Chapter 17

Derandomization and Extractors

"God does not play dice with the universe"
Albert Einstein

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
John von Neumann, quoted by Knuth 1981

"How hard could it be to find hay in a haystack?"
Howard Karloff

The concept of a *randomized algorithm*, though widespread, has both a philosophical and a practical difficulty associated with it.

The philosophical difficulty is best represented by Einstein's famous quote above. Do random events (such as the unbiased coin flip assumed in our definition of a randomized turing machine) truly exist in the world, or is the world deterministic? The practical difficulty has to do with actually generating random bits, assuming they exist. A randomized algorithm running on a modern computer could need billions of random bits each second. Even if the world contains some randomness —say, the ups and downs of the stock market — it may not have enough randomness to provide billions of uncorrelated random bits every second in the tiny space inside a microprocessor. Current computing environments rely on shortcuts such as taking a small "fairly random looking" bit sequence—e.g., interval between the programmer's keystrokes measured in microseconds—and applying a deterministic generator to turn them into a longer sequence of "sort of random looking" bits. Some recent devices try to use quantum phenomena. But for all of them it is unclear how random and uncorrelated those bits really are.

Such philosophical and practical difficulties look deterring; the philosophical aspect alone has been on the philosophers' table for centuries. The results in the current chapter may be viewed as complexity theory's contribution to these questions.

The first contribution concerns the place of randomness in our world. We indicated in Chapter 7 that randomization seems to help us design more efficient algorithms. A surprising conclusion in this chapter is this could be a mirage to some extent. If certain plausible complexity-theoretic conjectures are true (e.g., that certain problems can not be solved by subexponential-sized circuits) then *every* probabilistic algorithm can be simulated deterministically with only a polynomial slowdown. In other words, randomized algorithms can be *derandomized* and **BPP** = **P**. Nisan and Wigderson [**?**] named this research area *Hardness versus Randomness* since the existence of *hard* problems is shown to imply derandomization. Section 17.3 shows that the converse is also true to a certain extent: ability to derandomize implies circuit lowerbounds (thus, hardness) for concrete problems. Thus the Hardness $\leftrightarrow$ Randomness connection is very real.

Is such a connection of any use at present, given that we have no idea how to prove circuit lowerbounds? Actually, yes. Just as in cryptography, we can use *conjectured* hard problems in the derandomization instead of *provable* hard problems, and end up with a win-win situation: if the conjectured hard problem is truly hard then the derandomization will be successful; and if the derandomization fails then it will lead us to an algorithm for the conjectured hard problem.

The second contribution of complexity theory concerns another practical question: how can we run randomized algorithms given only an imperfect source of randomness? We show the existence of *randomness extractors*: efficient algorithms to extract (uncorrelated, unbiased) random bits from any *weakly random* device.Their analysis is unconditional and uses no unproven assumptions. Below, we will give a precise definition of the properties that such a weakly random device needs to have. We do not resolve the question of whether such weakly random devices exist; this is presumably a subject for physics (or philosophy).

A central result in both areas is Nisan and Wigderson's beautiful construction of a certain *pseudorandom generator*. This generator is tailor-made for derandomization and has somewhat different properties than the *secure* pseudorandom generators we encountered in Chapter 10.

Another result in the chapter is a (unconditional) derandomization of randomized logspace computations, albeit at the cost of some increase in the space requirement.

EXAMPLE 17.1 (POLYNOMIAL IDENTITY TESTING)
One example for an algorithm that we would like to derandomize is the algorithm described in Section 7.2.2 for testing if a given polynomial (represented in the form of an arithmetic zero) is the identically zero polynomial. If p is an n-variable nonzero polynomial of total degree d over a large enough finite field $\mathbb{F}$ ($|\mathbb{F}| > 10d$ will do) then most of the vectors $\mathbf{u} \in \mathbb{F}^n$ will satisfy $p(\mathbf{u}) \neq 0$ (see Lemma A.23. Therefore, checking whether $p \equiv 0$ can be done by simply choosing a random $\mathbf{u} \in_R \mathbb{F}^n$ and applying p on $\mathbf{u}$. In fact, it is easy to show that there exists a set of m^2-vectors $\mathbf{u}^1, \ldots, \mathbf{u}^{m^2}$ such that for *every* such nonzero polynomial p that can be computed by a size m arithmetic circuit, there exists an $i \in [m^2]$ for which $p(\mathbf{u}^i) \neq 0$.

This suggests a natural approach for a deterministic algorithm: show a deterministic algorithm that for every $m \in \mathbb{N}$, runs in poly(m) time and outputs a set $\mathbf{u}^1, \ldots, \mathbf{u}^{m^2}$ of vectors satisfying the above property. This shouldn't be too difficult— after all the vast majority of the sets of vectors have this property, so hard can it be to find a single one? (Howard Karloff calls this task "finding a hay in a haystack"). Surprisingly this turns out to be quite hard: without using complexity assumptions, we do not know how to obtain such a set, and in Section 17.3 we will see that in fact such an algorithm will imply some nontrivial circuit lowerbounds.[1]

17.1 Pseudorandom Generators and Derandomization

The main tool in derandomization is a pseudorandom generator. This is a twist on the definition of a *secure* pseudorandom generator we gave in Chapter 10, with the difference that here we consider nonuniform distinguishers

[1]Perhaps it should not be so surprising that "finding a hay in a haystack" is so hard. After all, the hardest open problems of complexity— finding explicit functions with high circuit complexity— are of this form, since the vast majority of the functions from $\{0,1\}^n$ to $\{0,1\}$ have exponential circuit complexity.

–in other words, circuits— and allow the generator to run in exponential time.

Definition 17.2 (Pseudorandom generators)
Let R be a distribution over $\{0,1\}^m$, $S \in \mathbb{N}$ and $\epsilon > 0$. We say that R is an (S,ϵ)-*pseudorandom distribution* if for every circuit C of size at most S,

$$|\Pr[C(R) = 1] - \Pr[C(U_m) = 1]| < \epsilon$$

where U_m denotes the uniform distribution over $\{0,1\}^m$.

If $S : \mathbb{N} \to \mathbb{N}$ is a polynomial-time computable monotone function (i.e., $S(m) \geq S(n)$ for $m \geq n$)[2] then a function $G : \{0,1\}^* \to \{0,1\}^*$ is called an $(S(\ell)$-*pseudorandom generator* (see Figure 17.1) if:

- For every $z \in \{0,1\}^\ell$, $|G(z)| = S(\ell)$ and $G(z)$ can be computed in time $2^{c\ell}$ for some constant c. We call the input z the *seed* of the pseudorandom generator.
- For every $\ell \in \mathbb{N}$, $G(U_\ell)$ is an $(S(\ell)^3, {}^1\!/\!{}_{10})$-pseudorandom distribution.

Remark 17.3
The choices of the constant 3 and $1/10$ in the definition of an $S(\ell)$-pseudorandom generator are arbitrary and made for convenience.

The relation between pseudorandom generators and simulating probabilistic algorithm is straightforward:

Lemma 17.4
Suppose that there exists an $S(\ell)$-pseudorandom generator for some polynomial-time computable monotone $S : \mathbb{N} \to \mathbb{N}$. Then for every polynomial-time computable function $\ell : \mathbb{N} \to \mathbb{N}$, $\mathbf{BPTIME}(S(\ell(n))) \subseteq \mathbf{DTIME}(2^{c\ell(n)})$ for some constant c.

Proof: A language L is in $\mathbf{BPTIME}(S(\ell(n)))$ if there is an algorithm A that on input $x \in \{0,1\}^n$ runs in time $cS(\ell(n))$ for some constant c, and satisfies

$$\Pr_{r \in_R \{0,1\}^m}[A(x,r) = L(x)] \geq \frac{2}{3}$$

[2]We place these easily satisfiable requirements on the function S to avoid weird cases such as generators whose output length is not computable or generators whose output shrinks as the input grows.

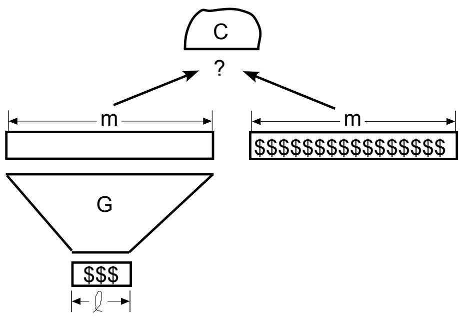

Figure 17.1: A *pseudorandom generator* G maps a short uniformly chosen seed $z \in_R \{0,1\}^\ell$ into a longer output $G(z) \in \{0,1\}^m$ that is indistinguishable from the uniform distribution U_m by any small circuit C.

where $m \leq S(\ell(n))$ and we define $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise.

The main idea is that if we replace the truly random string r with the string $G(z)$ produced by picking a random $z \in \{0,1\}^{\ell(n)}$, then an algorithm like A that runs in only $S(\ell)$ time cannot detect this switch most of the time, and so the probability $2/3$ in the previous expression does not drop below $2/3 - 0.1$. Thus to derandomize A, we do not need to enumerate over all r; it suffices to enumerates over all $z \in \{0,1\}^{\ell(n)}$ and check how many of them make A accept. This derandomized algorithm runs in $\exp(\ell(n))$ time instead of the trivial 2^m time.

Now we make this formal. Our deterministic algorithm B will on input $x \in \{0,1\}^n$, go over all $z \in \{0,1\}^{\ell(n)}$, compute $A(x, G(z))$ and output the majority answer. Note this takes $2^{O(\ell(n))}$ time. We claim that for n sufficiently large, the fraction of z's such that $A(x, G(z)) = L(x)$ is at least $\frac{2}{3} - 0.1$. (This suffices to prove that $L \in \mathbf{DTIME}(2^{c\ell(n)})$ as we can "hardwire" into the algorithm the correct answer for finitely many inputs.)

Suppose this is false and there exists an infinite sequence of x's for which $\Pr[A(x, G(z)) = L(x) < 2/3 - 0.1$. Then we would get a distinguisher for the pseudorandom generator —just use the Cook-Levin transformation to construct a circuit that computes the function $z \mapsto A(x, G(z))$, where x is hardwired into the circuit. This circuit has size $O(S(\ell(n)))^2$ which is smaller than $S(\ell(n))^3$ for sufficiently large n. ■

Remark 17.5

The proof shows why it is OK to allow the pseudorandom generator in Definition 17.2 to run in time exponential in its seed length. The derandomized algorithm enumerates over all possible seeds of length ℓ, and thus would take exponential time (in ℓ) even if the generator itself were to run in less than exponential time.

Notice, these generators have to fool distinguishers that run for *less* time than they do. By contrast, the definition of *secure pseudorandom generators* (Definition 10.11 in Chapter 10) required the generator to run in polynomial time, and yet have the ability to fool distinguishers that have super-polynomial running time. This difference in these definitions stems from the intended usage. In the cryptographic setting the generator is used by honest users and the distinguisher is the adversary attacking the system — and it is reasonable to assume the attacker can invest more computational resources than those needed for normal/honest use of the system. In derandomization, generator is used by the derandomized algorithm, the "distinguisher" is the probabilistic algorithm that is being derandomized, and it is reasonable to allow the derandomized algorithm higher running time than the original probabilistic algorithm.

Of course, allowing the generator to run in exponential time as in this chapter potentially makes it easier to prove their existence compared with secure pseudorandom generators, and this indeed appears to be the case. (Note that if we place no upperbounds on the generator's efficiency, we could prove the existence of generators *unconditionally* as shown in Exercise 2, but these do not suffice for derandomization.)

We will construct pseudorandom generators based on complexity assumptions, using quantitatively stronger assumptions to obtain quantitatively stronger pseudorandom generators (i.e., $S(\ell)$-pseudorandom generators for larger functions S). The strongest (though still reasonable) assumption will yield a $2^{\Omega(\ell)}$-pseudorandom generator, thus implying that $\mathbf{BPP} = \mathbf{P}$. These are described in the following easy corollaries of the Lemma that are left as Exercise 1.

Corollary 17.6

1. *If there exists a $2^{\epsilon\ell}$-pseudorandom generator for some constant $\epsilon > 0$ then $\mathbf{BPP} = \mathbf{P}$.*

2. *If there exists a $2^{\ell^{\epsilon}}$-pseudorandom generator for some constant $\epsilon > 0$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP} = \mathbf{DTIME}(2^{polylog(n)})$.*

3. *If there exists an $S(\ell)$-pseudorandom generator for some super-polynomial function S (i.e., $S(\ell) = \ell^{\omega(1)}$) then $\mathbf{BPP} \subseteq \mathbf{SUBEXP} = \cap_{\epsilon>0}\mathbf{DTIME}(2^{n^{\epsilon}})$.*

17.1.1 Hardness and Derandomization

We construct pseudorandom generators under the assumptions that certain explicit functions are hard. In this chapter we use assumptions about *average-case* hardness, while in the next chapter we will be able to construct pseudorandom generators assuming only *worst-case* hardness. Both worst-case and average-case hardness refers to the size of the minimum Boolean circuit computing the function:

DEFINITION 17.7 (HARDNESS)
Let $f : \{0,1\}^* \to \{0,1\}$ be a Boolean function. The *worst-case hardness* of f, denoted $\mathsf{H}_{\mathsf{wrs}}(f)$, is a function from $\mathbb{N}$ to $\mathbb{N}$ that maps every $n \in \mathbb{N}$ to the largest number S such that every Boolean circuit of size at most S fails to compute f on some input in $\{0,1\}^n$.

The *average-case hardness* of f, denoted $\mathsf{H}_{\mathsf{avg}}(f)$, is a function from $\mathbb{N}$ to $\mathbb{N}$ that maps every $n \in \mathbb{N}$, to the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < \frac{1}{2} + \frac{1}{S}$ for every Boolean circuit C on n inputs with size at most S.

Note that for every function $f : \{0,1\}^* \to \{0,1\}$ and $n \in \mathbb{N}$, $\mathsf{H}_{\mathsf{avg}}(f)(n) \leq \mathsf{H}_{\mathsf{wrs}}(f)(n) \leq n2^n$.

REMARK 17.8
This definition of average-case hardness is tailored to the application of derandomization, and in particular only deals with the uniform distribution over the inputs. See Chapter 15 for a more general treatment of average-case complexity. We will also sometimes apply the notions of worst-case and average-case to *finite* functions from $\{0,1\}^n$ to $\{0,1\}$, where $\mathsf{H}_{\mathsf{wrs}}(f)$ and $\mathsf{H}_{\mathsf{avg}}(f)$ are defined in the natural way. (E.g., if $f : \{0,1\}^n \to \{0,1\}$ then $\mathsf{H}_{\mathsf{wrs}}(f)$ is the largest number S for which every Boolean circuit of size at most S fails to compute f on some input in $\{0,1\}^n$.)

EXAMPLE 17.9
Here are some examples of functions and their conjectured or proven hardness:

1. If f is a random function (i.e., for every $x \in \{0,1\}^*$ we choose $f(x)$ using an independent unbiased coin) then with high probability, both the worst-case and average-case hardness of f are exponential (see

Exercise 3). In particular, with probability tending to 1 with n, both $\mathsf{H}_{\mathsf{wrs}}(f)(n)$ and $\mathsf{H}_{\mathsf{avg}}(f)(n)$ exceed $2^{0.99n}$. We will often use the shorthand $\mathsf{H}_{\mathsf{wrs}}(f), \mathsf{H}_{\mathsf{avg}}(f) \geq 2^{0.99n}$ for such expressions.

2. If $f \in \mathbf{BPP}$ then, since $\mathbf{BPP} \subseteq \mathbf{P}/\mathrm{poly}$, both $\mathsf{H}_{\mathsf{wrs}}(f)$ and $\mathsf{H}_{\mathsf{avg}}(f)$ are bounded by some polynomial.

3. It seems reasonable to believe that 3SAT has exponential worst-case hardness; that is, $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT}) \geq 2^{\Omega(n)}$. It is even more believable that $\mathbf{NP} \nsubseteq \mathbf{P}/\mathrm{poly}$, which implies that $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT})$ is superpolynomial. The average case complexity of 3SAT is unclear, and in any case dependent upon the way we choose to represent formulas as strings.

4. If we believe in current cryptosystems, then we do believe that **NP** contains functions that are hard on the average. If g is a one-way permutation that cannot be inverted with polynomial probability by polynomial-sized circuits, then by Theorem 10.14, the function f that maps the pair $x, r \in \{0,1\}^n$ to $g^{-1}(x) \odot r$ has super-polynomial *average-case* hardness: $\mathsf{H}_{\mathsf{avg}}(f) \geq n^{\omega(1)}$. (Where $x \odot r = \sum_{i=1}^{n} x_i r_i \pmod 2$.) More generally there is a polynomial relationship between the size of the minimal circuit that inverts g (on the average) and the average-case hardness of f.

The main theorem of this section uses hard-on-the average functions to construct pseudorandom generators:

THEOREM 17.10 (CONSEQUENCES OF NW GENERATOR)
For every polynomial-time computable monotone $S : \mathbb{N} \to \mathbb{N}$, if there exists a constant c and function $f \in \mathbf{DTIME}(2^{cn})$ such that $\mathsf{H}_{\mathsf{avg}}(f) \geq S(n)$ then there exists a constant $\epsilon > 0$ such that an $S(\epsilon\ell)^{\epsilon}$-pseudorandom generator exists. In particular, the following corollaries hold:

1. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{\epsilon n}$ then* $\mathbf{BPP} = \mathbf{P}$.
2. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n^{\epsilon}}$ then* $\mathbf{BPP} \subseteq \mathbf{QuasiP}$.
3. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{avg}}(f) \geq n^{\omega(1)}$ then* $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$.

REMARK 17.11
We can replace $\mathbf{E}$ with $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(n)})$ in Corollaries 2 and 3 above. Indeed, for every $f \in \mathbf{DTIME}(2^{n^c})$, the function g that on input $x \in \{0,1\}^*$ outputs the f applies to the first $|x|^{1/c}$ bits of x is in $\mathbf{DTIME}(2^n)$ and satisfies $\mathsf{H}_{\mathsf{avg}}(g)(n) \geq \mathsf{H}_{\mathsf{avg}}(f)(n^{1/c})$. Therefore, if there exists $f \in \mathbf{EXP}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n^{\epsilon}}$ then there there exists a constant $\epsilon' > 0$ and a function $g \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(g) \geq 2^{n^{\epsilon'}}$, and so we can replace $\mathbf{E}$ with $\mathbf{EXP}$ in Corollary 2. A similar observation holds for Corollary 3. Note that $\mathbf{EXP}$ contains many classes we believe to have hard problems, such as $\mathbf{NP}$,$\mathbf{PSPACE}$,$\oplus\mathbf{P}$ and more, which is why we believe it does contain hard-on-the-average functions. In the next chapter we will give even stronger evidence to this conjecture, by showing it is implied by the assumption that $\mathbf{EXP}$ contains hard-in-the-worst-case functions.

REMARK 17.12
The original paper of Nisan and Wigderson [**?**] did not prove Theorem 17.10 as stated above. It was proven in a sequence of works [**?**]. Nisan and Wigderson only proved that under the same assumptions there exists an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S\left(\epsilon\sqrt{\ell}\log(S(\epsilon\sqrt{\ell})\right)^{\epsilon}$ for some $\epsilon > 0$. Note that this is still sufficient to derive all three corollaries above. It is this weaker version we prove in this book.

17.2 Proof of Theorem 17.10: Nisan-Wigderson Construction

How can we use a hard function to construct a pseudorandom generator?

17.2.1 Warmup: two toy examples

For starters, we demonstrate this by considering the "toy example" of a pseudorandom generator whose output is only one bit longer than its input. Then we show how to extend by two bits. Of course, neither suffices to prove Theorem 17.10 but they do give insight to the connection between hardness and randomness.

Extending the input by one bit using Yao's Theorem.

The following Lemma uses a hard function to construct such a "toy" generator:

Lemma 17.13 (One-bit generator)
Suppose that there exist $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. Then, there exists an $S(\ell)$-pseudorandom generator G for $S(\ell) = \ell + 1$.

Proof: The generator G will be very simple: for every $z \in \{0,1\}^\ell$, we set

$$G(z) = z \circ f(z)$$

(where $\circ$ denotes concatenation). G clearly satisfies the output length and efficiency requirements of an $(\ell+1)$-pseudorandom generator. To prove that its output is $1/10$-pseudorandom we use Yao's Theorem from Chapter 10 showing that pseudorandomness is implied by unpredictiability:[3]

Theorem 17.14 (Theorem 10.12, restated)
Let Y be a distribution over $\{0,1\}^m$. Suppose that there exist $S > 10n, \epsilon > 0$ such that for every circuit C of size at most $2S$ and $i \in [m]$,

$$\Pr_{r \in_R Y}[C(r_1, \ldots, r_{i-1}) = r_i] \leq \frac{1}{2} + \frac{\epsilon}{m}$$

Then Y is (S, ϵ)-pseudorandom.

[3] Although this theorem was stated and proved in Chapter 10 for the case of *uniform* Turing machines, the proof easily extends to the case of circuits.

Using Theorem 17.14 it is enough to show that there does not exist a circuit C of size $2(\ell+1)^3 < \ell^4$ and a number $i \in [\ell+1]$ such that

$$\Pr_{r=G(U_\ell)}[C(r_1,\ldots,r_{i-1}) = r_i] > \tfrac{1}{2} + \tfrac{1}{20(\ell+1)}. \qquad (1)$$

However, for every $i \leq \ell$, the i^{th} bit of $G(z)$ is completely uniform and independent from the first $i-1$ bits, and hence cannot be predicted with probability larger than $1/2$ by a circuit of any size. For $i = \ell+1$, Equation (1) becomes,

$$\Pr_{z \in_R \{0,1\}^\ell}[C(z) = f(z)] > \frac{1}{2} + \frac{1}{20(\ell+1)} > \frac{1}{2} + \frac{1}{\ell^4},$$

which cannot hold under the assumption that $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. ■

Extending the input by two bits using the averaging principle.

We now continue to progress in "baby steps" and consider the next natural toy problem: constructing a pseudorandom generator that extends its input by two bits. This is obtained in the following Lemma:

Lemma 17.15 (Two-bit generator)
Suppose that there exists $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. Then, there exists an $(\ell+2)$-pseudorandom generator G.

Proof: The construction is again very natural: for every $z \in \{0,1\}^\ell$, we set

$$G(z) = z_1 \cdots z_{\ell/2} \circ f(z_1,\ldots,z_{\ell/2}) \circ z_{\ell/2+1} \cdots z_\ell \circ f(z_{\ell/2+1},\ldots,z_\ell).$$

Again, the efficiency and output length requirements are clearly satisfied.

To show $G(U_\ell)$ is $1/10$-pseudorandom, we again use Theorem 17.14, and so need to prove that there does not exists a circuit C of size $2(\ell+1)^3$ and $i \in [\ell+2]$ such that

$$\Pr_{r=G(U_\ell)}[C(r_1,\ldots,r_{i-1}) = r_i] > \frac{1}{2} + \frac{1}{20(\ell+2)}. \qquad (2)$$

Once again, (2) cannot occur for those indices i in which the i^{th} output of $G(z)$ is truly random, and so the only two cases we need to consider are $i = \ell/2+1$ and $i = \ell+2$. Equation (2) cannot hold for $i = \ell/2+1$ for the same reason as in Lemma 17.13. For $i = \ell+2$, Equation (2) becomes:

$$\Pr_{r,r' \in_R \{0,1\}^{\ell/2}}[C(r \circ f(r) \circ r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)} \qquad (3)$$

This may seem somewhat problematic to analyze since the input to C contains the bit $f(r)$, which C could not compute on its own (as f is a hard function). Couldn't it be that the input $f(r)$ helps C in predicting the bit $f(r')$? The answer is NO, and the reason is that r' and r are *independent*. Formally, we use the following principle (see Section A.2.2 in the appendix):

> The Averaging Principle: If A is some event depending on two independent random variables X, Y, then there exists some x in the range of X such that
>
> $$\Pr_Y[A(x,Y) \geq \Pr_{X,Y}[A(X,Y)]$$

Applying this principle here, if (3) holds then there exists a string $r \in \{0,1\}^{\ell/2}$ such that

$$\Pr_{r' \in_R \{0,1\}^{\ell/2}}[C(r, f(r), r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)} \, .$$

(Note that this probability is now only over the choice of r'.) If this is the case, we can "hardwire" the $\ell/2+1$ bits $r \circ f(r)$ to the circuit C and obtain a circuit D of size at most $(\ell+2)^3 + 2\ell < (\ell/2)^4$ such that

$$\Pr_{r' \in_R \{0,1\}^{\ell/2}}[D(r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)} \, ,$$

contradicting the hardness of f. ■

Beyond two bits:

A generator that extends the output by two bits is still useless for our goals. We can generalize the proof Lemma 17.15 to obtain a generator G that extends the output by k bits setting

$$G(z_1, \ldots, z_\ell) = z^1 \circ f(z^1) \circ z^2 \circ f(z^2) \cdots z^k \circ f(z^k) \, , \tag{4}$$

where z^i is the i^{th} block of ℓ/k bits in z. However, no matter how big we set k and no matter how hard the function f is, we cannot get a generator that expands its input by a multiplicative factor larger than two. Note that to prove Theorem 17.10 we need a generator that, depending on the hardness we assume, has output that can be exponentially larger than the input! Clearly, we need a new idea.

17.2.2 The NW Construction

The new idea is still inspired by the construction of (4), but instead of taking $z^1, \ldots, z^k$ to be independently chosen strings (or equivalently, disjoint pieces of the input z), we take them to be *partly dependent* by using *combinatorial designs*. Doing this will allow us to take k so large that we can drop the actual inputs from the generator's output and use only $f(z^1) \circ f(z^2) \cdots \circ f(z^k)$. The proof of correctness is similar to the above toy examples and uses Yao's technique, except the fixing of the input bits has to be done more carefully because of dependence among the strings.

First, some notation. For a string $z \in \{0,1\}^\ell$ and subset $I \subseteq [\ell]$, we define $z_{\restriction I}$ to be $|I|$-length string that is the projection of z to the coordinates in I. For example, $z_{\restriction [1..i]}$ is the first i bits of z.

Definition 17.16 (NW Generator)
If $\mathcal{I} = \{I_1, \ldots, I_m\}$ is a family of subsets of $[\ell]$ with each $|I_j| = l$ and $f : \{0,1\}^n \to \{0,1\}$ is any function then the $(\mathcal{I}, f)$-*NW generator* (see Figure 17.2) is the function $\mathsf{NW}^f_{\mathcal{I}} : \{0,1\}^\ell \to \{0,1\}^m$ that maps any $z \in \{0,1\}^\ell$ to

$$\mathsf{NW}^f_{\mathcal{I}}(z) = f(z_{\restriction I_1}) \circ f(z_{\restriction I_2}) \cdots \circ f(z_{\restriction I_m}) \quad (5)$$

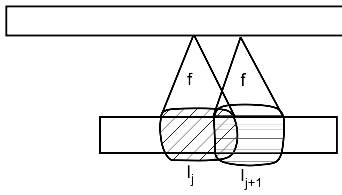

Figure 17.2: The *NW generator*, given a set system $\mathcal{I} = \{I_1, \ldots, I_m\}$ of size n subsets of $[\ell]$ and a function $f : \{0,1\}^n \to \{0,1\}$ maps a string $z \in \{0,1\}^\ell$ to the output $f(z_{\restriction I_1}), \ldots, f(z_{\restriction I_m})$. Note that these sets are not necessarily disjoint (although we will see their intersections need to be small).

Conditions on the set systems and function.

We will see that in order for the generator to produce pseudorandom outputs, function f must display some *hardness*, and the family of subsets must come from an efficiently constructible combinatorial design.

Definition 17.17 (Combinatorial designs)
If $d, n, \ell \in \mathbb{N}$ are numbers with $\ell > n > d$ then a family $\mathcal{I} = \{I_1, \ldots, I_m\}$ of subsets of $[\ell]$ is an (ℓ, n, d)-*design* if $|I_j| = n$ for every j and $|I_j \cap I_k| \leq d$ for every $j \neq k$.

The next lemma yields efficient constructions of these designs and is proved later.

Lemma 17.18 (Construction of designs)
There is an algorithm A such that on input $\ell, d, n \in \mathbb{N}$ where $n > d$ and $\ell > 10n^2/d$, runs for $2^{O(\ell)}$ steps and outputs an (ℓ, n, d)-design $\mathcal{I}$ containing $2^{d/10}$ subsets of $[\ell]$.

The next lemma shows that if f is a hard function and $\mathcal{I}$ is a design with sufficiently good parameters, than $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is indeed a pseudorandom distribution:

Lemma 17.19 (Pseudorandomness using the NW generator)
If $\mathcal{I}$ is an (ℓ, n, d)-design with $|\mathcal{I}| = 2^{d/10}$ and $f : \{0,1\}^n \to \{0,1\}$ a function satisfying $2^d < \sqrt{\mathsf{H}_{\mathsf{avg}}(f)(n)}$, then the distribution $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is a $(\mathsf{H}_{\mathsf{avg}}(f)(n)/10, 1/10)$-pseudorandom distribution.

Proof: Let S denote $\mathsf{H}_{\mathsf{avg}}(f)(n)$. By Yao's Theorem, we need to prove that for every $i \in [2^{d/10}]$ there does *not* exist an $S/2$-sized circuit C such that

$$\Pr_{\substack{Z \sim U_\ell \\ R = \mathsf{NW}^f_{\mathcal{I}}(Z)}} [C(R_1, \ldots, R_{i-1}) = R_i] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} \,. \tag{6}$$

For contradiction's sake, assume that (6) holds for some circuit C and some i. Plugging in the definition of $\mathsf{NW}^f_{\mathcal{I}}$, Equation (6) becomes:

$$\Pr_{Z \sim U_\ell}[C(f(Z_{\restriction I_1}), \cdots, f(Z_{\restriction I_{i-1}})) = f(Z_{\restriction I_i})] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} \,. \tag{7}$$

Letting Z_1 and Z_2 denote the two independent variables corresponding to the coordinates of Z in I_i and $[\ell] \setminus I_i$ respectively, Equation (7) becomes:

$$\Pr_{\substack{Z_1 \sim U_n \\ Z_2 \sim U_{\ell - n}}} [C(f_1(Z_1, Z_2), \ldots, f_{i-1}(Z_1, Z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} \,, \tag{8}$$

where for every $j \in [2^{d/10}]$, f_j applies f to the coordinates of Z_1 corresponding to $I_j \cap I_i$ and the coordinates of Z_2 corresponding to $I_j \setminus I_i$. By the averaging principle, if (8) holds then there exists a string $z_2 \in \{0,1\}^{\ell-n}$ such that

$$\Pr_{Z_1 \sim U_n}[C(f_1(Z_1,z_2),\dots,f_{i-1}(Z_1,z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}\,. \quad (9)$$

We may now appear to be in some trouble, since all of $f_j(Z_1,z_2)$ for $j \leq i-1$ do depend upon Z_1, and the fear is that if they together contain enough information about Z_1 then a circuit *could potentially* predict $f_i(Z_1)$ after looking at all of them. To prove that this fear is baseless we use the fact that the circuit C is small and f is a very hard function.

Since $|I_j \cap I_i| \leq d$ for $j \neq i$, the function $Z_1 \mapsto f_j(Z_1,z_2)$ depends at most d coordinates of z_1 and hence can be computed by a $d2^d$-sized circuit. (Recall that z_2 is fixed.) Thus if if (8) holds then there exists a circuit B of size $2^{d/10} \cdot d2^d + S/2 < S$ such that

$$\Pr_{Z_1 \sim U_n}[B(Z_1) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}. \quad (10)$$

But this contradicts the fact that $\mathsf{H}_{\mathsf{avg}}(f)(n) = S$. ■

Remark 17.20 (Black-box proof)
Lemma 17.19 shows that if $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is distinguishable from the uniform distribution $U_{2^{d/10}}$ by some circuit D, then there exists a circuit B (of size polynomial in the size of D and in 2^d) that computes the function f with probability noticeably larger than 1/2. The construction of this circuit B actually uses the circuit D as a *black-box*, invoking it on some chosen inputs. This property of the NW generator (and other constructions of pseudorandom generators) turned out to be useful in several settings. In particular, Exercise 5 uses it to show that under plausible complexity assumptions, the complexity class **AM** (containing all languages with a constant round interactive proof, see Chapter 9) is equal to **NP**. We will also use this property in the construction of *randomness extractors* based on pseudorandom generators.

Putting it all together: Proof of Theorem 17.10 from Lemmas 17.18 and 17.19

As noted in Remark 17.12, we do not prove here Theorem 17.10 as stated but only the weaker statement, that given $f \in \mathbf{E}$ and $S : \mathbb{N} \to \mathbb{N}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq S$, we can construct an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S\left(\epsilon\sqrt{\ell}\log(S(\epsilon\sqrt{\ell})\right)^{\epsilon}$ for some $\epsilon > 0$.

For such a function f, we denote our pseudorandom generator by NW^f. Given input $z \in \{0,1\}^\ell$, the generator NW^f operates as follows:

- Set n to be the largest number such that $\ell > 100n^2/\log S(n)$. Set $d = \log S(n)/10$. Since $S(n) < 2^n$, we can assume that $\ell \leq 300n^2/\log S(n)$.
- Run the algorithm of Lemma 17.18 to obtain an (ℓ, n, d)-design $\mathcal{I} = \{I_1, \ldots, I_{2^{d/5}}\}$.
- Output the first $S(n)^{1/40}$ bits of $\mathsf{NW}^f_{\mathcal{I}}(z)$.

Clearly, $\mathsf{NW}^f(z)$ runs in $2^{O(\ell)}$ time. Moreover, since $2^d \leq S(n)^{1/10}$, Lemma 17.19 implies that the distribution $\mathsf{NW}^f(U_\ell)$ is $(S(n)/10, {}^1\!/_{10})$-pseudorandom. Since $n \geq \sqrt{\ell}\log S(n)/300 \geq \sqrt{\ell}\log S(\frac{\sqrt{\ell}}{300})/300$ (with the last inequality following from the fact that S is monotone), this concludes the proof of Theorem 17.10. ■

Construction of combinatorial designs.

All that is left to complete the proof is to show the construction of combinatorial designs with the required parameters:

Proof of Lemma 17.18 (construction of combinatorial designs): On inputs ℓ, d, n with $\ell > 10n^2/d$, our Algorithm A will construct an (ℓ, n, d)-design $\mathcal{I}$ with $2^{d/10}$ sets using the simple greedy strategy:

> Start with $\mathcal{I} = \emptyset$ and after constructing $\mathcal{I} = \{I_1, \ldots, I_m\}$ for $m < 2^{d/10}$, search all subsets of $[\ell]$ and add to $\mathcal{I}$ the first n-sized set I satisfying $|I \cap I_j| \leq d$ for every $j \in [m]$. We denote this latter condition by **(*)**.

Clearly, A runs in $\mathrm{poly}(m)2^\ell = 2^{O(\ell)}$ time and so we only need to prove it never gets stuck. In other words, it suffices to show that if $\ell = 10n^2/d$ and $\{I_1, \ldots, I_m\}$ is a collection of n-sized subsets of $[\ell]$ for $m < 2^{d/10}$, then there exists an n-sized subset $I \subseteq [\ell]$ satisfying **(*)**. We do so by showing that if we pick I at random by choosing independently every element $x \in [\ell]$ to be in I with probability $2n/\ell$ then:

$$\Pr[|I| \geq n] \geq 0.9 \tag{11}$$

$$\Pr[|I \cap I_j| \geq d] \leq 0.5 \cdot 2^{-d/10} \qquad (\forall j \in [m]) \tag{12}$$

Because the expected size of I is $2n$, while the expected size of the intersection $I \cap I_j$ is $2n^2/\ell < d/5$, both (12) and (11) follow from the Chernoff bound. Yet together these two conditions imply that with probability at least 0.4, the set I will simultaneously satisfy **(*)** and have size at least n. Since we can always remove elements from I without damaging **(*)**, this completes the proof. ■

17.3 Derandomization requires circuit lowerbounds

We saw in Section 17.2 that if we can prove certain strong circuit lowerbounds, then we can partially (or fully) derandomize **BPP**. Now we prove a result in the reverse direction: derandomizing **BPP** requires proving circuit lowerbounds. Depending upon whether you are an optimist or a pessimist, you can view this either as evidence that derandomizing **BPP** is difficult, or, as a reason to double our efforts to derandomize **BPP**.

We say that a function is in $\mathbf{AlgP}/poly$ if it can be computed by a polynomial size arithmetic circuit whose gates are labeled by $+$, $-$, $\times$ and $\div$, which are operations over some underlying field or ring. We let perm denote the problem of computing the permanent of matrices over the integers. (The proof can be extended to permanent computations over finite fields of characteristic > 2.) We prove the following result.

Theorem 17.21 ([?])
$\mathbf{P} = \mathbf{BPP} \Rightarrow \mathbf{NEXP} \not\subseteq \mathbf{P}/poly$ *or* $\mathsf{perm} \notin \mathbf{AlgP}/poly$.

Remark 17.22
It is possible to replace the "poly" in the conclusion $\mathsf{perm} \notin \mathbf{AlgP}/poly$ with a subexponential function by appropriately modifying Lemma 17.25. It is open whether the conclusion $\mathbf{NEXP} \not\subseteq \mathbf{P}/poly$ can be similarly strengthened.

In fact, we will prove the following stronger theorem. Recall the *Polynomial Identity Testing* (ZEROP) problem in which the input consists of a polynomial represented by an arithmetic circuit computing it (see Section 7.2.2 and Example 17.1), and we have to decide if it is the identically zero polynomial. This problem is in $\mathbf{coRP} \subseteq \mathbf{BPP}$ and we will show that if it is in $\mathbf{P}$ then the conclusions of Theorem 17.21 hold:

Theorem 17.23 (Derandomization implies lower bounds)
If $\mathsf{ZEROP} \in \mathbf{P}$ *then either* $\mathbf{NEXP} \not\subseteq \mathbf{P}/poly$ *or* $\mathsf{perm} \notin \mathbf{AlgP}/poly$.

The proof relies upon many results described earlier in the book.[4] Recall that **MA** is the class of languages that can be proven by a one round interactive proof between two players Arthur and Merlin (see Definition 9.7). Merlin is an all-powerful prover and Arthur is a polynomial-time verifier that can flip random coins. That is, given an input x, Merlin first sends Arthur a "proof" y. Then Arthur with y in hand flips some coins and decides whether or not to accept x. For this to be an **MA** protocol, Merlin must convince Arthur to accept strings in L with probability one while at the same time Arthur must not be fooled into accepting strings not in L except with probability smaller than $1/2$. We will use the following result regarding **MA**:

LEMMA 17.24 ([?],[?])
$\mathbf{EXP} \subseteq \mathbf{P}/poly \Rightarrow \mathbf{EXP} = \mathbf{MA}$.

PROOF: Suppose $\mathbf{EXP} \subseteq \mathbf{P}/poly$. By the Karp-Lipton theorem (Theorem 6.14), in this case **EXP** collapses to the second level Σ_2^p of the polynomial hierarchy. Hence $\Sigma_2^p = \mathbf{PH} = \mathbf{PSPACE} = \mathbf{IP} = \mathbf{EXP} \subseteq \mathbf{P}/poly$. Thus every $L \in \mathbf{EXP}$ has an interactive proof, and furtheremore, since $\mathbf{EXP} = \mathbf{PSPACE}$, we can just the use the interactive proof for TQBF, for which the prover is a **PSPACE** machine. Hence the prover can be replaced by a polynomial size circuit family C_n. Now we see that the interactive proof can actually be carried out in 2 rounds, with Merlin going first. Given an input x of length n, Merlin gives Arthur a polynomial size circuit C, which is supposed to be the C_n for L. Then Arthur runs the interactive proof for L, using C as the prover. Note that if the input is not in the language, then *no* prover has a decent chance of convincing the verifier, so this is true also for prover described by C. Thus we have described an **MA** protocol for L implying that $\mathbf{EXP} \subseteq \mathbf{MA}$ and hence that $\mathbf{EXP} = \mathbf{MA}$. ■

Our next ingredient for the proof of Theorem 17.23 is the following lemma:

LEMMA 17.25
If ZEROP $\in \mathbf{P}$, *and* perm $\in \mathbf{AlgP}/poly$. *Then* $\mathbf{P}^{\mathsf{perm}} \subseteq \mathbf{NP}$.

PROOF: Suppose perm has algebraic circuits of size n^c, and that ZEROP has a polynomial-time algorithm. Let L be a language that is decided by an n^d-time TM M using queries to a perm-oracle. We construct an **NP** machine N for L.

[4] This is a good example of "third generation" complexity results that use a clever combination of both "classical" results from the 60's and 70's and newer results from the 1990's.

Suppose x is an input of size n. Clearly, M's computation on x makes queries to perm of size at most $m = n^d$. So N will use nondeterminism as follows: it guesses a sequence of m algebraic circuits $C_1, C_2, \ldots, C_m$ where C_i has size i^c. The hope is that C_i solves perm on $i \times i$ matrices, and N will verify this in poly(m) time. The verification starts by verifying C_1, which is trivial. Inductively, having verified the correctness of $C_1, \ldots, C_{t-1}$, one can verify that C_t is correct using downward self-reducibility, namely, that for a $t \times t$ matrix A,

$$\mathsf{perm}(A) = \sum_{i=1}^{t} a_{1i}\mathsf{perm}(A_{1,i}),$$

where $A_{1,i}$ is the $(t-1)\times(t-1)$ sub-matrix of A obtained by removing the 1st row and ith column of A. Thus if circuit C_{t-1} is known to be correct, then the correctness of C_t can be checked by substituting $C_t(A)$ for $\mathsf{perm}(A)$ and $C_{t-1}(A_{1,i})$ for $\mathsf{perm}(A_{1,i})$: this yields an identity involving algebraic circuits with t^2 inputs which can be verified deterministically in poly(t) time using the algorithm for ZEROP. Proceeding this way N verifies the correctness of $C_1, \ldots, C_m$ and then simulates M^{perm} on input x using these circuits. ■

The heart of the proof is the following lemma, which is interesting in its own right:

Lemma 17.26 ([**?**])
$\mathbf{NEXP} \subseteq \mathbf{P}/poly \Rightarrow \mathbf{NEXP} = \mathbf{EXP}$.

Proof: We prove the contrapositive. Suppose that $\mathbf{NEXP} \neq \mathbf{EXP}$ and let $L \in \mathbf{NEXP} \setminus \mathbf{EXP}$. Since $L \in \mathbf{NEXP}$ there exists a constant $c > 0$ and a relation R such that

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{2^{|x|^c}} \text{ s.t. } R(x,y) \text{ holds},$$

where we can test whether $R(x,y)$ holds in time $2^{|x|^{c'}}$ for some constant c'.

For every constant $d > 0$, let M_d be the following machine: on input $x \in \{0,1\}^n$ enumerate over all possible Boolean circuits C of size n^{100d} that take n^c inputs and have a single output. For every such circuit let $\mathsf{tt}(C)$ be the 2^{n^c}-long string that corresponds to the truth table of the function computed by C. If $R(x, \mathsf{tt}(C))$ holds then halt and output 1. If this does not hold for any of the circuits then output 0.

Since M_d runs in time $2^{n^{101d}+n^c}$, under our assumption that $L \notin \mathbf{EXP}$, for every d there exists an infinite sequence of inputs $\mathcal{X}_d = \{x_i\}_{i\in\mathbb{N}}$ on which $M_d(x_i)$ outputs 0 even though $x_i \in L$ (note that if $M_d(x) = 1$ then $x \in L$).

This means that for every string x in the sequence $\mathcal{X}_d$ and every y such that $R(x, y)$ holds, the string y represents the truth table of a function on n^c bits that cannot be computed by circuits of size n^{100d}, where $n = |x|$. Using the pseudorandom generator based on worst-case assumptions (Theorem **??**), we can use such a string y to obtain an ℓ^d-pseudorandom generator.

Now, if $\mathbf{NEXP} \subseteq \mathbf{P}/\mathrm{poly}$ then as noted above $\mathbf{NEXP} \subseteq \mathbf{MA}$ and hence every language in **NEXP** has a proof system where Merlin proves that an n-bit string is in the language by sending a proof which Arthur then verifies using a probabilistic algorithm of at most n^d steps. Yet, if n is the input length of some string in the sequence $\mathcal{X}_d$ and we are given $x \in \mathcal{X}_d$ with $|x| = n$, then we can replace Arthur by non-deterministic $\mathrm{poly}(n^d)2^{n^c}$ time algorithm that does not toss any coins: Arthur will guess a string y such that $R(x, y)$ holds and then use y as a function for a pseudorandom generator to verify Merlin's proof.

This means that there is a constant $c > 0$ such that *every* language in **NEXP** can be decided on infinitely many inputs by a non-deterministic algorithm that runs in $\mathrm{poly}(2^{n^c})$-time and uses n bits of advice (consisting of the string $x \in \mathcal{X}_d$). Under the assumption that $\mathbf{NEXP} \subseteq \mathbf{P}/\mathrm{poly}$ we can replace the $\mathrm{poly}(2^{n^c})$ running time with a circuit of size $n^{c'}$ where c' is a constant depending only on c, and so get that there is a constant c' such that every language in **NEXP** can be decided on infinitely many inputs by a circuit family of size $n + n^{c'}$. Yet this can be ruled out using elementary diagonalization. ■

REMARK 17.27
It might seem that Lemma 17.26 should have an easier proof that goes along the proof that $\mathbf{EXP} \subseteq \mathbf{P}/\mathrm{poly} \Rightarrow \mathbf{EXP} = \mathbf{MA}$, but instead of using the interactive proof for TQBF uses the *multi-prover* interactive proof system for **NEXP**. However, we do not know how to implement the provers' strategies for this latter system in **NEXP**. (Intuitively, the problem arises from the fact that a **NEXP** statement may have several certificates, and it is not clear how we can ensure all provers use the same one.)

We now have all the ingredients for the proof of Theorem 17.23.

PROOF OF THEOREM 17.23: For contradiction's sake, assume that the following are all true:

$$\mathsf{ZEROP} \in \mathbf{P} \tag{13}$$
$$\mathbf{NEXP} \subseteq \mathbf{P}/poly, \tag{14}$$
$$\mathsf{perm} \in \mathbf{AlgP}/poly. \tag{15}$$

Statement (14) together with Lemmas 17.24 and 17.26 imply that $\mathbf{NEXP} = \mathbf{EXP} = \mathbf{MA}$. Now recall that $\mathbf{MA} \subseteq \mathbf{PH}$, and that by Toda's Theorem (Theorem 8.11) $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$. Recall also that by Valiant's Theorem (Theorem 8.8) perm is $\#\mathbf{P}$-complete. Thus, under our assumptions

$$\mathbf{NEXP} \subseteq \mathbf{P}^{\mathsf{perm}}. \tag{16}$$

Since we assume that $\mathsf{ZEROP} \in \mathbf{P}$, Lemma 17.25 together with statements (15) and (16) implies that $\mathbf{NEXP} \subseteq \mathbf{NP}$, contradicting the Nondeterministic Time Hierarchy Theorem (Theorem 4.3). Thus the three statements at the beginning of the proof cannot be simultaneously true. ■

17.4 Weak Random Sources and Extractors

Suppose, that despite the philosophical difficulties, we are happy with probabilistic algorithms, and sees no need to "derandomize" them, especially at the expense of some unproven assumptions. We still need to tackle the fact that real world sources of randomness and unpredictability rarely, if ever, behave as a sequence of perfectly uncorrelated and unbiased coin tosses. Can we still execute probabilistic algorithms using real-world "weakly random" sources?

Min Entropy

For starters, we need to define what we mean by a weakly random source. It turns out that following notion best captures the amount of randomness in a distribution:

Definition 17.28
Let X be a random variable. The *min entropy* of X, denoted by $H_\infty(X)$, is the largest real number k such that $\Pr[X = x] \leq 2^{-k}$ for every x in the range of X.

It is not hard to see that if X is a random variable over $\{0,1\}^n$ then $H_\infty(X) \leq n$ with $H_\infty(X) = n$ if and only if X is distributed according to the uniform distribution U_n. Our goal in this section is to be able to execute probabilistic algorithms given access to a distribution X with $H_\infty(X)$ as small as possible. It can be shown that, in general, to execute a probabilistic algorithm that uses k random bits, we need access to a distribution X with $H_\infty(X) \geq k$ (see Exercise **??**), we call such a distribution an (n,k)-*source.*

Example 17.29
Here are some examples for distributions X over $\{0,1\}^n$ and their min-entropy:

- (Bit fixing and generalized bit fixing sources) If there is subset $S \subseteq [n]$ with $|S| = k$ such that X's projection to the coordinates in S is uniform over $\{0,1\}^k$, and X's projection to $[n] \setminus S$ is a fixed string (say the all-zeros string) then $H_\infty(X) = k$. The same holds if X's projection to $[n] \setminus S$ is a fixed deterministic function of its projection to S. For example, if the bits in the odd positions of X are independent and uniform and for every even position $2i$, $X_{2i} = X_{2i-1}$ then $H_\infty(X) = \lceil \frac{n}{2} \rceil$. This may model a scenario where we measure some real world data at too high a rate (think of measuring every second a physical event that changes only every minute).

- (Linear subspaces) If X is the uniform distribution over a linear subspace of $\mathrm{GF}(2)^n$ of dimension k, then $H_\infty(X) = k$. (In this case X is actually a generalized bit-fixing source— can you see why?)

- (Biased coins) If X is composed of n independent coins, each outputting 1 with probability $\delta < 1/2$ and 0 with probability $1-\delta$, then as n grows, $H_\infty(X)$ tends to $H(\delta)n$ where H is the Shannon entropy function. That is, $H(\delta) = \delta \log \frac{1}{\delta} + (1-\delta)\log \frac{1}{1-\delta}$.

- (Santha-Vazirani sources) If X has the property that for every $i \in [n]$, and every string $x \in \{0,1\}^{i-1}$, conditioned on $X_1 = x_1, \ldots, X_{i-1} = x_{i-1}$ it holds that both $\Pr[X_i = 0]$ and $\Pr[X_i = 1]$ are between δ and $1-\delta$ then $H_\infty(X) \geq H(\delta)n$. This can model sources such as stock market fluctuations, where current measurements do have some limited dependence on the previous history.

- (Uniform over subset) If X is the uniform distribution over a set $S \subseteq \{0,1\}^n$ with $|S| = 2^k$ then $H_\infty(X) = k$. As we will see, this is a very general case that "essentially captures" all distributions X with $H_\infty(X) = k$.

We see that min entropy is a pretty general notion, and distributions with significant min entropy can model many real-world sources of randomness.

Thoughts on our definition of an imperfect source. The reader may wonder how sensitive our results will be to our definition of an imperfect source of randomness, specifically, the use of min-entropy. We briefly argue now that our definition is the "right" one. Of course, this does not imply that such sources actually exist; that is ultimately a question for physics.

First, note that the definition of an (N, k)-source (due to Zuckerman) was not arbitrary. The definition evolved over several years, and contains all previous attempted definitions as subcases.

Second, our goal in the definition is to capture the minimum properties necessary for running probabilistic computations, and for this purpose high min-entropy is necessary and sufficient. Suppose we have an (N, k) source and desire to run a Turing machine computation that requires m random bits and runs in time T. The exercises ask you to prove that if the Turing machine is simulated in "black box" fashion (i.e., without looking at the working of the algorithm), then m cannot exceed $k + O(\log T)$. Thus nontrivial min-entropy is necessary for simulating randomized algorithms. The sufficiency follows from a simulation we give below in Section 17.5.2 that allows $m = k^{1/3}$, regardless of T. The proof can be tightened to work for $m = k^{1-\epsilon}$ for any $\epsilon > 0$ and more recent constructions allow $m = \Omega(k)$.

Statistical distance and Extractors

Now we try to formalize what it means to *extract* random —more precisely, almost random— bits from an (n, k) source. To do so we will need the following way of quantifying when two distributions are close.

DEFINITION 17.30 (STATISTICAL DISTANCE)
For two random variables X and Y with range $\{0,1\}^m$, their *statistical distance* (also known as *variation distance*) is defined as $\delta(X, Y) = \max_{S\subseteq\{0,1\}^m}\{|\mathbf{Pr}[X \in S] - \mathbf{Pr}[Y \in S]|\}$. We say that X, Y are *ϵ-close*, denoted $X \approx_\epsilon Y$, if $\delta(X, Y) \leq \epsilon$.

Statistical distance lies in $[0, 1]$ and satisfies triangle inequality, as suggested by its name. The next lemma gives some other useful properties; the proof is left as an exercise.

LEMMA 17.31
Let X, Y be any two distributions taking values in $\{0,1\}^n$.

1. $\delta(X, Y) = \frac{1}{2}\sum_{x\in\{0,1\}^n} |\mathbf{Pr}[X = x] - \mathbf{Pr}[Y = x]|$.

2. *(Restatement of Definition 17.30)* $\delta(X,Y) \geq \epsilon$ *iff there is a boolean function* $D : \{0,1\}^m \to \{0,1\}$ *such that* $|\Pr_{x\in X}[D(x)=1] - \Pr_{y\in Y}[D(y)=1]| \geq \epsilon$.

3. *If* $f : \{0,1\}^n \to \{0,1\}^s$ *is any function, then* $\delta(f(X), f(Y)) \leq \delta(X,Y)$. *(Here* $f(X)$ *is a distribution on* $\{0,1\}^s$ *obtained by taking a sample of* X *and applying* f*.)*

Now we define an extractor. This is a (deterministic) function that transforms an (N,k) source into an almost uniform distribution. It uses a small number of additional truly random bits, denoted by t in the definition below.

Definition 17.32
A function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is a (k,ϵ) extractor if for any (n,k)-source X, the distribution $\mathsf{Ext}(X, U_t)$ is ϵ-close to U_m. (For every ℓ, U_ℓ denotes the uniform distribution over $\{0,1\}^\ell$.)

Equivalently, if $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is a (k,ϵ) extractor, then for every distribution X ranging over $\{0,1\}^n$ of min-entropy k, and for every $S \subseteq \{0,1\}^m$, we have

$$|\mathbf{Pr}_{a\in X, z\in\{0,1\}^t}[\mathsf{Ext}(a,z) \in S] - \mathbf{Pr}_{r\in\{0,1\}^m}[r \in S]| \leq \epsilon$$

We use this fact to show in Section 17.5.2 how to use extractors and (n,k)-sources to to simulate any probabilistic computation.

Why an additional input? Our stated motivation for extractors is to execute probabilistic algorithms without access to perfect unbiased coins. Yet, it seems that an extractor is not sufficient for this task, as we only guarantee that its output is close to uniform if it is given an additional input that is uniformly distributed. First, we note that the requirement of an additional input is necessary: for every function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ and every $k \leq n-1$ there exists an (n,k)-source X such that the first bit of $\mathsf{Ext}(X)$ is constant (i.e, is equal to some value $b \in \{0,1\}$ with probability 1), and so is at least of statistical distance $1/2$ from the uniform distribution (Exercise 7). Second, if the length t of the second input is sufficiently short (e.g., $t = O(\log n)$) then, for the purposes of simulating probabilistic algorithms, we can do without any access to true random coins, by enumerating over all the 2^t possible inputs (see Section 17.5.2). Clearly, t has to be somewhat short for the extractor to be non-trivial: for $t \geq m$, we can have a trivial extractor that ignores its first input and outputs the second input. This second input is called the *seed* of the extractor.

17.4.1 Extractors based upon hash functions

One can use pairwise independent (and even weaker notions of) hash functions to obtain extractors. In this section, $\mathcal{H}$ denotes a family of hash functions $h : \{0,1\}^n \to \{0,1\}^k$. We say it has *collision error* δ if for any $x_1 \neq x_2 \in \{0,1\}^n$, $\Pr_{h\in\mathcal{H}}[h(x_1) = h(x_2)] \leq (1+\delta)/2^k$. We assume that one can choose a random function $h \in \mathcal{H}$ by picking a string at random from $\{0,1\}^t$. We define the extractor $\mathsf{Ext}\colon \times \{0,1\}^t \to \{0,1\}^{k+t}$ as follows:

$$\mathsf{Ext}(x,h) = h(x) \circ h, \tag{17}$$

where $\circ$ denotes concatenation of strings.

To prove that this is an extractor, we relate the min-entropy to the collision probability of a distribution, which is defined as $\sum_a p_a^2$, where p_a is the probability assigned to string a.

Lemma 17.33
If a distribution X has min-entropy at least k then its collision probability is at most $1/2^k$.

Proof: For every a in X's range, let p_a be the probability that $X = a$. Then, $\sum_a p_a^2 \leq \max_a \{p_a\} \left(\sum_a p_a\right) \leq \frac{1}{2^k} \cdot 1 = \frac{1}{2^k}$. ■

Lemma 17.34 (Leftover hash lemma)
If x is chosen from a distribution on $\{0,1\}^n$ with min-entropy at least k/δ and $\mathcal{H}$ has collision error δ, then $h(X) \circ h$ has distance at most $\sqrt{2\delta}$ to the uniform distribution.

Proof: Left as exercise. (Hint: use the relation between the L_2 and L_1 norms ■

17.4.2 Extractors based upon random walks on expanders

This section assumes knowledge of random walks on expanders, as described in Chapter 16.

Lemma 17.35
Let $\epsilon > 0$. For every n and $k \leq n$ there exists a (k,ϵ)-extractor Ext : $\{0,1\}^n \times \{0,1\}^t \to \{0,1\}^n$ where $t = O(n - k + \log 1/\epsilon)$.

Proof: Suppose X is an (n,k)-source and we are given a sample a from it. Let G be a $(2^n, d, 1/2)$-graph for some constant d (see Definition 16.9 and Theorem 16.24).

Let z be a truly random seed of length $t = 10 \log d(n - k + \log 1/\epsilon) = O(n - k + \log 1/\epsilon)$. We interpret z as a random walk in G of length $10(n - k + \log 1/\epsilon)$ starting from the node whose label is a. (That is, we think of z as $10(n - k + \log 1/\epsilon)$ labels in $[d]$ specifying the steps taken in the walk.) The output $\mathsf{Ext}(a, z)$ of the extractor is the label of the final node on the walk.

We have $\|X - \mathbf{1}\|_2^2 \leq \|X\|_2^2 = \sum_a \Pr[X = a]^2$, which is at most 2^{-k} by Lemma 17.33 since X is an (n, k)-source. Therefore, after a random walk of length t the distance to the uniform distribution is (by the upperbound in (**??**)):

$$\|M^t X - \frac{1}{2^N}\mathbf{1}\|_1 \leq \lambda_2^t \|X - \frac{1}{2^N}\mathbf{1}\|_2 \sqrt{2^N} \leq \lambda_2^t 2^{(N-k)/2}.$$

When t is a sufficiently large multiple of $N - k + \log 1/\varepsilon$, this distance is smaller than ε. ■

17.4.3 An extractor based upon Nisan-Wigderson

Now we describe an elegant construction of extractors due to Trevisan.

Suppose we are given a string x obtained from an (N, k)-source. How can we extract k random bits from it, given $O(\log N)$ truly random bits? Let us check that the trivial idea fails. Using $2 \log N$ random bits we can compute a set of k (where $k < N - 1$) indices that are uniformly distributed and pairwise independent. Maybe we should just output the corresponding bits of x? Unfortunately, this does not work: the source is allowed to set $N - k$ bits (deterministically) to 0 so long as the remaining k bits are completely random. In that case the expected number of random bits in our sample is at most k^2/N, which is less than even 1 if $k < \sqrt{N}$.

This suggests an important idea: we should first apply some transformation on x to "smear out" the randomness, so it is not localized in a few bit positions. For this, we will use error-correcting codes. Recall that such codes are used to introduce error-tolerance when transmitting messages over noisy channels. Thus intuitively, the code must have the property that it "smears" every bit of the message all over the transmitted message.

Having applied such an encoding to the weakly random string, the construction selects bits from it using a better sampling method than pairwise independent sampling, namely, the Nisan-Wigderson combinatorial design.

Nisan-Wigderson as a sampling method:

In (**??**) we defined a function $NW_{f,\mathcal{S}}(z)$ using any function $f : \{0,1\}^l \to \{0,1\}$ and a combinatorial design $\mathcal{S}$. Note that the definition works for every function, not just hard-to-compute functions. Now we observe that $NW_{f,\mathcal{S}}(z)$ is actually a way to sample entries from the truth table of f.

Think of f as a bitstring of length 2^l, namely, its truth table. (Likewise, we can think of any circuit with l-bit inputs and with 0/1 outputs as computing a string of length 2^l.) Given any z ("the seed"), $NW_{f,\mathcal{S}}(z)$ is just a method to use z to sample a sequence of m bits from f. This is completely analogous to pairwise independent sampling considered above; see Figure **??**.

Figure unavailable in pdf file.

Figure 17.3: Nisan-Wigderson as a sampling method: An (l, α)-design $(S_1, S_2, \ldots, S_m)$ where each $S_i \subseteq [t], |S_i| = l$ can be viewed as a way to use $z \in \{0,1\}^t$ to sample m bits from any string of length 2^l, which is viewed as the truth table of a function $f : \{0,1\}^l \to \{0,1\}$.

List-decodable codes

The construction will use the following kind of codes.

Definition 17.36
If $\delta > 0$, a mapping $\sigma : \{0,1\}^N \to \{0,1\}^{\bar{N}}$ is called *an error-correcting code that is list-decodable up to error* $1/2 - \delta$ if for every $w \in \{0,1\}^{\bar{N}}$, the number of $y \in B^N$ such that $w, \sigma(y)$ disagree in at most $1/2 - \delta$ fraction of bits is at most $1/\delta^2$.

The set $\left\{\sigma(x) : x \in \{0,1\}^N\right\}$ is called the set of *codewords*.

The name "list-decodable" owes to the fact that if we transmit x over a noisy channel after first encoding with σ then even if the channel flips $1/2 - \delta$ fraction of bits, there is a small "list" of y that the received message could be decoded to. (Unique decoding may not be possible, but this will be of no consequence in the construction below.) The exercises ask you to prove that list-decodable codes exist with $\bar{N} = \text{poly}(N, 1/\delta)$, where σ is computable in polynomial time.

Trevisan's extractor:

Suppose we are given an (N, k)-source. We fix $\sigma : \{0,1\}^N \to \{0,1\}^{\bar{N}}$, a polynomial-time computable code that is list-decodable upto to error $1/2 - \epsilon/m$. We assume that $\bar{N}$ is a power of 2 and let $l = \log_2 \bar{N}$. Now every string $x \in \{0,1\}^{\bar{N}}$ may be viewed as a boolean function $< x >: \{0,1\}^{\log \bar{N}} \to \{0,1\}$ whose truth table is x. Let $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$.

The extractor $ExtNW : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is defined as

$$ExtNW_{\sigma,\mathcal{S}}(x,z) = NW_{<\sigma(x)>,\mathcal{S}}(z) \ .$$

That is, $ExtNW$ encodes its first ("weakly random") input x using an error-correcting code, then uses Nisan-Wigderson sampling on the resulting string using the second ("truly random") input z as a seed.

Lemma 17.37
For sufficiently large m and for $\epsilon > 2^{-m^2}$, $ExtNW_{\sigma,\mathcal{S}}$ is a $(m^3, 2\epsilon)$-extractor.

Proof: Let X be an (N,k) source where the min-entropy k is m^3. To prove that the distribution $ExtNW(a,z)$ where $a \in X, z \in \{0,1\}^t$ is close to uniform, it suffices (see our remarks after Definition 17.30) to show for each function $D : \{0,1\}^m \to \{0,1\}$ that

$$\left|\mathbf{Pr}_r[D(r) = 1] - \mathbf{Pr}_{a\in X, z\in\{0,1\}^t}[D(ExtNW(a,z)) = 1]\right| \leq 2\epsilon. \qquad (18)$$

For the rest of this proof, we fix an arbitrary D and prove that (18) holds for it.

The role played by this test D is somewhat reminiscent of that played by the distinguisher algorithm in the definition of a pseudorandom generator, except, of course, D is allowed to be arbitrarily inefficient. This is why we will use the black-box version of the Nisan-Wigderson analysis (Corollary **??**), which does not care about the complexity of the distinguisher.

Let B be the set of bad a's for this D, where string $a \in X$ is *bad for D* if

$$\left|\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z)) = 1]\right| > \epsilon.$$

We show that B is small using a counting argument: we exhibit a 1-1 mapping from the set of bad a's to another set G, and prove G is small. Actually, here is G:

$$G = \left\{\text{circuits of size } O(m^2)\right\} \times \{0,1\}^{2\log(m/\epsilon)} \times \{0,1\}^2 .$$

The number of circuits of size $O(m^2)$ is $2^{O(m^2 \log m)}$, so $|G| \leq 2^{O(m^2\log m)} \times 2(m/\epsilon)^2 = 2^{O(m^2 \log m)}$.

Let us exhibit a 1-1 mapping from B to G. When a is bad, Corollary **??** implies that there is a circuit C of size $O(m^2)$ such that either the circuit $D(C())$ or its negation –XORed with some fixed bit b—agrees with $\sigma(a)$ on a fraction $1/2 + \epsilon/m$ of its entries. (The reason we have to allow either $D(C())$ or its complement is the $|\cdot|$ sign in the statement of Corollary **??**.) Let $w \in \{0,1\}^{\bar{N}}$ be the string computed by this circuit. Then $\sigma(a)$ disagrees with w in at most $1/2 - \epsilon/m$ fraction of bits. By the assumed property of the code σ, at most $(m/\epsilon)^2$ other codewords have this property. Hence a is completely specified by the following information: (a) circuit C; this is specified by $O(m^2 \log m)$ bits (b) whether to use $D(C())$ or its complement to compute w, and also the value of the unknown bit b; this is specified by 2 bits (c) which of the $(m/\epsilon)^2$ codewords around w to pick as $\sigma(a)$; this is specified by $\lceil 2\log(m/\epsilon)\rceil$ bits assuming the codewords around w are ordered in some canonical way. Thus we have described the mapping from B to G.

We conclude that for any fixed D, there are at most $2^{O(m^2 \log m)}$ bad strings. The probability that an element a taken from X is bad for D is (by Lemma **??**) at most $2^{-m^3} \cdot 2^{O(m^2 \log m)} < \epsilon$ for sufficiently large m. We then have

$$\begin{aligned}
& \left|\mathbf{Pr}_r[D(r)=1] - \mathbf{Pr}_{a\in X, z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \\
\le\ & \sum_a \mathbf{Pr}[X=a] \left|\mathbf{Pr}[D(r)=1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \\
\le\ & \mathbf{Pr}[X\in B] + \epsilon \le 2\epsilon,
\end{aligned}$$

where the last line used the fact that if $a \notin B$, then by definition of B, $\left|\mathbf{Pr}[D(r)=1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \le \epsilon$. ■

The following theorem is an immediate consequence of the above lemma.

Theorem 17.38
Fix a constant ϵ; for every N and $k = N^{\Omega(1)}$ there is a polynomial-time computable (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ where $m = k^{1/3}$ and $t = O(\log N)$.

17.5 Applications of Extractors

Extractors are deterministic objects with strong pseudorandom properties. We describe a few important uses for them; many more will undoubtedly be found in future.

17.5.1 Graph constructions

An extractor is essentially a graph-theoretic object; see Figure **??**. (In fact, extractors have been used to construct expander graphs.) Think of a (k,ϵ) extractor $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ as a bipartite graph whose left side contains one node for each string in $\{0,1\}^N$ and the right side contains a node for each string in $\{0,1\}^m$. Each node a on the left is incident to 2^t edges, labelled with strings in $\{0,1\}^t$, with the right endpoint of the edge labeled with z being $\mathsf{Ext}(a,z)$.

An (N,k)-source corresponds to any distribution on the left side with min-entropy at least k. The extractor's definition implies that picking a node according to this distribution and a random outgoing edge gives a node on the right that is essentially uniformly distributed.

Figure unavailable in pdf file.

Figure 17.4: An extractor $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^T \to \{0,1\}^m$ defines a bipartite graph where every node on the left has degree 2^T.

This implies in particular that for every set X on the left side of size exactly 2^k —notice, this is a special case of an (N,k)-source— its neighbor set $\Gamma(X)$ on the right satisfies $|\Gamma(X)| \geq (1-\epsilon)2^m$.

One can in fact show a converse, that high expansion implies that the graph is an extractor; see Chapter notes.

17.5.2 Running randomized algorithms using weak random sources

We now describe how to use extractors to simulate probabilistic algorithms using weak random sources. Suppose that $A(\cdot,\cdot)$ is a probabilistic algorithm that on an input of length n uses $m = m(n)$ random bits, and suppose that for every x we have $\mathbf{Pr}_r[A(x,r) = \text{ right answer }] \geq 3/4$. If A's answers are 0/1, then such algorithms can be viewed as defining a **BPP** language, but here we allow a more general scenario. Suppose $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a $(k, 1/4)$-extractor.

Consider the following algorithm A': on input $x \in \{0,1\}^n$ and given a string $a \in \{0,1\}^N$ from the weakly random source, the algorithm enumerates all choices for the seed z and computes $A(x, \mathsf{Ext}(a,z))$. Let

$$A'(x,a) = \text{majority value of } \left\{A(x,\mathsf{Ext}(a,z)) : z \in \{0,1\}^t\right\} \qquad (19)$$

The running time of A' is approximately 2^t times that of A. We show that if a comes from an $(n, k+2)$ source, then A' outputs the correct answer with probability at least 3/4.

Fix the input x. Let $R = \{r \in \{0,1\}^m : A(x,r) = \text{ right answer }\}$, and thus $|R| \geq \frac{3}{4}2^m$. Let B be the set of strings $a \in \{0,1\}^N$ for which the majority answer computed by algorithm A' is incorrect, namely,

$$\begin{aligned} B &= \left\{ a : \mathbf{Pr}_{z\in\{0,1\}^t}[A(x, \mathsf{Ext}(a,z)) = \text{ right answer}] < 1/2 \right\} \\ &= \left\{ a : \mathbf{Pr}_{z\in\{0,1\}^t}[\mathsf{Ext}(a,z) \in R] < 1/2 \right\} \end{aligned}$$

CLAIM: $|B| \leq 2^k$.
Let random variable Y correspond to picking an element uniformly at random from B. Thus Y has min-entropy $\log B$, and may be viewed as a $(N, \log B)$-source. By definition of B,

$$\mathbf{Pr}_{a\in Y, z\in\{0,1\}^t}[\mathsf{Ext}(a,z) \in R] < 1/2.$$

But $|R| = \frac{3}{4}2^m$, so we have

$$\left| \mathbf{Pr}_{a\in Y, z\in\{0,1\}^t}[\mathsf{Ext}(a,z) \in R] - \mathbf{Pr}_{r\in\{0,1\}^m}[r \in R] \right| > 1/4,$$

which implies that the statistical distance between the uniform distribution and $\mathsf{Ext}(Y, z)$ is at least $1/4$. Since Ext is a $(k, 1/4)$-extractor, Y must have min-entropy less than k. Hence $|B| \leq 2^k$ and the Claim is proved.

The correctness of the simulation now follows since

$$\begin{aligned} \mathbf{Pr}_{a\in X}[A'(x,a) = \text{ right answer }] &= 1 - \mathbf{Pr}_{a\in X}[a \in B] \\ &\geq 1 - 2^{-(k+2)} \cdot |B| \geq 3/4, \quad \text{(by Lemma ??)}. \end{aligned}$$

Thus we have shown the following.

THEOREM 17.39
Suppose A is a probabilistic algorithm running in time $T_A(n)$ and using $m(n)$ random bits on inputs of length n. Suppose we have for every $m(n)$ a construction of a $(k(n), 1/4)$-extractor $\mathsf{Ext}_n : \{0,1\}^N \times \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$ running in $T_E(n)$ time. Then A can be simulated in time $2^t(T_A + T_E)$ using one sample from a $(N, k+2)$ source.

17.5.3 Recycling random bits

We addressed the issue of recycling random bits in Section **??**. An extractor can also be used to recycle random bits. (Thus it should not be surprising that random walks on expanders, which were used to recycle random bits in Section **??**, were also used to construct extractors above.)

Suppose A be a randomized algorithm that uses m random bits. Let $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^{t)} \to \{0,1\}^m$ be any (k, ϵ)-extractor. Consider the following algorithm. Randomly pick a string $a \in \{0,1\}^N$, and obtain 2^t strings in $\{0,1\}^m$ obtained by computing $\mathsf{Ext}(a,z)$ for all $z \in \{0,1\}^t$. Run A for all these random strings. Note that this manages to run A as many as 2^t times while using only N random bits. (For known extractor constructions, $N \ll 2^t m$, so this is a big saving.)

Now we analyse how well the error goes down. Suppose $D \subseteq \{0,1\}^m$ be the subset of strings for which A gives the correct answer. Let $p = |D|/2^m$; for a **BPP** algorithm $p \geq 2/3$. Call an $a \in \{0,1\}^N$ *bad* if the above algorithm sees the correct answer for less than $p - \epsilon$ fraction of z's. If the set of all bad a's were to have size more than 2^k, the (N,k)-source X corresponding to drawing uniformly at random from the bad a's would satisfy

$$\Pr[\mathsf{Ext}(X,U_t) \in D] - \Pr[U_m \in D] > \epsilon,$$

which would contradict the assumption that Ext is a (k,ϵ)-extractor. We conclude that the probability that the above algorithm gets an incorrect answer from A in $p - \epsilon$ fraction of the repeated runs is at most $2^k/2^N$.

17.5.4 Pseudorandom generators for spacebounded computation

Now we describe Nisan's pseudo-random generators for space-bounded randomized computation, which allows randomized logspace computations to be run with $O(\log^2 n)$ random bits.

Throughout this section we represent logspace machines by their *configuration graph*, which has size poly(n).

Theorem 17.40 (Nisan)
For every d there is a $c > 0$ and a polynomial-time computable function $g : \{0,1\}^{c\log^2 n} \to \{0,1\}^{n^d}$ such that for every space-bounded machine M that has a configuration graph of size $\leq n^d$ on inputs of size n:

$$\left| \Pr_{r\in\{0,1\}^{n^d}}[M(x,r)=1] - \Pr_{z\in\{0,1\}^{c\log^2 n}}[M(x,g(z))=1] \right| < \frac{1}{10}. \qquad (20)$$

We give a proof due to Impagliazzo, Nisan, and Wigderson [**?**] (with further improvements by Raz and Reingold [**?**]) that uses extractors. Nisan's original paper did not explicitly use extractors —the definition of extractors came later and was influenced by results such as Nisan's.

In fact, Nisan's construction proves a result stronger than Theorem 17.40: there is a polynomial-time simulation of every algorithm in **BPL** using

$O(\log^2 n)$ space. (See Exercises.) Note that Savitch's theorem (Theorem **??**) also implies that $\mathbf{BPL} \subseteq \mathbf{SPACE}(\log^2 n)$, but the algorithm in Savitch's proof takes $n^{\log n}$ time. Saks and Zhou [?] improved Nisan's ideas to show that $\mathbf{BPL} \subseteq \mathbf{SPACE}(\log^{1.5} n)$, which leads many experts to conjecture that $\mathbf{BPL} = \mathbf{L}$ (i.e., randomness does not help logspace computations at all). (For partial progress, see Section **??** later.)

The main intuition behind Nisan's construction —and also the conjecture $\mathbf{BPL} = \mathbf{L}$— is that the logspace machine has one-way access to the random string and only $O(\log n)$ bits of memory. So it can only "remember" $O(\log n)$ of the random bits it has seen. To exploit this we will use the following simple lemma, which shows how to recycle a random string about which only a little information is known. (Throughout this section, $\circ$ denotes concatenation of strings.)

LEMMA 17.41 (RECYCLING LEMMA)
Let $f : \{0,1\}^n \to \{0,1\}^s$ be any function and $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ *be a $(k, \epsilon/2)$-extractor, where $k = n - (s+1) - \log \frac{1}{\epsilon}$. When $X \in_R \{0,1\}^n$, $W \in_R \{0,1\}^m$, $z \in_R \{0,1\}^t$, then*

$$f(X) \circ W \approx_\epsilon f(X) \circ \mathsf{Ext}(X, z).$$

REMARK 17.42
When the lemma is used, $s \ll n$ and $n = m$. Thus $f(X)$, which has length s, contains only a small amount of information about X. The Lemma says that using an appropriate extractor (whose random seed can have length as small as $t = O(s + \log(1/\epsilon))$ if we use Lemma 17.35) we can get a new string $\mathsf{Ext}(X, z)$ that looks essentially random, even to somebody who knows $f(X)$.

PROOF: For $v \in \{0,1\}^s$ we denote by X_v the random variable that is uniformly distributed over the set $f^{-1}(v)$. Then we can express $\parallel (f(X) \circ W - f(X) \circ \mathsf{Ext}(X, z) \parallel$ as

$$= \frac{1}{2} \sum_{v,w} \left| \Pr[f(X) = v \wedge W = w] - \Pr_z[f(X) = v \wedge \mathsf{Ext}(X, z) = w] \right|$$

$$= \sum_v \Pr[f(X) = v] \cdot \parallel W - \mathsf{Ext}(X_v, z) \parallel \qquad (21)$$

Let $V = \left\{ v : \Pr[f(X) = v] \geq \epsilon/2^{s+1} \right\}$. If $v \in V$, then we can view X_v as a (n, k)-source, where $k \geq n - (s+1) - \log \frac{1}{\epsilon}$. Thus by definition of an extractor, $\mathsf{Ext}(X_v, r) \approx_{\epsilon/2} W$ and hence the contributions from $v \in V$ sum to at most $\epsilon/2$. The contributions from $v \notin V$ are upperbounded by $\sum_{v \notin V} \Pr[f(X) = v] \leq 2^s \times \frac{\epsilon}{2^{s+1}} = \epsilon/2$. The lemma follows. ■

Now we describe how the Recycling Lemma is useful in Nisan's construction. Let M be a logspace machine. Fix an input of size n and view the graph of all configurations of M on this input as a *leveled branching program.* For some $d \geq 1$, M has $\leq n^d$ configurations and runs in time $L \leq n^d$. Assume without loss of generality —since unneeded random bits can always be ignored— that it uses 1 random bit at each step. Without loss of generality (by giving M a separate worktape that maintains a time counter), we can assume that the configuration graph is leveled: it has L levels, with level i containing configurations obtainable at time i. The first level contains only the start node and the last level contains two nodes, "accept" and "reject;" every other level has $W = n^d$ nodes. Each level i node has two outgoing edges to level $i+1$ nodes and the machine's computation at this node involves using the next bit in the random string to pick one of these two outgoing edges. We sometimes call L the *length* of the configuration graph and W the *width.*

Figure unavailable in pdf file.

Figure 17.5: Configuration graph for machine M

For simplicity we first describe how to reduce the number of random bits by a factor 2. Think of the L steps of the computation as divided in two halves, each consuming $L/2$ random bits. Suppose we use some random string X of length $L/2$ to run the first half, and the machine is now at node v in the middle level. The only information known about X at this point is the index of v, which is a string of length $d \log n$. We may thus view the first half of the branching program as a (deterministic) function that maps $\{0,1\}^{L/2}$ bits to $\{0,1\}^{d \log n}$ bits. The Recycling Lemma allows us to use a random seed of length $O(\log n)$ to recycle X to get an almost-random string $\mathsf{Ext}(X, z)$ of length $L/2$, which can be used in the second half of the computation. Thus we can run L steps of computation using $L/2 + O(\log n)$ bits, a saving of almost a factor 2. Using a similar idea recursively, Nisan's generator runs L steps using $O(\log n \log L)$ random bits.

Now we formally define Nisan's generator.

Definition 17.43 (Nisan's generator)
For some $r > 0$ let $\mathsf{Ext}_k : \{0,1\}^{kr} \times \{0,1\}^r \to \{0,1\}^{kr}$ be an extractor function for each $k \geq 0$. For every integer $k \geq 0$ the associated Nisan generator G_k :

$\{0,1\}^{kr} \to \{0,1\}^{2^k}$ is defined recursively as (where $|a| = (k-1)r, |z| = r$)

$$G_k(a \circ z) = \begin{cases} z_1 \quad \text{(i.e., first bit of } z\text{)} & k = 1 \\ \\ G_{k-1}(a) \circ G_{k-1}(\mathsf{Ext}_{k-1}(a, z)) & k > 1 \end{cases}$$

Now we use this generator to prove Theorem 17.40. We only need to show that the probability that the machine goes from the start node to the "accept" node is similar for truly random strings and pseudorandom strings. However, we will prove a stronger statement involving intermediate steps as well.

If nodes u is a node in the configuration graph, and s is a string of length 2^k, then we denote by $f_{u,2^k}(s)$ the node that the machine reaches when started in u and its random string is s. Thus if s comes from some distribution $\mathcal{D}$, we can define a distribution $f_{u,2^k}(\mathcal{D})$ on nodes that are 2^k levels further from u.

THEOREM 17.44
Let $r = O(\log n)$ be such that for each $k \le d \log n$, $\mathsf{Ext}_k : \{0,1\}^{kr} \times \{0,1\}^r \to \{0,1\}^{kr}$ is a $(kr - 2d\log n, \epsilon)$-extractor. For every machine of the type described in the previous paragraphs, and every node u in its configuration graph:

$$\| f_{u,2^k}(U_{2^k}) - f_{u,2^k}(G_k(U_{kr})) \| \le 3^k \epsilon, \tag{22}$$

where U_l denotes the uniform distribution on $\{0,1\}^l$.

REMARK 17.45
To prove Theorem 17.40 let $u = u_0$, the start configuration, and $2^k = L$, the length of the entire computation. Choose $3^k\epsilon < 1/10$ (say), which means $\log 1/\epsilon = O(\log L) = O(\log n)$. Using the extractor of Section 17.4.2 as Ext_k, we can let $r = O(\log n)$ and so the seed length $kr = O(r \log L) = O(\log^2 n)$.

PROOF: (Theorem 17.44) Let ϵ_k denote the maximum value of the left hand side of (22) over all machines. The lemma is proved if we can show inductively that $\epsilon_k \le 2\epsilon_{k-1} + 2\epsilon$. The case $k = 1$ is trivial. At the inductive step, we need to upperbound the distance between two distributions $f_{u,2^k}(\mathcal{D}_1)$, $f_{u,2^k}(\mathcal{D}_4)$, for which we introduce two distributions $\mathcal{D}_2, \mathcal{D}_3$ and use triangle inequality:

$$\| f_{u,2^k}(\mathcal{D}_1) - f_{u,2^k}(\mathcal{D}_4) \| \le \sum_{i=1}^{3} \| f_{u,2^k}(\mathcal{D}_i) - f_{u,2^k}(\mathcal{D}_{i+1}) \| . \tag{23}$$

The distributions will be:

$\mathcal{D}_1 = U_{2^k}$
$\mathcal{D}_4 = G_k(U_{kr})$
$\mathcal{D}_2 = U_{2^{k-1}} \circ G_{k-1}(U_{(k-1)r})$
$\mathcal{D}_3 = G_{k-1}(U_{(k-1)r}) \circ G_{k-1}(U'_{(k-1)r}) \qquad (U, U'$ are identical but independent).

We bound the summands in (23) one by one.

Claim 1: $\| f_{u,2^k}(\mathcal{D}_1) - f_{u,2^k}(\mathcal{D}_2) \| \leq \epsilon_{k-1}$.
Denote $\Pr[f_{u,2^{k-1}}(U_{2^{k-1}}) = w]$ by $p_{u,w}$ and $\Pr[f_{u,2^{k-1}}(G_{k-1}(U_{(k-1)r})) = w]$ by $q_{u,w}$. According to the inductive assumption,

$$\frac{1}{2}\sum_w |p_{u,w} - q_{u,w}| = \| f_{u,2^{k-1}}(U_{2^{k-1}}) - f_{u,2^{k-1}}(G_{k-1}(U_{(k-1)r})) \| \leq \epsilon_{k-1}.$$

Since $\mathcal{D}_1 = U_{2^k}$ may be viewed as two independent copies of $U_{2^{k-1}}$ we have

$$\| f_{u,2^k}(\mathcal{D}_1) - f_{u,2^k}(\mathcal{D}_2) \| = \sum_v \frac{1}{2}\left|\sum_w p_{uw}p_{wv} - \sum_w p_{uw}q_{wv}\right|$$

where w, v denote nodes 2^{k-1} and 2^k levels respectively from u

$$= \sum_w p_{uw} \frac{1}{2}\sum_v |p_{wv} - q_{wv}|$$
$$\leq \epsilon_{k-1} \quad \text{(using inductive hypothesis and } \sum_w p_{uw} = 1)$$

Claim 2: $\| f_{u,2^k}(\mathcal{D}_2) - f_{u,2^k}(\mathcal{D}_3) \| \leq \epsilon_{k-1}$.

The proof is similar to the previous case.

Claim 3: $\| f_{u,2^k}(\mathcal{D}_3) - f_{u,2^k}(\mathcal{D}_4) \| \leq 2\epsilon$.
We use the Recycling Lemma. Let $g_u : \{0,1\}^{(k-1)r} \to [1, W]$ be defined as $g_u(a) = f_{u,2^{k-1}}(G_{k-1}(a))$. (To put it in words, apply the Nisan generator to the seed a and use the result as a random string for the machine, using u as the start node. Output the node you reach after 2^{k-1} steps.) Let $X, Y \in U_{(k-1)r}$ and $z \in U_r$. According to the Recycling Lemma,

$$g_u(X) \circ Y \approx_\epsilon g_u(X) \circ \mathsf{Ext}_{k-1}(X, z),$$

and then part 3 of Lemma 17.31 implies that the equivalence continues to hold if we apply a (deterministic) function to the second string on both sides. Thus

$$g_u(X) \circ g_w(Y) \approx_\epsilon g_u(X) \circ g_w(\mathsf{Ext}_{k-1}(X, z))$$

for all nodes w that are 2^{k-1} levels after u. The left distribution corresponds to $f_{u,2^k}(\mathcal{D}_3)$ (by which we mean that $\Pr[f_{u,2^k}(\mathcal{D}_3) = v] = \sum_w \Pr[g_u(X) = w \wedge g_w(Y) = v]$) and the right one to $f_{u,2^k}(\mathcal{D}_4)$ and the proof is completed. ■

Chapter notes and history

The results of this section have not been presented in chronological order and some important intermediate results have been omitted. Yao [?] first pointed out that cryptographic pseudorandom generators can be used to derandomize **BPP**. A short paper of Sipser [?] initiated the study of "hardness versus randomness," and pointed out the usefulness of a certain family of highly expanding graphs that are now called *dispersers* (they are reminiscent of extractors). This research area received its name as well as a thorough and brilliant development in a paper of Nisan and Wigderson [?]. MISSING DISCUSSION OF FOLLOWUP WORKS TO NW94

Weak random sources were first considered in the 1950s by von Neumann [?]. The second volume of Knuth's seminal work studies real-life pseudorandom generators and their limitations. The study of weak random sources as defined here started with Blum [?]. Progressively weaker models were then defined, culminating in the "correct" definition of an (N, k) source in Zuckerman [?]. Zuckerman also observed that this definition generalizes all models that had been studied to date. (See [?] for an account of various models considered by previous researchers.) He also gave the first simulation of probabilistic algorithms with such sources assuming $k = \Omega(N)$. A succession of papers has improved this result; for some references, see the paper of Lu, Reingold, Vadhan, and Wigderson [?], the current champion in this area (though very likely dethroned by the time this book appears).

The earliest work on extractors —in the guise of *leftover hash lemma* of Impagliazzo, Levin, and Luby [?] mentioned in Section 17.4.1— took place in context of cryptography, specifically, cryptographically secure pseudorandom generators. Nisan [?] then showed that hashing could be used to define provably good pseudorandom generators for logspace.

The notion of an extractor was first formalized by Nisan and Zuckerman [**?**]. Trevisan [**?**] pointed out that any "black-box" construction of a pseudorandom generator gives an extractor, and in particular used the Nisan-Wigderson generator to construct extractors as described in the chapter. His methodology has been sharpened in many other papers (e.g.,see [**?**]).

Our discussion of derandomization has omitted many important papers that successively improved Nisan-Wigderson and culminated in the result of Impagliazzo and Wigderson [**?**]that either **NEXP** = **BPP** (randomness is truly powerful!) or **BPP** has an a subexponential "simulation." [5] Such results raised hopes that we were getting close to at least a partial derandomization of **BPP**, but these hopes were dashed by the Impagliazzo-Kabanets [**?**] result of Section 17.3.

Trevisan's insight about using pseudorandom generators to construct extractors has been greatly extended. It is now understood that three combinatorial objects studied in three different fields are very similar: pseudorandom generators (cryptography and derandomization), extractors (weak random sources) and list-decodable error-correcting codes (coding theory and information theory). Constructions of any one of these objects often gives constructions of the other two. For a survey, see Vadhan's lecture notes [**?**].

Exercises

§1 Verify Corollary 17.6.

§2 Show that there exists a number $\epsilon > 0$ and a function $G : \{0,1\}^* \to \{0,1\}^*$ that satisfies all of the conditions of a $2^{\epsilon n}$-pseudorandom generator per Definition **??**, save for the computational efficiency condition.

Hint: show that if for every n, a random function mapping n bits to $2^{n/10}$ bits will have desired properties with high probabilities.

§3 Show by a counting argument (i.e., probabilistic method) that for every large enough n there is a function $f : \{0,1\}^n \to \{0,1\}$, such that $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n/10}$.

§4 Prove that if there exists $f \in \mathbf{E}$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{avg}}(f)(n) \geq 2^{\epsilon n}$ for every $n \in \mathbb{N}$, then **MA** = **NP**.

[5]The "simulation" is in quotes because it could fail on some instances, but finding such instances itself requires exponential computational power, which nature presumably does not have.

§5 We define an *oracle Boolean circuit* to be a Boolean circuit that have special gates with unbounded fanin that are marked ORACLE. For a Boolean circuit C and language $O \subseteq \{0,1\}^*$, we define by $C^O(x)$ the output of C on x, where the operation of the oracle gates when fed input q is to output 1 iff $q \in O$.

(a) Prove that if every $f \in \mathbf{E}$ can be computed by a polynomial-size circuits with oracle to SAT, then the polynomial hierarchy collapses.

(b) For a function $f : \{0,1\}^* \to \{0,1\}$ and $O \subseteq \{0,1\}^*$, define $\mathsf{H_{avg}}^O(f)$ to be the function that maps every $n \in \mathbb{N}$ to the largest S such that $\Pr_{x \in_R \{0,1\}^n}[C^O(x) = f(x)] \leq 1/2 + 1/S$.

§6 Prove Lemma 17.31.

§7 Prove that for every function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ and there exists an $(n, n-1)$-source X and a bit $b \in \{0,1\}$ such that $\Pr[\mathsf{Ext}(X)_1 = b] = 1$ (where $\mathsf{Ext}(X)_1$ denotes the first bit of $\mathsf{Ext}(X)$). Prove that this implies that $\delta(\mathsf{Ext}(X), U_m) \geq 1/2$.

§8 Show that there is a constant $c > 0$ such that if an algorithm runs in time T and requires m random bits, and $m > k + c \log T$, then it is not possible in general to simulate it in a blackbox fashion using an (N, k) source and $O(\log n)$ truly random bits.

Hint: For each source show that there is a randomized algorithm —it need not be efficient, since it is being used as a "black box"— for which the simulation fails.

§9 A *flat* (N, k) source is a (N, k) source where for every $x \in \{0,1\}^N$ p_x is either 0 or exactly 2^{-k}.

Show that a source X is an (N, k)-source iff it is a distribution on flat sources. In other words, there is a set of flat (N, k)-sources $X_1, X_2, \ldots$ and a distribution $\mathcal{D}$ on them such that drawing a sample of X corresponds to picking one of the X_i's according to $\mathcal{D}$, and then drawing a sample from X_i.

Hint: You need to view a distribution as a point in a 2^N-dimensional space, and show that X is in the convex hull of the points that represent all possible flat sources.

§10 Use Nisan's generator to give an algorithm that produces universal traversal sequences for n-node graphs (see Definition **??**) in $n^{O(\log n)}$-time and $O(\log^2 n)$ space.

§11 Suppose boolean function f is (S, ϵ)-hard and let D be the distribution on m-bit strings defined by picking inputs $x_1, x_2, \ldots, x_m$ uniformly at random and outputting $f(x_1)f(x_2)\cdots f(x_m)$. Show that the statistical distance between D and the uniform distribution is at most ϵm.

§12 Prove Lemma 17.34.

§13 (Klivans and van Melkebeek 1999) Suppose the conclusion of Lemma **??** is true. Then show that $\mathbf{MA} \subseteq \mathbf{i.o.}-[\mathbf{NTIME}(2^n)/n]$.

(Slightly harder) Show that if $\mathbf{NEXP} \neq \mathbf{EXP}$ then $\mathbf{AM} \subseteq \mathbf{i.o.}-[\mathbf{NTIME}(2^n)/n]$.

§14 Prove Lemma **??**.

Hint: What are the eigenvectors of G^l? You only need to identify n of them.

§15 Show that if S is any subset of at most half the vertices in a multigraph $G = (V, E)$ then the number of edges $\left|E(S, \overline{S})\right|$ going from S to $\overline{S}$ is at least $(1 - \lambda(G))\,|S|\,/2$.

Hint: Use the Courant-Fisher characterization. You will need to pick a particular vector that is orthogonal to the first eigenvector, which is $(1, 1, \ldots, 1)$.

Acknowledgements

We thank Luca Trevisan for cowriting an early draft of this chapter. Thanks also to Valentine Kabanets, Omer Reingold, and Iannis Tourlakis for their help and comments.