8

Pseudorandom Generators

Great Ideas in Theoretical Computer Science
Saarland University, Summer 2014

Randomness is one of the fundamental computational resources and appears everywhere. In computer science, we study randomness due to the following reasons: First, computers are used to simulate various random processes in Physics, Chemistry and Biology. Second, randomness is widely used in Cryptography, and a few techniques of algorithm design (e.g. sampling, random walks) rely on randomness. Third, even though efficient deterministic algorithms are unknown for many problems, simple, elegant and fast randomized algorithms are known and easy to analyze. For many problems (e.g. primality testing), usually we have efficient randomized algorithms at first before we obtain deterministic polynomial-time algorithms eventually. This leads the question of *derandomization*, i.e., if we have a general way to derandomize randomized algorithms without significantly increasing their runtime.

The answer to these questions have lead to the study of *pseudorandomness*. In today's lecture, we will address the following three questions: (1) What is randomness? What are the possible ways to define randomness? (2) How do computer programs generate random numbers? Are these numbers truly random? (3) What the relationship between randomness and other disciplines of Computer Science and Mathematics?

8.1 What is Randomness?

What are *random* binary strings? The best way to think about perfect randomness is as an arbitrary long sequence of binary strings, where every 0 or 1 appears equally, i.e. every 0/1-bit appears with the same probability (50%) in a random order. For instance, the string

$$000000000100000000001000010000$$

is not random as the number of 0s is much more than the number of 1s.

How about this sequence?

$$00000000001111111111$$

This sequence contains the same number of 0s and 1s. However most people think that it is not random, as the occurrences of 0s and 1s are in a very unbalanced way. These intuitions can be formalized in several seemingly different ways to define randomness.

The first formulation is to use the notion of the *entropy*, which is due to Shannon's Information Theory [6]. Information theory focuses on the distributions that are not perfect random, and use entropy to evaluate the amount of information contained in a distribution. In this setting, perfect randomness is an extreme case, and it corresponds to the distribution with the maximum entropy.

Definition 8.1 (Entropy)**.** *Let X be a random variable. Then the entropy $\mathbf{H}(X)$ of X is defined as*

$$\mathbf{H}(X) \triangleq \mathbf{E}\left[-\log\left(\mathbf{Pr}[\,X = x\,]\right)\right].$$

The second theory is due to A. N. Kolmogorov [4]. Kolmogorov complexity theory measures the complexity of objects in terms of the shortest program (for a fixed universal program) that generates the object. Here perfect randomness appears as an extreme case since no shorter program can generate the perfect random string. In contrast to the information theory in which the entropy measures the randomness of a distribution, following the Kolmogorov complexity theory one can study randomness of a single string.

Definition 8.2 (Kolmogorov Complexity)**.** *Let $\ell(x)$ be the length of string x. The Kolmogorov Complexity $K_U(x)$ of a string x with respect to a universal computer U is defined as*

$$K_U(x) \triangleq \min_{p:U(p)=x} \ell(p),$$

the minimum length over all programs that print x and halts. Thus $K_U(x)$ is the shortest description length of x over all descriptions interpreted by computer U.

The third theory is due to M. Blum and S. Micali [1] in 1982, and independent by A. Yao [7] in the same year. They define randomness with respect to observers (computational models), and suggest to view distributions as equal if they cannot be distinguished by efficient observers. To motivate this definition, let us look at one example.

Example 8.3. [1] *Alice and Bob play "head or tail" in one of the following four ways. In all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome before the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability $1/2$. In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is determined in principle by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability $1/2$. The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate information on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob's recording equipment is directly connected to a powerful computer programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.*

This example indicates that proper definitions of randomness depend on the power of

[1] This example is from "A Primer on Pseudorandom Generators" by Oded Goldreich, 2010.

observers: One string that looks random for one observer may be not random for more powerful observers. In Computer Science, observers with different powers correspond to algorithms under different time/space constraints (e.g., polynomial-time, exponential-time, or polynomial-space). To capture this intuition, we define the notion of the *computational indistinguishable* which refers to pairs of the distributions which cannot be distinguished by any polynomial-time algorithm.

Definition 8.4 (probability ensembles)**.** *A probability ensemble $\mathcal{X}$ is a family $\mathcal{X} = \{X_n\}_{n\geq 1}$ such that X_n is a probability distribution on some finite domain.*

Definition 8.5 (Computational Indistinguishability, [2])**.** *Let $\mathcal{D}$ and $\mathcal{E}$ be probability ensembles. The success probability of a randomized algorithm A for distinguishing $\mathcal{D}$ and $\mathcal{E}$ is*

$$sp_n(A) = \Big|\mathbf{Pr}[\,A(X) = 1\,] - \mathbf{Pr}[\,A(Y) = 1\,]\Big|,$$

where X has distribution $\mathcal{D}$ and Y has distribution $\mathcal{E}$. Distributions $\mathcal{D}$ and $\mathcal{E}$ are called computationally indistinguishable, if for any probabilistic polynomial-time algorithm A, for any positive polynomial $p : \mathbb{N} \mapsto \mathbb{N}$, and for all sufficiently large n's, it holds that $sp_n(A) < 1/p(n)$.

We use $\mathcal{D} \sim^c \mathcal{E}$ to express that $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.

8.2 Pseudorandom Generators

Pseudorandom generators are *deterministic* polynomial-time algorithms which stretch *short random seeds* into longer sequences which "look" random. When we design a pseudorandom generator, there are a few factors taken into account:

The first is the *efficiency*: The generator should be efficient, i.e. the pseudorandom generators should produce the output (pseudorandom sequences) in polynomial-time. The main reason for the efficiency requirement is that, for many practical applications, pseudorandom generators are used as subroutines of other algorithms, and the use of pseudorandom generators should significantly increase the runtime of the original algorithm.

The second aspect is the *stretching*, i.e., a PRG is required to stretch its input seed to a longer output sequence. Specifically, a PRG stretches an n-bit input into an $\ell(n)$-bit long output, where $\ell(n) > n$. The function ℓ is called the *stretching function*.

The third aspect is the *pseudorandomness*. A PRG's output has to look random to any efficient observer, i.e., any procedure fails to distinguish the output of a PRG (on a random seed) from a truly random sequence of the same length in polynomial-time. For instance, a procedure could count the number of 0s and 1s and any pseudorandom generator need output almost the same number of 0s and 1s.

Now we give a formal definition of pseudorandom generators.

Definition 8.6 (Pseudorandom Generators)**.** *A deterministic polynomial-time algorithm G is called a pseudorandom generator if there exists a stretching function $\ell : \mathbb{N} \mapsto \mathbb{N}$, such that the following two probability ensembles, denoted $\{\mathcal{G}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{U}_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable:*

1. *Distribution $\mathcal{G}_n$ is defined as the output of G whose length is $\ell(n)$ on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution $\mathcal{U}_{\ell(n)}$ is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$, where $\ell(n) > n$.*

That is, we require that for any probabilistic polynomial-time algorithm A, for any positive polynomial $p : \mathbb{N} \mapsto \mathbb{N}$, and for all sufficiently large n's, it holds that

$$\left|\mathbf{Pr}[\, A(G(\mathcal{U}_n)) = 1\,] - \mathbf{Pr}[\, A\left(\mathcal{U}_{\ell(n)}\right) = 1\,]\right| < \frac{1}{p(n)}.$$

By Definition 8.6, pseudorandomness is defined in terms of its observer. It is the distribution which cannot be told apart from the uniform distribution by any polynomial-time observer. We remark that this pseudorandom sequence may be distinguishable from truly random ones by one observe with more (or even unbounded) computational power. For instance, the pseudorandom sequence that cannot be distinguished from truly random ones by any polynomial-time algorithm could be distinguished from truly random ones by an exponential-time algorithm. Hence, pseudorandomness is subjective to the abilities of the observer.

The next very interesting result shows that, once we have a PRG to generate a simple pseudorandom bit, we can use the same PRG to generate pseudorandom bits with arbitrary length.

Theorem 8.7 (amplification of stretch function)**.** *Suppose we have a PRG G with a stretch function $\ell(n) = n + 1$, then for every polynomial $\ell(n) > n$ there exists a PRG with stretch function $\ell(n)$.*

Proof. Let G be a PRG with a stretching function $n+1$. We construct a PRG G^i with stretching function $\ell(n) = n + i$. Define

$$G^i(\mathcal{U}_n) = \begin{cases} G\left(\mathcal{U}_n\right) & i = 1 \\ G^{i-1}\left(G\left(\mathcal{U}_n\right)_{1\cdots n}\right) \circ G\left(\mathcal{U}_n\right)_{n+1} & i > 1 \end{cases}$$

where $G(\mathcal{U}_n)_i$ is the $i$bit of $G(X_n)$, $G(\mathcal{U}_n)_{i\cdots j}$ is the substring of $G(\mathcal{U}_n)$ from the ith bit up to the jth bit, and $\circ$ represents the concatenation operator between two strings.

We define a sequence of the distributions as follows:

$$\mathcal{D}_0 : \mathcal{U}_n \circ \mathcal{U}_m, \;\; \mathcal{D}_1 : G(\mathcal{U}_n) \circ \mathcal{U}_{m-1}, \;\cdots,\; \mathcal{D}_m : G^m(\mathcal{U}_n).$$

Our goal is prove that G^m is a PRG, i.e., there is no efficient algorithm to distinguish $G^m(\mathcal{U}_n) = \mathcal{D}_m$ from $\mathcal{U}_{m+n} = \mathcal{D}_0$. Assume for contradiction that there is an algorithm A and polynomial p, such that

$$|\mathbf{Pr}[\, A(\mathcal{D}_m) = 1\,] - \mathbf{Pr}[\, A(\mathcal{D}_0) = 1\,]| \geq \frac{1}{p(n)}.$$

Because

$$\begin{aligned}\Big|\mathbf{Pr}[\,A(\mathcal{D}_m)=1\,]-\mathbf{Pr}[\,A(\mathcal{D}_0)=1\,]\Big| &= \left|\sum_{i=1}^{m}(\mathbf{Pr}[\,A(\mathcal{D}_i)=1\,]-\mathbf{Pr}[\,A(\mathcal{D}_{i-1})=1\,])\right| \\ &\le \sum_{i=1}^{m}\Big|\mathbf{Pr}[\,A(\mathcal{D}_i)=1\,]-\mathbf{Pr}[\,A(\mathcal{D}_{i-1})=1\,]\Big|,\end{aligned}$$

there is at least one index i $(1 \le i \le m)$ such that

$$\Big|\mathbf{Pr}[\,A(\mathcal{D}_i)=1\,]-\mathbf{Pr}[\,A(\mathcal{D}_{i-1})=1\,]\Big| > \frac{1}{m\cdot p(n)},$$

which contradicts to the assumption that $\mathcal{D}_i \sim^c \mathcal{D}_{i+1}$. □

8.3 Pseudorandom Generators versus One-Way Functions

Randomness and hardness seem to be very difficult aspects in computation. In this section we will show that they are strongly related. To formalize the notion of the computational difficulty, we introduce one-way functions. One-way functions are a family of functions which are easy to compute and hard to compute the inverted. The formal definition is as follows:

Definition 8.8 (One-way Functions)**.** *A function $f : \{0,1\}^\star \mapsto \{0,1\}^\star$ is one-way if f satisfies the following two conditions:*

1. *There exists a polynomial-time algorithm A to compute f, i.e. $\forall x : A(x) = f(x)$.*
2. *For all probabilistic polynomial-time algorithms A', polynomials $p(\cdot)$ and sufficiently large n's, it holds that*

$$\mathbf{Pr}[\,A'(f(\mathcal{U}_n)) = f^{-1}\circ f(\mathcal{U}_n)\,] < \frac{1}{p(n)},$$

i.e., there is no probabilistic polynomial-time algorithm A' such that, give $g(x)$ with a random x, A' obtains x with non-negligible probability.

Håstad, Impagliazzo, Levin and Luby [3] in 1993 proved the following definitive theorem: Starting with any one-way function, one can construct a pseudorandom generator.

Theorem 8.9 ([3])**.** *Pseudorandom generators exist if and only if one-way functions exist.*

Proof. We only show that the existence of PRGs implies the existence of one-way functions, and the proof of the other direction is more complicated.

Let $G : \{0,1\}^n \mapsto \{0,1\}^{2n}$ be a PRG. Consider $f(x,y) \triangleq G(x)$, where $|x| = |y|$. We assume for contradiction that f is not a one-way function, i.e., there is a probabilistic polynomial-time algorithm A' to invert f with probability greater than $1/p(n)$ for some polynomial p. Now we design an algorithm D to distinguish the distribution $G(\mathcal{U}_n)$ from $\mathcal{U}_{2n}$. The algorithm is described in Algorithm 8.1.

Algorithm 8.1 Distinguisher D

1: **Input:** $\alpha, \alpha \in \{0,1\}^{2n}$
2: $xy \leftarrow A'(\alpha)$
3: **if** $f(xy) = \alpha$ **then return 1**
4: **else return 0**

It is easy to see that

$$\begin{aligned}\mathbf{Pr}\left[\, D(G(\mathcal{U}_n)) = 1 \,\right] &= \mathbf{Pr}\left[\, D(f(\mathcal{U}_n)) = 1 \,\right] \\ &= \mathbf{Pr}\left[\, f(A'(f(\mathcal{U}_n))) = f(U_n) \,\right] \\ &= \mathbf{Pr}\left[\, A'(f(\mathcal{U}_n)) = f^{-1} \circ f(\mathcal{U}_n) \,\right] \\ &> \frac{1}{p(n)},\end{aligned}$$

where the second equality is by the algorithm description, and the last inequality is by the contradiction hypothesis. On the other hand, there are at most 2^n strings of length $2n$ which have a preimage under G (and so under f). Hence, a uniformly selected random string of length $2n$ has a preimage under f with probability at most $2^n/2^{2n}$. This implies that

$$\begin{aligned}\mathbf{Pr}\left[\, D(\mathcal{U}_{2n}) = 1 \,\right] &= \mathbf{Pr}\left[\, f(A'(\mathcal{U}_{2n})) = \mathcal{U}_{2n} \,\right] \\ &\leq \mathbf{Pr}\left[\, \mathcal{U}_{2n} \text{ is in the image of } f \,\right] \\ &\leq \frac{2^n}{2^{2n}} = 2^{-n}.\end{aligned}$$

Therefore, we have that

$$\mathbf{Pr}\left[\, D(G(\mathcal{U}_n)) = 1 \,\right] - \mathbf{Pr}\left[\, D(\mathcal{U}_{2n}) = 1 \,\right] > \frac{1}{p(n)} - \frac{1}{2^n} \geq \frac{1}{q(n)}$$

for some polynomial q. This contradicts to the assumption that G is a PRG. □

There are a few problems that seem to be one-way in practice and are conjectured to be one-way. The following are two typical candidates.

Problem 8.10 (Factoring Problem)**.** *Let $f(p,q) = pq$. There is no known polynomial-time algorithm that on input pq can produce p and q on average for randomly chosen pairs of primes p and q.*

Problem 8.11 (Discrete Logarithm Problem)**.** *Given a prime modulus p and a generator $g \in \mathbb{Z}_p^*$, for $y = g^x \mod p$ find the index x.*

8.4 PRGs for Space-Bounded Computation

So far we discussed the PRGs for polynomial-time algorithms, and constructing such PRGs relies on existence of one-way functions. In this section we show that, if we weaken the

abilities of the algorithms that PRGs need to fool, e.g. algorithms with bounded-space, then such PRGs can be constructed without assuming existence of one-way functions. The goal of this section is to prove the following theorem:

Theorem 8.12 ([5]). *For any R and S there exists an (explicitly given) pseudorandom generator which converts a random seed of length $O(S \log R)$ to R bits which look random to any algorithm running in space S.*

Definition 8.13. *Let M be a randomized algorithm that on input w requires $\ell(|w|)$ random bits. The family $\{G_n\}_{n=1}^{\infty}$ of functions $(G_n : \{0,1\}^{s(n)} \to \{0,1\}^{\ell(n)})$ is an $\varepsilon-$PRG for M if for any $w \in \{0,1\}^*$, it holds that*

$$\left|\mathbf{Pr}_{r\in\{0,1\}^{\ell(|w|)}}\left[M(w,r)=1\right] - \mathbf{Pr}_{z\in\{0,1\}^{s(|w|)}}\left[M\left(w, G_{|w|}(z)\right)=1\right]\right| < \varepsilon.$$

Randomized Logspace TM. For any randomized Turing machine/algorithm M, we can write the execution of M as a table, and every row in the table corresponds a step, and consists of the configuration in that step. We call such a table the *computation tableau* of M on a specific input. For any Turing machine M that runs in S space, the runtime of M is upper bounded by 2^S (otherwise the same configuration will appear in the computation tableau at least twice, and M goes into a loop). Figure 8.1 shows one example of the computation tableau of a Turing machine whose space is bounded by S.

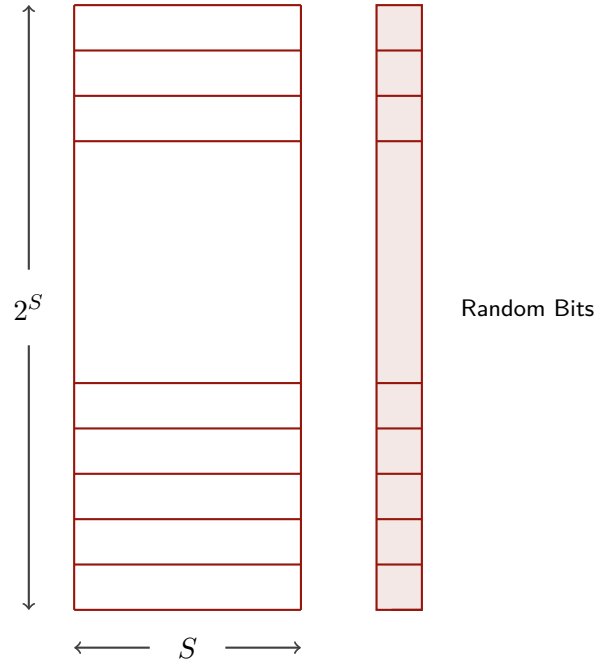

Figure 8.1: The computation tableau of a space S-bounded TM. Since the space is M, the runtime of this machine is upper bounded by 2^S. If the machine is randomized, then there is a random bit associated with each step.

Using Non-Perfect Random Bits. When M is a randomized machine, then M need at most one random bit every step, and the number of random bits required for M is at most 2^S. For simplicity we assume that the runtime of M is exactly 2^S. We divide the tableau into two halves with each half requiring random strings r_1 and r_2 respectively, each of length $r = 2^S/2$. If r_1 and r_2 are chosen independently, then by definition M outputs the correct answer with high probability. Now we study the performance of M if we choose r_1 and r_2 in a manner that their behavior is not significantly different from the case when they are chosen independently.

Formally, we assume that A_1 and A_2 are the upper and lower halves of the computation tableau of M, and x_1 and x_2 are independently chosen random strings, see Figure 8.2. Moreover, algorithms A_1 and A_2 are of the following form:

- Algorithm A_1 takes an input x_1 of length r, and outputs a string b_1 of length c.
- Algorithm A_2 takes as input the output b_1 of A_1 and another string x_2 of length r, and outputs a string b_2 of length c.

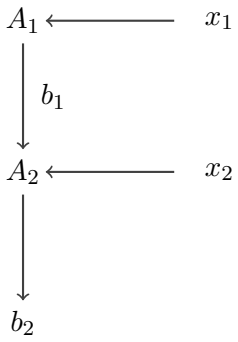

Figure 8.2: Algorithms A_1, A_2 with inputs from the generator.

What we are in search of is a generator that supplies strings x_1 and x_2 in a fashion better than choosing them independently. For simplicity, given a function g, we use $g^\ell(z)$ and $g^r(z)$ to express the left half and the right half of the string $g(z)$, i.e., $g(z) = g^\ell(z) \circ g^r(z)$ and $|g^\ell(z)| = |g^r(z)|$. We are interested in the performance of M when r_1 and r_2 are replaced by $g^\ell(z)$ and $g^r(z)$, see Figure 8.3.

The following definition summarizes the PRGs that we need:

Definition 8.14. *A function $g : \{0,1\}^t \to \{0,1\}^r \times \{0,1\}^r$ is defined to be a ε-generator for communication c if for all functions $A_1 : \{0,1\}^r \mapsto \{0,1\}^c$ and $A_2 : \{0,1\}^c \times \{0,1\}^r \mapsto \{0,1\}^c$, it holds for all $b \in \{0,1\}^c$ that*

$$\forall b \qquad \left| \mathbf{Pr}_{x_1,x_2 \in \{0,1\}^r} \left[A_2(A_1(x_1), x_2) = b \right] - \mathbf{Pr}_{z \in \{0,1\}^t} \left[A_2(A_1(g^\ell(z)), g^r(z)) = b \right] \right| < \varepsilon.$$

*For simplicity, we call a 2^{-c}-generator for communication c a c-*generator.

We assume the existence of a c-generator of the following form, and we will give a construction of c-generators at the end of the lecture.

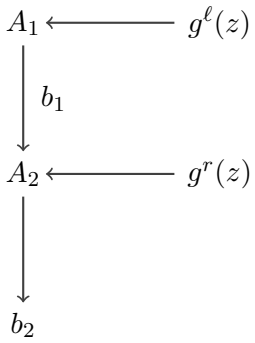

Figure 8.3: Algorithms A_1, A_2 with inputs from the generator.

Lemma 8.15. *There exists a constant $k > 0$ such that for all r, c, there exists a polynomial-time computable c-generator g of the form $g : \{0,1\}^{r+kc} \mapsto \{0,1\}^r \times \{0,1\}^r$.*

Construction of Space-Bounded PRGs. Now we break the tableau into several components, called $A_1, A_2, \cdots, A_{2^S}$, and let the output of g be the random strings for every pair of consecutive components (A_{2i-1} and A_{2i}) such that their behavior is not significantly different from using truly random bits, see Figure 8.4. Moreover, we expect that every application of g causes error at most $1/2^{2S}$. Because we invoke g at most 2^{S-1} times, so the total error is at most $2^{S-1} \cdot 1/2^{2S}$.

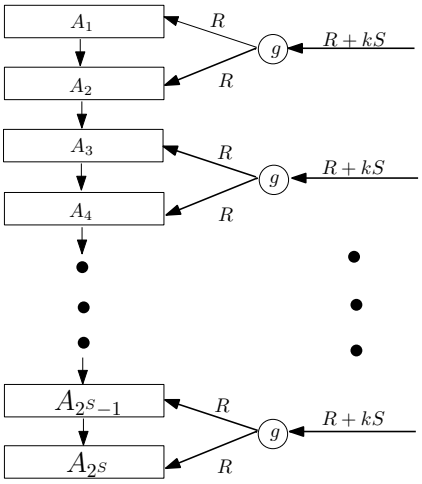

Figure 8.4: The first level of the recursive construction.

Notice that there are 2^{S-1} components in the framework above, each requiring $R + kS$ random bits. We use the same approach to reduce the number of random bits required by every pair of components from $R + kS$ each to $R + 2kS$ total. We perform this operation recursively till there is only one component left, see Figure 8.5.

Formally let $g_i : \{0,1\}^{R+ikS} \to \{0,1\}^{R+(i-1)kS} \times \{0,1\}^{R+(i-1)kS}$ be an S-generator for

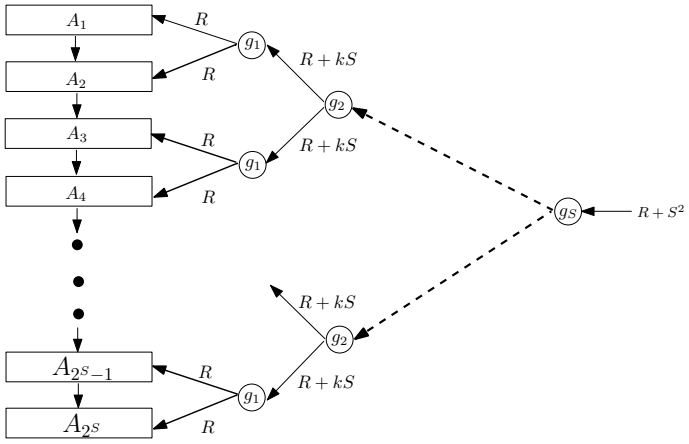

Figure 8.5: Recursive construction of G_S.

$1 \le i \le S$. Define $G_i : \{0,1\}^{R+ikS} \to \{0,1\}^{2^i \cdot R}$ inductively as follows:

$$G_i(z) = \begin{cases} z & i = 0 \\ G_{i-1}\left(g_i^\ell(z)\right) \circ G_{i-1}\left(g_i^r(z)\right) & i > 0 \end{cases}$$

Lemma 8.16. *The PRG G_S runs in space $O\left(R + kS^2\right)$.*

Lemma 8.17. *For all space S-bounded TM M, G_S is a 2^{-S}-generator for M.*

Proof. Since each application of g_i incurs an error of at most $1/2^{2S}$, and there are $2^{S-1} + 2^{S-2} + \cdots + 2 + 1 = 2^S - 1$ applications of g_i, therefore by the union bound the error probability is at most $(2^S - 1)/2^{2S} \le 1/2^S$. □

Proof of Lemma 8.15. Now we start to prove Lemma 8.15. The following result will be used.

Lemma 8.18. *Let $G = (V, E)$ be a d-regular graph with spectral expansion λ. Then for any subsets $S, T \subseteq V$ we have*

$$\left| \frac{|E(S,T)|}{|E|} - \frac{|S|}{|V|} \cdot \frac{|T|}{|V|} \right| \le \lambda \cdot \sqrt{\frac{|S|}{|V|} \cdot \frac{|T|}{|V|}}.$$

Proof. We first define the *double cover* of graph G. For any non-bipartite graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, the double cover $H = (L \cup R, E')$ of G is an bipartite graph with $L = \{\ell_1, \ldots, \ell_n\}$ and $R = \{r_1, \ldots, r_n\}$, such that $\{i, j\} \in E$ if and only if $\{\ell_i, r_j\} \in E'$ and $\{\ell_j, r_i\} \in E'$.

The statement of the lemma holds by applying the Expander Mixing Lemma on the double covering of G. □

Proof of Lemma 8.15. Let $G = (V, E)$ be a $d = 2^{6c}$-regular Ramanujan expander on $|V| = 2^r$ vertices. The generator $g : \{0,1\}^{r+6c} \to \{0,1\}^r \times \{0,1\}^r$ works as follows: On input $z = (x, i) \in \{0,1\}^r \times \{0,1\}^d$, output $\left(g^\ell(z), g^r(z)\right) = (x, y)$ where y is the vertex reached by taking the ith edge out of x.

Let b be any output of the algorithms (A_1, A_2). Fix b. For any $b' \in \{0,1\}^c$, define

$$S_{b'} = \{x \in \{0,1\}^r | A_1(x) = b'\},$$

$$T_{b'} = \{x \in \{0,1\}^r | A_2(b', x) = b\}.$$

Thus for truly random strings x_1 and x_2, the probability that algorithms (A_1, A_2) outputs b is expressed by

$$\begin{aligned}\mathbf{Pr}_{x_1,x_2}\left[A_2(A_1(x_1), x_2) = b\right] &= \sum_{b' \in \{0,1\}^c} \mathbf{Pr}\left[x_1 \in S_{b'} \wedge x_2 \in T_{b'}\right] \\ &= \sum_{b' \in \{0,1\}^c} \frac{|S_{b'}|}{|V|} \cdot \frac{|T_{b'}|}{|V|}.\end{aligned}$$

On the other hand, when using the output of g to instead truly random bits, the probability that (A_1, A_2) outputs b is

$$\begin{aligned}\mathbf{Pr}_{z \in \{0,1\}^{r+d}}\left[A_2(A_1(g^\ell(z)), g^r(z)) = b\right] &= \sum_{b' \in \{0,1\}^c} \mathbf{Pr}_{x,i}\left[x \in S_{b'} \wedge \ i\text{th edge out of } x \text{ leads to } T_{b'}\right] \\ &= \sum_{b' \in \{0,1\}^c} \frac{|E(S_{b'}, T_{b'})|}{|E|}.\end{aligned}$$

Therefore

$$\begin{aligned}&\left|\mathbf{Pr}_{x_1,x_2}\left[A_2(A_1(x_1), x_2) = b\right] - \mathbf{Pr}_{z \in \{0,1\}^{r+d}}\left[A_2(A_1(g^\ell(z)), g^r(z)) = b\right]\right| \\ =& \left|\sum_{b' \in \{0,1\}^c} \left(\frac{|S_{b'}|}{|V|} \cdot \frac{|T_{b'}|}{|V|} - \frac{|E(S_{b'}, T_{b'})|}{|E|}\right)\right| \\ \leq& \sum_{b' \in \{0,1\}^c} \left|\frac{|S_{b'}|}{|V|} \cdot \frac{|T_{b'}|}{|V|} - \frac{|E(S_{b'}, T_{b'})|}{|E|}\right|.\end{aligned}$$

Because $S_{b'}$ and $T_{b'}$ are subsets of V, by Lemma 8.18 we know that

$$\left|\frac{|S_{b'}|}{|V|} \cdot \frac{|T_{b'}|}{|V|} - \frac{|E(S_{b'}, T_{b'})|}{|E|}\right| \leq \lambda\sqrt{\frac{|S_{b'}|}{|V|} \cdot \frac{|T_{b'}|}{|V|}} \leq \lambda,$$

where λ is the spectral expansion of G and satisfies $\lambda \leq 2 \cdot \sqrt{d-1}/d$. Therefore

$$\begin{aligned}
&\left|\mathbf{Pr}_{x_1,x_2}\left[A_2(A_1(x_1),x_2)=b\right]-\mathbf{Pr}_{z\in\{0,1\}^{r+d}}\left[A_2(A_1(g^\ell(z)),g^r(z))=b\right]\right| \\
\leq &\sum_{b'\in\{0,1\}^c}\lambda \\
\leq &\sum_{b'\in\{0,1\}^c}\frac{2\cdot\sqrt{d-1}}{d} \\
< &2^c\cdot\frac{2}{2^{3c}} \\
\leq &\frac{1}{2^c},
\end{aligned}$$

which implies that g is a c-generator. □

References

[1] M. Blum and S. Micali. How to generate cryptographically strong sequence of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[2] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[3] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[4] A. N. Kolmogorov. Three approaches to the definition of the concept 'quantity of information'. *Problemy Peredachi Informatsii*, 1:3–11, 1965. In Russian.

[5] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[6] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[7] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 80–91, Los Alamitos, Ca., USA, November 1982. IEEE Computer Society Press.