo simulations
- Network constructions
- Cryptography
- Distributed computing

Outline of Talk

Brief introduction to randomized computation

Min-entropy:

- Appropriate measure of minimum randomness requirement.
- Can extract min-entropy using small number of auxiliary random bits.
- Can condense min-entropy: expander graph constructions.
- Good pseudorandom generators by extracting min-entropy.

Computational entropy + extracting min-entropy:

- General pseudorandom generators from 1-way functions.

Example: Matrix Multiplication Check

$\text{MMC} = \{\langle A, B, C\rangle | AB = C \text{ for square matrices } A,\ B,\ C\}$

Deterministic: matrix multiplication $O(n^3)$ ($n \times n$ matrices)
more sophisticated $O(n^{2.376\ldots})$

Randomized algorithm:

- Choose $v \in \{0,1\}^n$ uniformly at random.
- Accept iff $A(Bv) = Cv$.

Analysis:

- Time: $O(n^2)$.
- Correctness:

$$\begin{aligned} AB &= C \Rightarrow \Pr[\text{accept}] = 1 \\ AB &\neq C \Rightarrow \Pr[\text{accept}] \leq 1/2 \end{aligned}$$

- Improve error – repeat 100 times $\leq 1/2^{100}$.

Complexity Classes

P = {languages decidable by polynomial-time algorithms}.

RP = {languages L decidable by randomized ptime algorithms}:

$$x \in L \Rightarrow \Pr[\text{accept}] \;\geq\; 1/2 \quad (1 - 2^{-100})$$
$$x \notin L \Rightarrow \Pr[\text{accept}] \;=\; 0$$

$\overline{\text{MMC}}, \textsc{Composites}, \textsc{Primes} \in \text{RP}$. $\text{P} \subseteq \text{RP}$.

2-sided error: BPP.
0-sided error: $\text{ZPP} = \text{RP} \cap \text{coRP}$.

Difficulty with Randomization

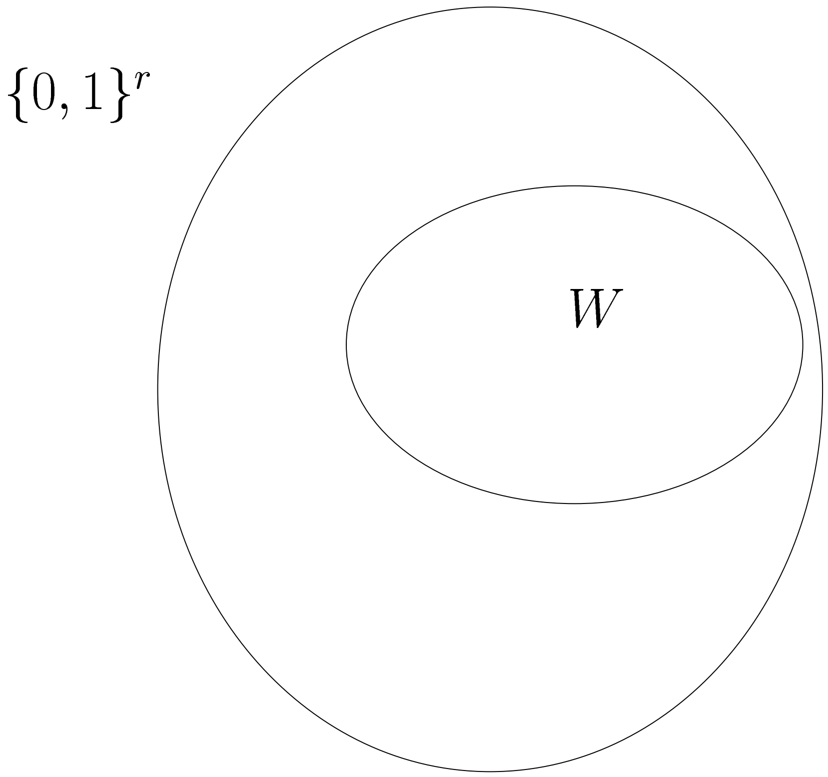
Computers don't have access to truly random bits.

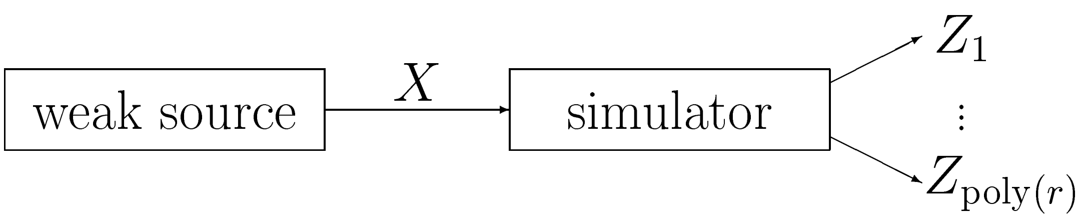
Important questions:

- Does RP = P?
- Are weakly random bits sufficient?
- Is a small number of truly random bits sufficient?

Goal for using weak random sources

Simulating "Abstract-RP": arbitrary W, $|W| \geq 2^{r-1}$.





$$(\forall W) \Pr_X[(\exists i) Z_i \in W] \geq 1 - o(1)$$

Simulation should work for *any* weak source meeting minimum randomness requirement.

Minimum Randomness Requirement

First try: lower bound entropy

$$H(X) = \sum_i p_i \log(1/p_i)$$

Problem: Say $H(X) \geq r/2$. $\Pr[X = 0^r] = 1/2$.

Better: lower bound Renyi (quadratic) entropy

$$H_2(X) = -\log(\sum_i p_i^2) = -\log \Pr[X = X'] \leq H(X)$$

Min-Entropy

More convenient: lower bound min-entropy

$$H_\infty(X) = \min \log(1/p_i)$$

$$H_\infty(X) \leq H_2(X) \leq 2H_\infty(X)$$

Randomness requirement:

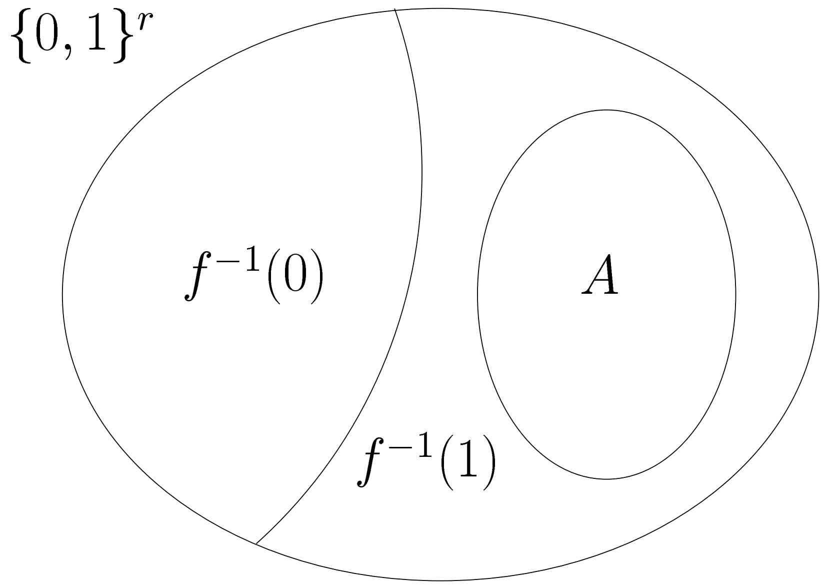
$$H_\infty(X) \geq k \iff (\forall x) \Pr[X = x] \leq 2^{-k}.$$

Worst case: uniform on A, $|A| = 2^k$.

Think of $k = r/100$ or $k = r^{1/100}$.

Can't extract 1 good bit

Suppose $f : \{0,1\}^r \longrightarrow \{0,1\}$ extracts.
$\exists$ source concentrating all probability on larger of $f^{-1}(0)$ or $f^{-1}(1)$.



Simulating RP

Possible in polynomial time.

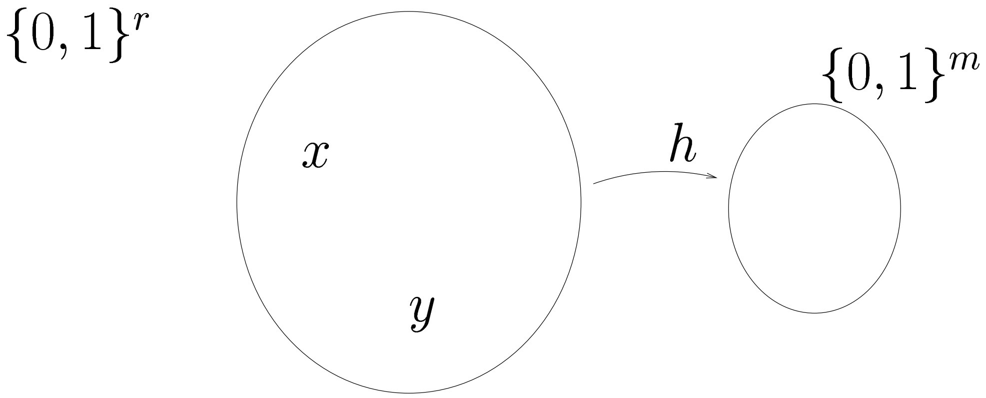
- $H_\infty(X) \geq \Omega(r)$ [Z]
- $H_\infty(X) \geq r^{\Omega(1)}$ [Saks, Srinivasan, Zhou]

Useful Tool: Random Catalyst

$$\begin{tikzcd} \boxed{H_\infty(X) \geq k} \arrow[d] & \boxed{\text{random}} \arrow[d] \\ \boxed{\approx \text{random}} & \boxed{\text{unchanged}} \end{tikzcd}$$

$\approx$ uncorrelated

Definitions



$H = \{h\}$ universal family of hash functions:

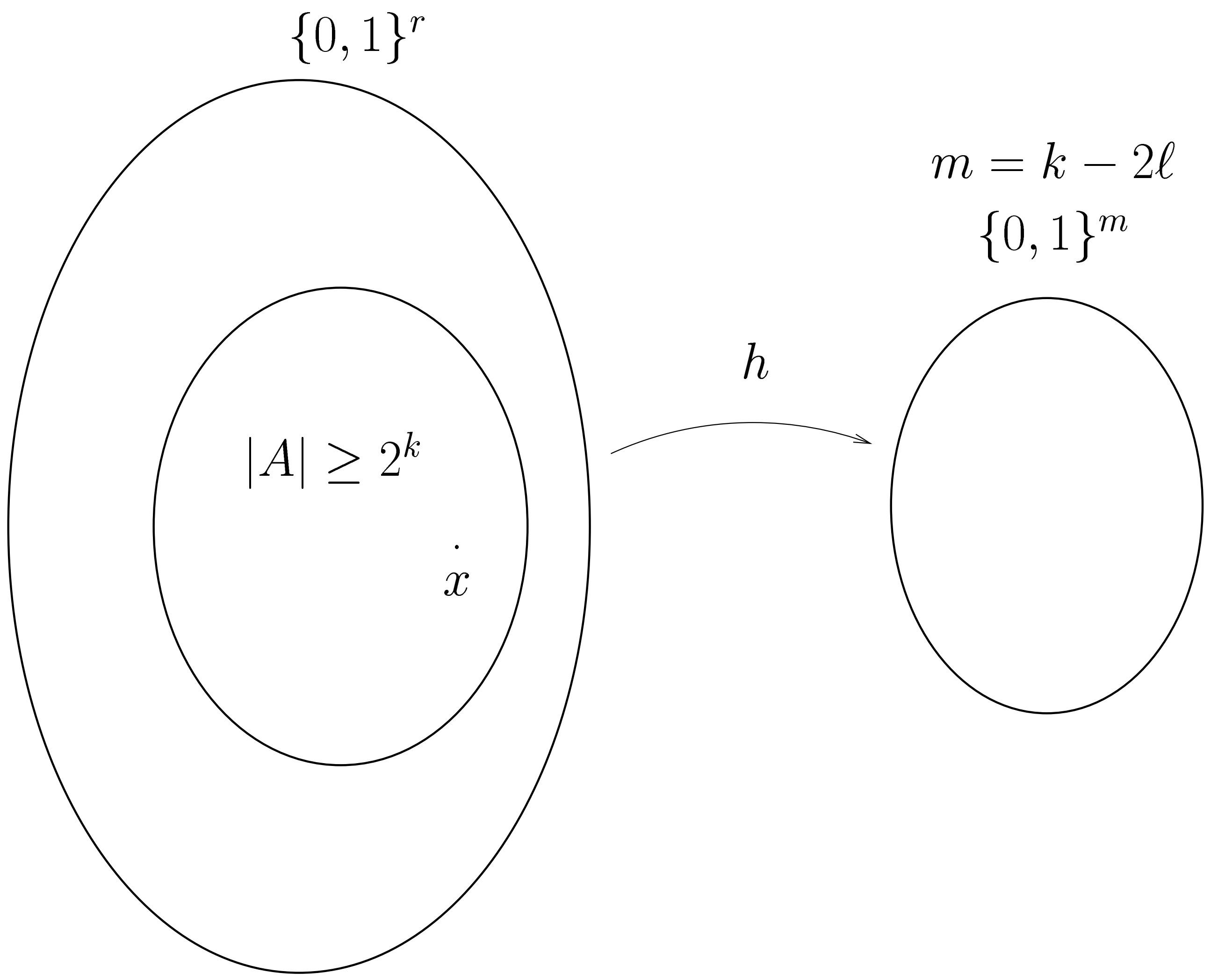
$$(\forall x \neq y) \Pr_{h \in_R H}[h(x) = h(y)] \leq 2^{-m}$$

Example: F = field on 2^r elements

$$h_a(x) = \text{last } m \text{ bits of } (a \cdot x)$$
$$|H| = 2^r$$

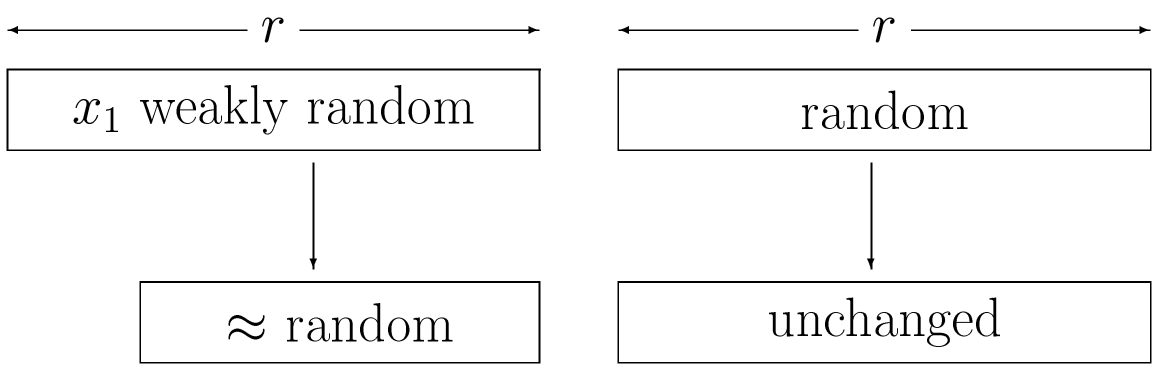
Large Random Catalyst

Entropy Smoothing Lemma [Impagliazzo,Levin,Luby]: For $x \in_R A$, $h \in_R H$, distribution of $h \circ h(x)$ is within $2^{-\ell}$ of uniform (variation, statistical, L_1)-distance.



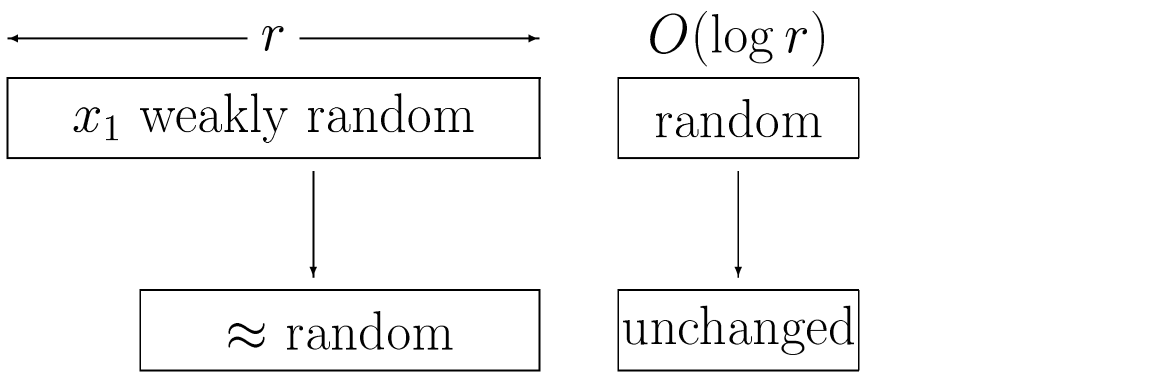
Extractor: Small random catalyst

Entropy Smoothing Lemma:



$\approx$ uncorrelated

Extractor [Nisan,Z]:



Much harder: can't use L_2 norm.

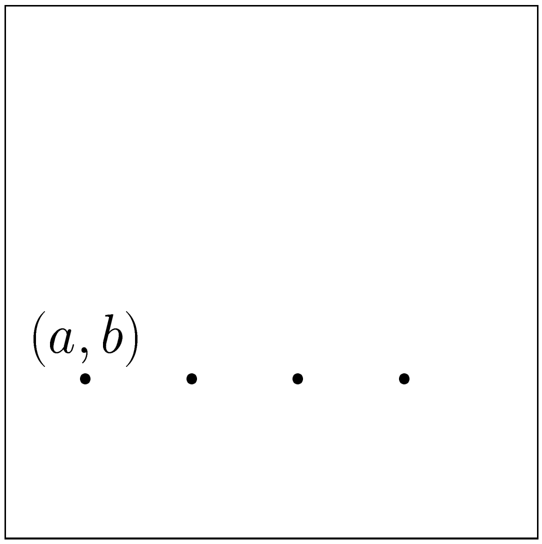
Extractor Constructions

Hash-based [Z;Nisan,Z] [WZ;SZ;SSZ;Z;Ta-S;RSW]

PRG-based [Trevisan] [RRV;ISW;TUZ]

Extractors from Reed-Muller Codes

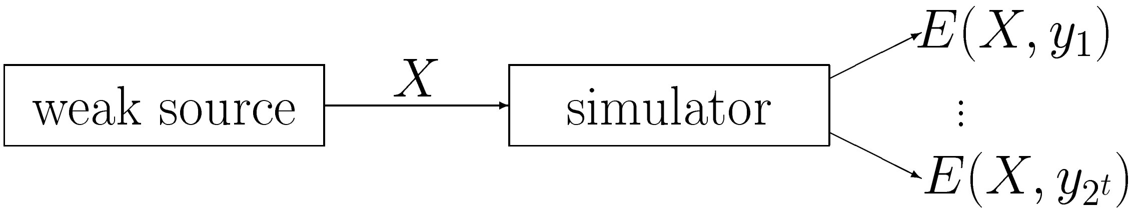
[Ta-Shma,Z,Safra]



$$\mathrm{BE}(\overbrace{\text{poly } p,\ \deg(p) < d}^{\text{weak source}};\ \overbrace{(a,b) \in \mathbb{Z}_q^2, v \in \{0,1\}^{\log q}}^{\text{truly random}})$$

$$= \langle p(a,b) \cdot v, p(a+1,b) \cdot v, \ldots, p(a+m-1,b) \cdot v \rangle$$

Extractor ⇒Simulate RP



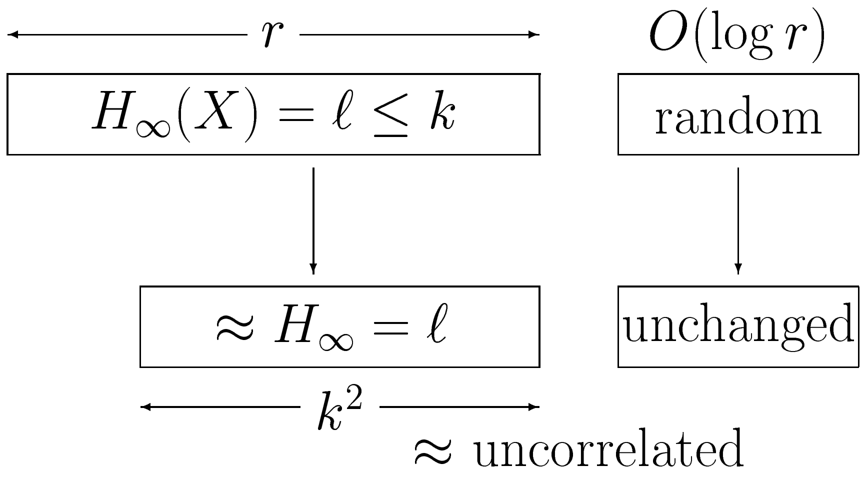
Cycle over all $2^t = 2^{O(\log r)} = \text{poly}(r)$ choices.

Other Applications of Extractors

- Pseudorandom generators for space-bounded computation [NZ]
- Random sampling using few random bits [Z]
- Expanders that beat the eigenvalue bound [WZ]
- Superconcentrators and non-blocking networks [WZ]
- Sorting and selecting in rounds [Pip]
- Time-Space tradeoffs [Sip]
- Unapproximability of clique [Z] and VC dimension [MU]
- Error-correcting codes with good list decoding properties [TZ]

$\vdots$

Condenser [Ta-Shma,Umans,Z]

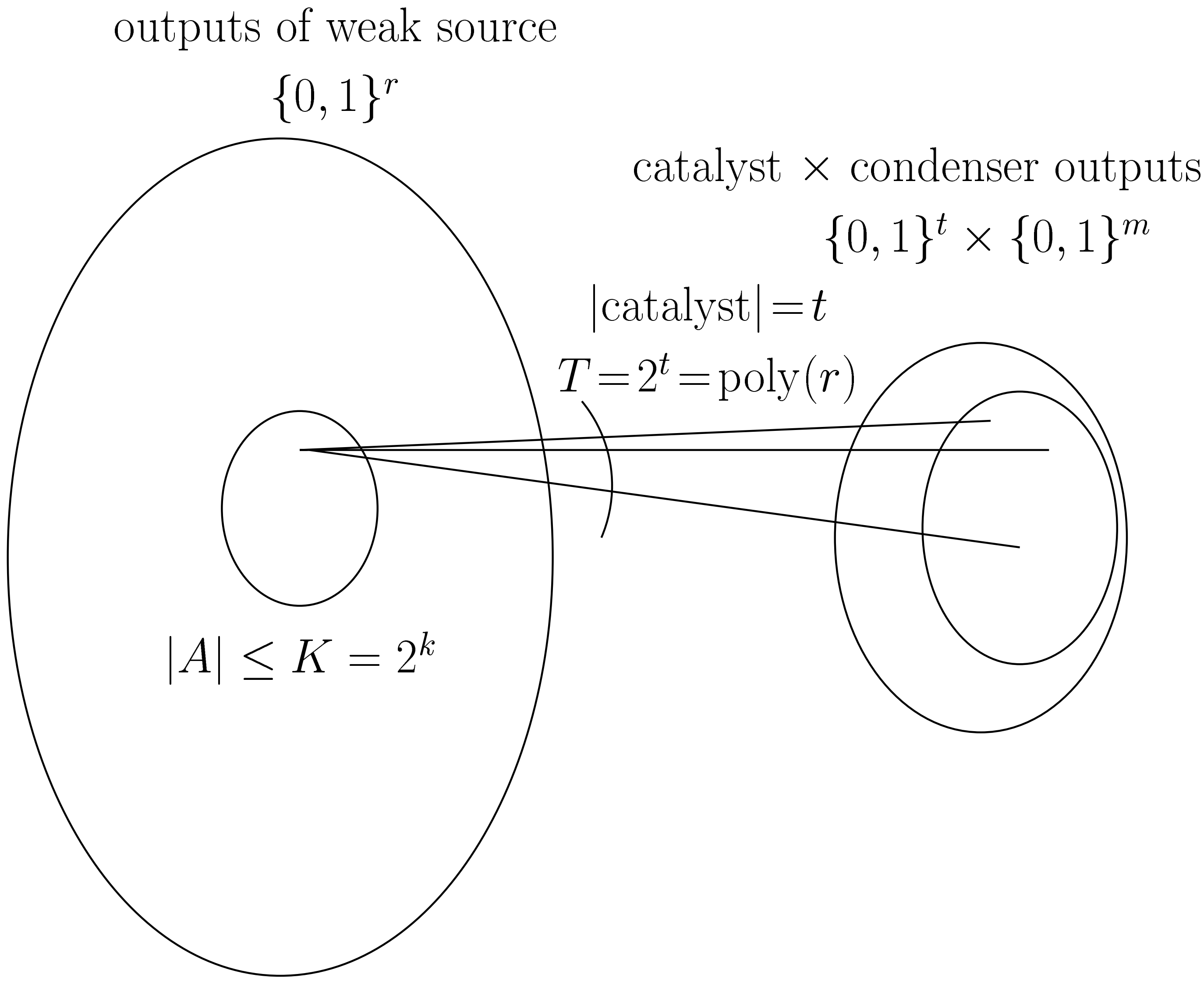


Useful notion:

Distribution close to one with large min-entropy.

Equivalent to condenser: $\forall A, |A| \le 2^k$, for most catalysts, condenser almost 1-1 on A.

Graph-Theoretic View of Condenser



Condenser property $\Rightarrow$expansion close to degree:

$$|A| \le K \Rightarrow |\Gamma(A)| \ge (1-\epsilon)T|A|$$

Constant-Degree Expander Graphs

[Capalbo, Reingold, Vadhan, Wigderson]

$$(\forall \epsilon > 0)(\exists \text{degree } D) \text{ expansion} \geq (1-\epsilon)D$$

Same number of vertices on each side (or differ by constant factor).

Beats eigenvalue bound.

Modify zig-zag product [Reingold, Vadhan, Wigderson]:
Use buffers to store min-entropy.

Pseudorandom Generators

Computers don't have access to truly random bits.

Pseudorandom generators (PRG's):

$$\underset{\text{random}}{01110100} \longrightarrow \boxed{\text{PRG}} \longrightarrow \underset{\text{“random” enough?}}{1101110011011000100111 1}$$

PRG deterministic; hope output is "random enough."

But $H(\text{output}) \leq H(\text{seed})$.

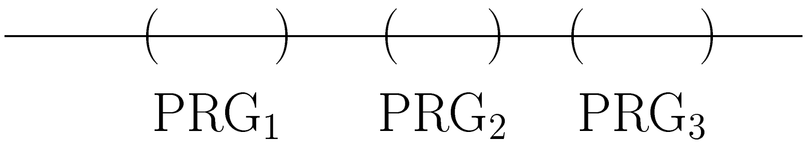
Pseudorandomness: Classical Approach
(e.g. [Knuth])

- Most popular PRG: Linear Congruential

$$x_{n+1} = (ax_n + b) \bmod m$$

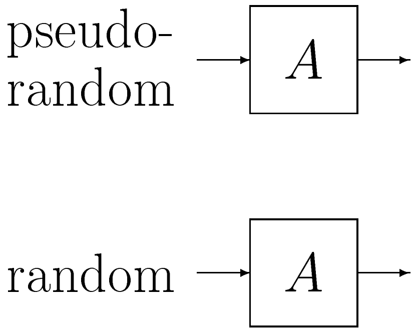
- Tests for pseudorandomness
  - long period
  - frequency tests, e.g. χ^2.
- Problems with PRG's
  - Tests ad hoc, e.g. $x_{n+1} = (x_n + 1) \bmod m$
  - More sophisticated uses of randomization $\Rightarrow$more errors with PRG's reported, for example [Johnson]:

95% confidence intervals



Pseudorandomness: Modern Approach

Goal:



$\approx$ same behavior
$\forall$ poly-time A

Almost all $f : \{0,1\}^{O(\log r)} \rightarrow \{0,1\}^r$ "fool" all poly-time machines.

$\exists$ PRG $\Rightarrow$NP $\neq$ P.

Short of Ideal PRG's

Fool large classes of poly-time A, e.g. space-bounded A.

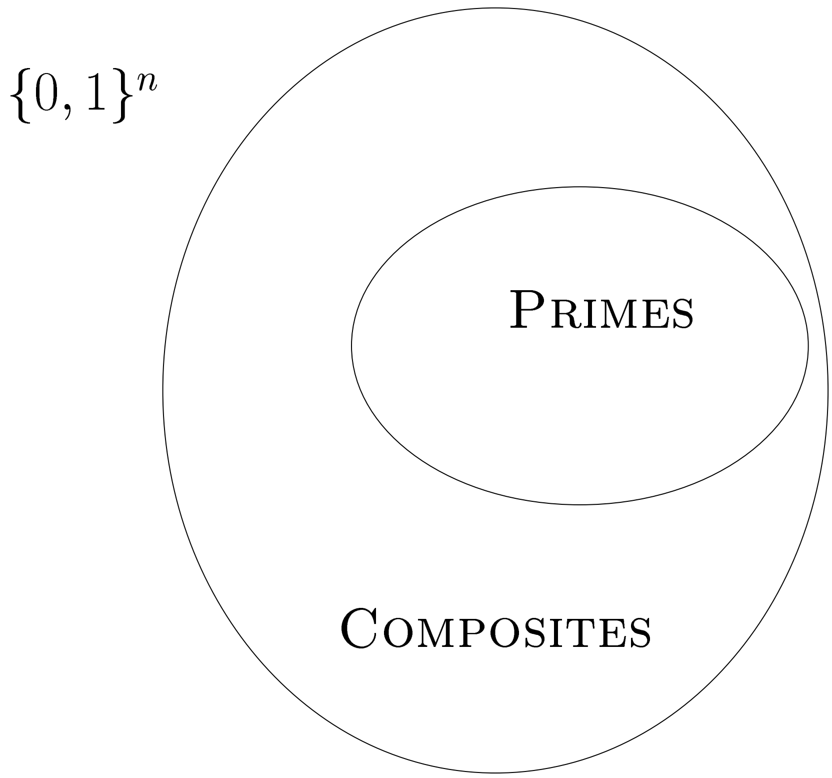
$\exists$ hard functions $\Rightarrow$general PRG's.

PRG Based on Extracting Randomness

Toy Example: Generate random n-bit prime.

Assume: Deterministic primality test.

Fact: $\Theta(1/n)$ fraction of n-bit numbers are prime.



Naive: Pick $x_1, x_2, \ldots \in_R \{0,1\}^n$ until some x_i prime.

E[# of random bits] $= O(n^2)$.

Using Fewer Random Bits

Improvement idea: If x_1 composite, extract randomness left in x_1.

Can't do deterministically (without specialized knowledge about composites).

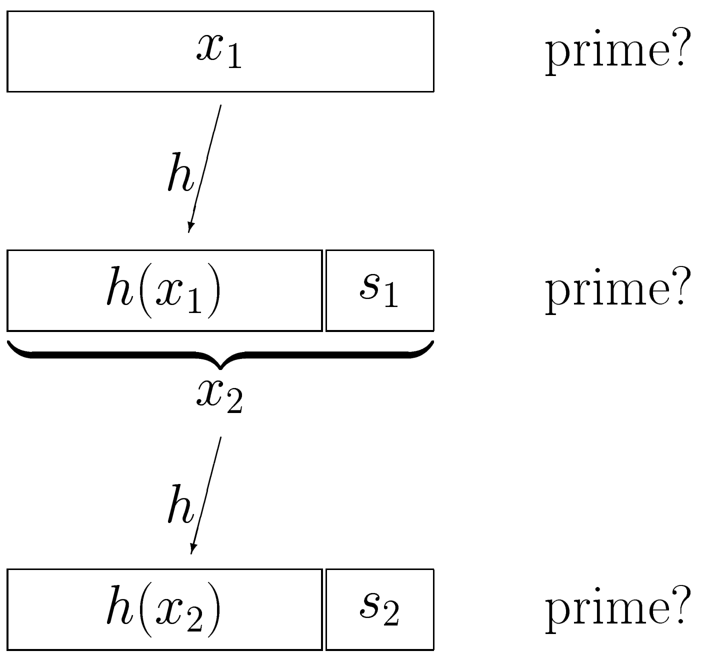
Method: Add random catalyst.

$$\begin{tikzcd} \boxed{x_1 \text{ weakly random}} \arrow[d] & \boxed{\text{random } h} \arrow[d] \\ \boxed{\approx \text{random}} & \boxed{h} \end{tikzcd}$$

$\approx$ uncorrelated

Recycling Randomness [Impagliazzo,Z]

Pick $h \in_R H$.



E[# of random bits used] $= O(n \log n)$.

Theorem: Method works whenever information used about each random number is small.

Partial explanation of success of practical PRG's

- Linear congruential, shift register generators are hash functions.

General PRG's under assumptions

Theorem: $\exists$ PRG's $\iff$ $\exists$ 1-way functions [Hastad,Impagliazzo,Levin,Luby]

1-way function:

- f easy: computable in polynomial time.
- f^{-1} hard on average.

Essential for modern cryptography.

Conjectured example: multiplication.

$\exists$ 1-way function $\Rightarrow$NP $\neq$ P.

PRG $\Rightarrow$1-way function

A PRG $G : \{0,1\}^{r^\epsilon} \to \{0,1\}^r$ is a 1-way function:

- G easy.
- Invert G $\Rightarrow$distinguish $G(U_{r^\epsilon})$ from U_r.

1-way function ⇒PRG

First show 1-way function ⇒pseudo-entropy generator.

Computational entropy $H_{comp}(X) \geq k$ iff $\exists X'$

- $H(X') \geq k$
- X and X' computationally indistinguishable:

$$\underset{\text{length } \ell}{01110100} \longrightarrow \boxed{\text{PEG}} \longrightarrow \underset{H_{comp} \geq \ell + 1}{11011100110110001001111}$$

Pseudo-entropy generator ⇒PRG

Entropy ⇒Renyi-entropy:

Concatenate independent copies:

$$X_1 \circ \ldots \circ X_s \approx Y$$

$$\text{where } H_2(Y) \geq s \cdot H(X_i) - s^{5/6}$$

Extracting from pseudo-entropy generator G:

$$\begin{tikzcd}
\boxed{x_1 \circ \ldots \circ x_s} \arrow[d] & \boxed{\text{random } h} \arrow[d] \\
\boxed{y = G(x_1) \circ \ldots G(x_s)} \arrow[d] & \boxed{h} \arrow[d] \\
\boxed{h(y)} & \boxed{h}
\end{tikzcd}$$

Entropy Function as Tool in Complexity

$H(XY) = H(X) + H(Y|X)$.

Lower bounds:

- Parallel computation [Boppana]
- Communication complexity $\iff$ monotone circuit depth [Karchmer, Wigderson]
- Communication complexity [Ponzio,Radhakrishnan,Venkatesh]
- Quantum communication complexity [Klauck, Nayak, Ta-Shma, Z]
- Branching programs [Beame, Jayram, Saks]

Other areas:

- Sorting partial orders [Kahn,Kim]
- Parellel repetition for interactive proofs [Raz]
- Small part of condenser construction [Ta-Shma,Umans,Z]

Min-entropy, Renyi-entropy don't behave as nicely.

Conclusions

Utility of min-entropy:

Computational Entropy

General PRG

Extractors

Min-Entropy

Extracting PRG

Condensers

Expanders