Computational Complexity: A Modern Approach

Sanjeev Arora and Boaz Barak
Princeton University

http://www.cs.princeton.edu/theory/complexity/

complexitybook@gmail.com

Chapter 20

Derandomization

"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

Randomization is an exciting and powerful paradigm in computer science and, as we saw in Chapter 7, often provides the simplest or most efficient algorithms for many computational problems. In fact, in some areas of Computer Science, such as distributed algorithms and cryptography, randomization is *proven* to be necessary to achieve certain tasks or achieve them efficiently. Thus it's natural to conjecture (as many scientists initially did) that at least for some problems, randomization is *inherently* necessary: one cannot replace the probabilistic algorithm with a deterministic one without a significant loss of efficiency. One concrete version of this conjecture would be that $\mathbf{BPP} \not\subseteq \mathbf{P}$ (see Chapter 7 for definition of **BPP**). Surprisingly, recent research has provided more and more evidence that this is likely *not* to hold. As we will see in this chapter, under very reasonable complexity assumptions, there is in fact a way to *derandomize* (i.e., transform into a deterministic algorithm) *every* probabilistic algorithm of the **BPP** type with only a polynomial loss of efficiency. Thus today most researchers believe that $\mathbf{BPP} = \mathbf{P}$. We note that this need not imply that randomness is useless in every setting —we already saw in Chapter 8 its crucial role in the definition of interactive proofs.

In Section 20.1 we define *pseudorandom generators*, which will serve as our main tool for derandomizing probabilistic algorithms. Their definition is a relaxation of the definition of *secure* pseudorandom generators in Chapter 9. This relaxation will allow us to construct such generators with better parameters and under weaker assumptions than what is possible for secure pseudorandom generators. In Section 20.2 we provide a construction of such pseudorandom generators under the assumptions that there exist explicit functions with high *average-case* circuit complexity. In Chapter 19 we show how to provide a construction that merely requires high *worst-case* circuit complexity.

While the circuit lower bounds we assume are widely believed to be true, they also seem to be very difficult to prove. This raises the question of whether assuming or proving such lower bounds is *necessary* to obtain derandomization. In Section 20.3 we show that it's possible to obtain at least a partial derandomization result based only on the assumption that $\mathbf{BPP} \neq \mathbf{EXP}$. Alas, as we show in Section 20.4, full derandomization of **BPP** will require proving circuit lower bounds.

Even though we still cannot prove sufficiently strong circuit lower bounds, just as in cryptography, we can use *conjectured* hard problems for derandomization instead of *provable* hard problems, and to a certain extent 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.

Example 20.1 *(Polynomial identity testing)*
We explain the notion of derandomization with an example. One algorithm that we would like to derandomize is the one described in Section 7.2.3 for testing if a given polynomial (represented in the form of an arithmetic circuit) 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 7.5). 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 how hard can it be to find a single one? 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 20.4 we will see that in fact obtaining such a set (or even any other deterministic algorithm for this problem) will imply some nontrivial circuit lower bounds.

20.1 Pseudorandom Generators and Derandomization

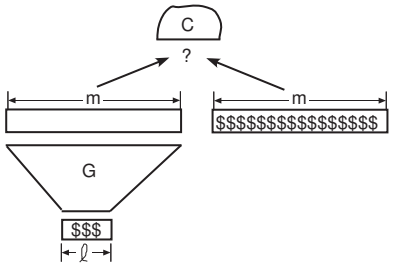
The main tool we will use for derandomization is a pseudorandom generator. This is a twist on the definition of a cryptographically *secure* pseudorandom generator we saw in Chapter 9, with the main difference that here we will allow the generator to run in *exponential* time (and in particular allow the generator more time than the distinguisher). Another difference is that we consider *non-uniform* distinguishers— in other words, circuits— rather than Turing machines, as was done in Chapter 9. (This second difference is not an essential one. As mentioned in the notes at the end of Chapter 9, we could have used circuits there as well.)

Definition 20.2 *(Pseudorandom generators)*
A distribution R over $\{0,1\}^m$ is (S,ϵ)-*pseudorandom* (for $S \in \mathbb{N}, \epsilon > 0$) if for every circuit C of size at most S,

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

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

Let $S : \mathbb{N} \to \mathbb{N}$ be some function. A 2^n-time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is an $S(\ell)$-*pseudorandom generator* if $|G(z)| = S(|z|)$ for every $z \in \{0,1\}^*$ and for every $\ell \in \mathbb{N}$ the distribution $G(U_\ell)$ is $(S(\ell)^3, 1/10)$-pseudorandom.



The choices of the constants 3 and 1/10 in the definition of an $S(\ell)$-pseudorandom generator are arbitrary and made for convenience. To avoid annoying cases, we will restrict our attention to $S(\ell)$-pseudorandom generators for functions $S : \mathbb{N} \to \mathbb{N}$ that are *time-constructible* and *non-decreasing* (i.e., $S(\ell') \geq S(\ell)$ for $\ell' \geq \ell$).

20.1.1 Derandomization using pseudorandom generators

The relation between pseudorandom generators and simulating probabilistic algorithms is rather straightforward:

Lemma 20.3 *Suppose that there exists an $S(\ell)$-pseudorandom generator for a time-constructible non-decreasing $S : \mathbb{N} \to \mathbb{N}$. 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.* ♢

Before proving Lemma 20.3 it is instructive to see what derandomization results it implies for various values of S. This is observed in the following simple corollary, left as Exercise 20.1:

Corollary 20.4

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^{ℓ^ϵ}-pseudorandom generator for some constant $\epsilon > 0$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP} = \mathbf{DTIME}(2^{\mathrm{polylog}(n)})$.*
3. *If for every $c > 1$ there exists an ℓ^c-pseudorandom generator then $\mathbf{BPP} \subseteq \mathbf{SUBEXP} = \cap_{\epsilon>0}\mathbf{DTIME}(2^{n^\epsilon})$.* ♢

Proof of Lemma 20.3: 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

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

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 such as 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 $2/3 - 0.1 > 0.5$. Thus to derandomize A, we do not need to enumerate over all $r \in \{0,1\}^m$: it suffices to enumerate over all the strings $G(z)$ for $z \in \{0,1\}^{\ell(n)}$ and check whether or not the majority of these make A accept. This derandomized algorithm runs in $2^{O(\ell(n))}$ time instead of the trivial 2^m time.

Now we make this formal. On input $x \in \{0,1\}^n$, our deterministic algorithm B will go over all $z \in \{0,1\}^{\ell(n)}$, compute $A(x, G(z))$ and output the majority answer.[1] We claim that for n sufficiently large, the fraction of z's such that $A(x, G(z)) = L(x)$ is at least $2/3 - 0.1$. 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 there exists a distinguisher for the pseudorandom generator: just use the Cook-Levin transformation (e.g., as in the proof of Theorem 6.6) to construct a circuit computing the function $r \mapsto A(x, r)$, where x is hardwired into the circuit. (This "hardwiring" is the place in the proof where we use non-uniformity.) This circuit has size $O(S(\ell(n)))^2$ which is smaller than $S(\ell(n))^3$ for sufficiently large n. ■

The proof of Lemma 20.3 shows why it is OK to allow the pseudorandom generator in Definition 20.2 to run in time exponential in its seed length. The reason is that the derandomized algorithm enumerates over all possible seeds of length ℓ, and thus would take exponential (in ℓ) time even if the generator itself were to run in less than $\exp(\ell)$ time.

[1] If $m < S(\ell(n))$ then $A(x, G(z))$ denotes the output of A on input x using the first m bits of $G(z)$ for its random choices.

Notice also that allowing the generator $\exp(\ell)$ time means that it has to "fool" distinguishers that run for *less* time than it does. By contrast, the definition of cryptographically *secure pseudorandom generators* (Definition 9.8 in Chapter 9) required the generator to run in some fixed polynomial time, and yet fool arbitrary polynomial-time distinguishers. 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, the generator is used by the derandomized algorithm and the "distinguisher" is the probabilistic algorithm that is being derandomized. In this case it is reasonable to allow the derandomized algorithm more running time than the original probabilistic algorithm. Of course, allowing the generator to run in exponential time potentially makes it easier to prove their existence compared with secure pseudorandom generators, and this indeed appears to be the case. If we relaxed the definition even further and made no efficiency requirements then showing the existence of such generators becomes almost trivial (see Exercise 20.2) but they no longer seem useful 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 $\mathbf{BPP} = \mathbf{P}$.

20.1.2 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, but using the results of Chapter 19 we will be able to also construct pseudorandom generators assuming only *worst-case* hardness. Both worst-case and average-case hardness refer to the size of the minimum Boolean circuit computing the function. Recall that we define the *average-case hardness* of a function $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{H}_{\mathsf{avg}}(f)$, to be the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < 1/2 + 1/S$ for every Boolean circuit C on n inputs with size at most S (see Definition 19.1). For $f : \{0,1\}^* \to \{0,1\}$, we let $\mathsf{H}_{\mathsf{avg}}(f)(n)$ denote the average-case hardness of the restriction of f to $\{0,1\}^n$.

Example 20.5

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

1. If $f : \{0,1\}^* \to \{0,1\}$ 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 20.3). In particular, with probability tending to 1 as n grows, both $\mathsf{H}_{\mathsf{wrs}}(f)(n)$ and $\mathsf{H}_{\mathsf{avg}}(f)(n)$ exceed $2^{0.99n}$.
2. If $f \in \mathbf{BPP}$ then, since $\mathbf{BPP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, 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, $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT}) \geq 2^{\Omega(n)}$. A weaker assumption is that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$, in which case $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT})$ is not bounded by any polynomial. The average case complexity of 3SAT for uniformly chosen inputs is unclear, and in any case is dependent upon the way we choose to represent formulae as strings.
4. Under widely believed cryptographic assumptions, $\mathbf{NP}$ contains functions that are hard on the average. If g is a one-way permutation (as defined in Chapter 9) that cannot be inverted with polynomial probability by polynomial-sized circuits, then by Theorem 9.12, the function f that maps the pair $x, r \in \{0,1\}^n$ to $g^{-1}(x) \odot r$ (where $x \odot r = \sum_{i=1}^{n} x_i r_i \pmod 2$) has super-polynomial *average-case* hardness: $\mathsf{H}_{\mathsf{avg}}(f) \geq n^{\omega(1)}$.

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

Theorem 20.6 *(PRGs from average-case hardness)*
Let $S : \mathbb{N} \to \mathbb{N}$ be time-constructible and non-decreasing. If there exists $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{avg}}(f)(n) \geq S(n)$ for every n then exists an $S(\delta\ell)^{\delta}$-pseudorandom generator for some constant $\delta > 0$.

Combining this result with Theorem 19.27 of Chapter 19 we obtain the following theorem that gives even stronger evidence (given the plethora of plausible hard functions mentioned above) for the conjecture that derandomizing probabilistic algorithms is possible:

Theorem 20.7 *(Derandomization under worst-case assumptions)*
Let $S : \mathbb{N} \to \mathbb{N}$ be time-constructible and non-decreasing. If there exists $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n then exists an $S(\delta\ell)^{\delta}$-pseudorandom generator for some constant $\delta > 0$. In particular, the following corollaries hold:

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

We can replace $\mathbf{E}$ with $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(n)})$ in Corollaries 2 and 3 above. Indeed, for every $f \in \mathbf{DTIME}(2^{n^c})$, let g be the function that on input $x \in \{0,1\}^*$ outputs f applied to the first $|x|^{1/c}$ bits of x. Then, g 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 $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n^{\delta}}$ then there there exists a constant $\delta' > 0$ and a function $g \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(g) \geq 2^{n^{\delta'}}$, 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.

Remark 20.8
Nisan and Wigderson [NW88] were the first to show a pseudorandom generator from average-case hardness, but they did not prove Theorem 20.6 as it is stated above. Rather, Theorem 20.6 was proven by Umans [Uma03] following a long sequence of works including [BFNW93, IW97, ISW99, STV99, SU01]. Nisan and Wigderson only proved that under the same assumptions there exists an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S(n)^{\delta}$ for some constant $\delta > 0$ and n satisfying $n \geq \delta\sqrt{\ell \log S(n)}$. Note that this bound is still sufficient to derive all three corollaries above. It is this weaker version we prove in this book.

20.2 Proof of Theorem 20.6: Nisan-Wigderson Construction

How can we use a hard function to construct a pseudorandom generator? As a warmup we start with two "toy examples". We first show how to use a hard function to construct a pseudorandom generator whose output is only a single bit longer than its input. Then

we show how to obtain such a generator whose output is two bits longer than the input. Of course, neither of these suffices to prove Theorem 20.6 but they do give insight into the connection between hardness and randomness.

20.2.1 Two toy examples

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

The following Lemma uses a hard function to construct a "toy" generator that extends its input by a single bit:

Lemma 20.9 *(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 is 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 $((\ell+1)^3, 1/10)$-pseudorandom we use Yao's Theorem from Chapter 9 showing that pseudorandomness is implied by unpredictability:[2]

Theorem 20.10 *(Theorem 9.11, restated) Let Y be a distribution over $\{0,1\}^m$. Suppose that there exist $S > 10n$ and $\epsilon > 0$ 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. ♢

By Theorem 20.10, to prove Lemma 20.9 it suffices 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

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

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 (2) 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 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 20.11 *(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

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

[2] Although this theorem was stated and proved in Chapter 9 for the case of *uniform* Turing machines, the proof extends to the case of circuits; see Exercise 20.5.

Again, the efficiency and output length requirements are clearly satisfied. To show $G(U_\ell)$ is $((\ell+2)^3, 1/10)$-pseudorandom, we again use Theorem 20.10, 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

$$\Pr_{r=G(U_\ell)}[C(r_1,\ldots,r_{i-1}) = r_i] > \frac{1}{2} + \frac{1}{20(\ell+2)}. \quad (3)$$

Once again, (3) 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 (3) cannot hold for $i = \ell/2+1$ for the same reason as in Lemma 20.9. For $i = \ell+2$, Equation (3) becomes:

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

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 (4) 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 $r \circ f(r)$ (as fixing r to some constant also fixes $f(r)$) to the circuit C and obtain a circuit D of size at most $2(\ell+2)^3 < (\ell/2)^4$ 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 of Lemma 20.11 to obtain a generator G that extends its input by k bits setting

$$G(z_1,\ldots,z_\ell) = z^1 \circ f(z^1) \circ z^2 \circ f(z^2) \cdots z^k \circ f(z^k), \quad (5)$$

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 in this way a generator that expands its input by a multiplicative factor larger than two. Note that to prove Theorem 20.6 we need a generator whose output might even be *exponentially larger* than the input! Clearly, we need a new idea.

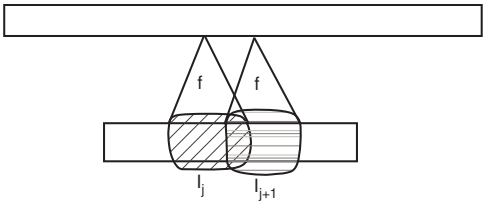
20.2.2 The NW Construction

The new idea is still inspired by the construction (5), 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* (non-disjoint pieces) 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.

Definition 20.12 *(NW Generator)*
Let $\mathcal{I} = \{I_1, \ldots, I_m\}$ be a family of subsets of $[\ell]$ with $|I_j| = n$ for every j, and let $f : \{0,1\}^n \to \{0,1\}$. The $(\mathcal{I}, f)$-*NW generator* is the function $\mathsf{NW}^f_{\mathcal{I}} : \{0,1\}^\ell \to \{0,1\}^m$ that maps every $z \in \{0,1\}^\ell$ to

$$\mathsf{NW}^f_{\mathcal{I}}(z) = f(z_{I_1}) \circ f(z_{I_2}) \cdots \circ f(z_{I_m}), \tag{6}$$

where for $z \in \{0,1\}^\ell$ and $I \subseteq [\ell]$, z_I denotes the restriction of z to the coordinates in I.



Conditions on the set systems and function.

We will see that in order for the generator to produce pseudorandom outputs, the function f must display some *hardness*, and the family of subsets should come from a *combinatorial design*, defined as follows:

Definition 20.13 *(Combinatorial designs)* Let d, n, ℓ satisfy $\ell > n > d$. 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 (whose proof we defer to the end of this section) yields sufficiently efficient constructions of such designs:

Lemma 20.14 *(Construction of designs) There is an algorithm A that on input $\langle \ell, d, n \rangle$ 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, then $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is indeed a pseudorandom distribution:

Lemma 20.15 *(Pseudorandomness using the NW generator) If $\mathcal{I}$ is an (ℓ, n, d)-design with $|\mathcal{I}| = 2^{d/10}$ and $f : \{0,1\}^n \to \{0,1\}$ satisfies $\mathsf{H}_{\mathsf{avg}}(f) > 2^{2d}$ then the distribution $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is $(\mathsf{H}_{\mathsf{avg}}(f)/10, 1/10)$-pseudorandom.* ♢

Proof: Let $S = \mathsf{H}_{\mathsf{avg}}(f)$. By Yao's Theorem, to show that $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is $(S/10, 1/10)$-pseudorandom it suffices to prove that for every $i \in [2^{d/10}]$ there does *not* exist an $S/2$-sized circuit C such that

$$\Pr_{\substack{Z \sim U_\ell \\ R = \mathsf{NW}^f_{\mathcal{I}}(Z)}} [C(R_1, \ldots, R_{i-1}) = R_i] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}. \tag{7}$$

For contradiction's sake, assume that (7) holds for some circuit C and some i. Plugging in the definition of $\mathsf{NW}^f_{\mathcal{I}}$, (7) becomes:

$$\Pr_{Z \sim U_\ell} [C(f(Z_{I_1}), \cdots, f(Z_{I_{i-1}})) = f(Z_{I_i})] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}. \tag{8}$$

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, (8) becomes:

$$\Pr_{\substack{Z_1 \sim U_n \\ Z_2 \sim U_{\ell-n}}} [C(f_1(Z_1, Z_2), \ldots, f_{i-1}(Z_1, Z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}, \tag{9}$$

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 (9) holds then there exists a string $z_2 \in \{0,1\}^{\ell-n}$ such that

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

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, one might worry that they together contain enough information about Z_1 and so a circuit *could potentially* predict $f_i(Z_1)$ after seeing all of them. To prove that this fear is baseless we use the fact that $\mathcal{I}$ is a design and f is a sufficiently hard function.

Since $|I_j \cap I_i| \leq d$ for $j \neq i$, the function $Z_1 \mapsto f_j(Z_1,z_2)$ (for the fixed string z_2) depends at most d coordinates of Z_1 and hence can be trivially computed by a $d2^d$-sized circuit (or even $O(2^d/d)$ sized circuit, see Exercise 6.1). Thus if (9) holds then there exists a circuit B of size $2^{d/10} \cdot d2^d + S/2 < S$ such that

$$\Pr_{Z_1 \sim U_n}[B(Z_1) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}. \tag{11}$$

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

The proof of Lemma 20.15 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 20.7 uses it to show that under plausible complexity assumptions, the complexity class **AM** (containing all languages with a constant round interactive proof, see Chapter 8) is equal to **NP**. We will also use this property in Chapter 21 to construct *randomness extractors* based on pseudorandom generators.

Putting it all together: Proof of Theorem 20.6 from Lemmas 20.14 and 20.15

As noted in Remark 20.8, we do not prove here Theorem 20.6 as stated but only the weaker statement, that given $f \in \mathbf{DTIME}(2^{O(n)})$ and $S : \mathbb{N} \to \mathbb{N}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq S$, we can construct an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S(n)^\epsilon$ for some $\epsilon > 0$ and n satisfying $n \geq \epsilon\sqrt{\ell \log S(n)}$. On input $z \in \{0,1\}^\ell$, our generator will operate as follows:

- Set n to be the largest number such that $\ell > \frac{100n^2}{\log S(n)}$. Thus, $\ell \leq \frac{100(n+1)^2}{\log S(n+1)} \leq \frac{200n^2}{\log S(n)}$, and hence $n \geq \sqrt{\ell \log S(n)/200}$.
- Set $d = \log S(n)/10$.
- Run the algorithm of Lemma 20.14 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 $\mathsf{NW}^f_{\mathcal{I}}(z)$.

This generator makes $2^{d/5}$ invocations of f, taking a total of $2^{O(n)+d}$ steps. By possibly reducing n by a constant factor, we can ensure the running time is bounded by 2^ℓ. Moreover, since $2^d \leq S(n)^{1/10}$, Lemma 20.15 implies that the distribution $\mathsf{NW}^f(U_\ell)$ is $(S(n)/10, 1/10)$-pseudorandom. ■

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 20.14 (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 the following condition **(*)**: $|I \cap I_j| \leq d$ for every $j \in [m]$.

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:

$$\Pr[|I| \geq n] \geq 0.9 \tag{12}$$

$$\text{For every } j \in [m], \Pr[|I \cap I_j| \geq d] \leq 0.5 \cdot 2^{-d/10} \tag{13}$$

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 (13) and (12) follow from the Chernoff bound. Yet, because $m \leq 2^{d/10}$, 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. ■

20.3 Derandomization under uniform assumptions

Circuit lower bounds are notoriously hard to prove. Despite decades of effort, at the moment we do not know of a single function in **NP** requiring more than $5n$-sized circuits to compute, not to mention super-linear or super-polynomial circuits. A natural question is whether such lower bounds are *necessary* to achieve derandomization.[3] Note that pseudorandom generators as in Definition 20.2 can be easily shown to imply circuit lower bounds: see Exercise 20.4. However, there could potentially be a different way to show **BPP** = **P** without constructing pseudorandom generators.

The following result shows that to some extent this is possible: one can get a non-trivial derandomization of **BPP** under a *uniform* hardness assumption. Namely, that **BPP** $\neq$ **EXP**.

Theorem 20.16 *(Uniform derandomization* [IW98]*)*
Suppose that **BPP** $\neq$ **EXP**. *Then for every* $L \in$ **BPP** *there exists a subexponential (i.e.,* $2^{n^{o(1)}}$*) time deterministic algorithm* A *such that for infinitely many* n*'s*

$$\Pr_{x \in_R \{0,1\}^n}[A(x) = L(x)] \geq 1 - 1/n\,.$$

This means that unless randomness is a panacea, and every problem with an exponential time algorithm (including 3SAT, TQBF, the permanent, etc..) can be solved in probabilistic polynomial time, we can at least partially derandomize **BPP**: obtain a subexponential deterministic simulation that succeeds well in the average-case. In fact, the conclusion of Theorem 20.16 can be considerably strengthened: we can find an algorithm A that will solve L with probability $1 - 1/n$ not only for inputs chosen according to the uniform distribution, but on inputs chosen according to *every* distribution that can be sampled in polynomial time. Thus, while this deterministic simulation may sometimes fail, it is hard to find inputs on which it does!

Proof sketch of Theorem 20.16: We only sketch the proof of Theorem 20.16 here. We start by noting that we may assume in the proof that **EXP** $\subseteq \mathbf{P}_{/\mathbf{poly}}$, since otherwise there

[3] Note that we do not know much better lower bounds for Turing machines either. However, a-priori it seems that a result such as **BPP** $\neq$ exp may be easier to prove than circuit lower bounds, and that a natural first step to proving such a result is to get derandomization results without assuming such lower bounds.

is some problem in **EXP** with superpolynomial circuit complexity, and such a problem can be used to build a pseudorandom generator that is strong enough to imply the conclusion of the Theorem. (This follows from Theorem 20.7 and Exercise 20.8.) Next, note that if $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then **EXP** is contained in the polynomial hierarchy (see Theorem 6.20 and also Lemma 20.18 below). But that implies $\mathbf{EXP} = \mathbf{PH}$ and hence we can conclude from Toda's and Valiant's theorems (Theorems 17.14 and 17.11) that the *permanent* function perm is **EXP**-complete under polynomial-time reductions. In addition, the Lemma hypothesis implies that perm is not in **BPP**. This is a crucial point in the proof since perm is a very special function that is downward self-reducible (see Chapter 8).

The next idea is to build a pseudorandom generator G with super-polynomial output size using the permanent as a hard function. We omit the details, but this can be done following the proofs of Theorems 19.27 and 20.6 (one needs to handle the fact that the permanent's output is not a single bit, but this can be handled for example using the Goldreich-Levin Theorem of Chapter 9). Looking at the proof of correctness for this pseudorandom generator G, it can be shown to yield an algorithm T to transform for every n a distinguisher D between G's output) and the uniform distribution into a polynomial-sized circuit C_n that computes perm_n (which this denotes the restriction of the permanent to length n inputs). This algorithm T is similar to the transformation shown in the proof of the standard NW generator (proof of Theorem 20.6): the only reason it is not efficient is that it requires computing the hard function (in this case the permanent) on several randomly chosen inputs, which are then "hardwired" into the distinguisher.[4]

Suppose for the sake of contradiction that the conclusion of Theorem 20.16 is false. This means that there is a probabilistic algorithm A whose derandomization using G fails with noticeable probability (over the choice of a random input) for all but finitely many input lengths. This implies that not only there is a sequence of polynomial-sized circuits $\{D_n\}$ distinguishing the output of G from the uniform distribution on all but finitely many input lengths, but in fact there is a probabilistic polynomial-time algorithm that on input 1^n will find such a circuit D_n with probability at least $1/n$ (Exercise 20.9). We now make the simplifying assumption that this probabilistic algorithm in facts finds such a distinguisher D_n with high probability, say at least $1 - 1/n^2$.[5] Plugging this into the proof of pseudorandomness for the generator G, this means that there exists a probabilistic polynomial-time algorithm T that can "learn" the permanent function: given oracle access to perm_n (the restriction of perm to length n inputs) the algorithm T runs in poly(n) time and produces a poly(n)-sized circuit computing perm_n.

But using T we can come up with a probabilistic polynomial-time algorithm for the permanent that doesn't use any oracle! To compute the permanent on length n inputs, we compute inductively the circuits $C_1, \ldots, C_n$. Given the circuit C_{n-1} we can compute the permanent on length n inputs using the permanent's *downward self-reducibility* property (see Section 8.6.2 and the proof of Lemma 20.19 below), and so implement the oracle to T that allows us to build the circuit C_n. Since we assumed $\mathbf{BPP} \neq \mathbf{EXP}$, and under $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ the permanent is **EXP**-complete, we get a contradiction.■

20.4 Derandomization requires circuit lower bounds

Section 20.3 shows that circuit lower bounds imply derandomization. However, circuit lower bounds have proved tricky, so one can hope that derandomization could somehow be done *without* circuit lower bounds. In this section we show this is not the case: proving that $\mathbf{BPP} = \mathbf{P}$ or even that a specific problem in **BPP** (namely the problem ZEROP of testing

[4] The proof of Theorem 20.6 only showed that there exists some inputs that when these inputs and their answers are"hardwired" into the distinguisher then we get a circuit computing the hard function. However, because the proof used the probabilistic method / averaging argument, it's not hard to show that with good probability random inputs will do.

[5] This gap can be handled using the fact that the permanent is a low-degree polynomial and hence has certain self-correction / self-testing properties, see sections 8.6.2 and 19.4.2.

whether a given polynomial is identically zero) will imply super-polynomial lower bounds for either Boolean or arithmetic circuits. 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** since once we do so we'll get "two for the price of one": both derandomization and circuit lower bounds.

Recall (Definition 16.7) that we say that a function f defined over the integers is in $\mathbf{AlgP}^{\mathbb{Z}}_{/\mathbf{poly}}$ (or just $\mathbf{AlgP}_{/\mathbf{poly}}$ for short) if f can be computed by a polynomial size algebraic circuit whose gates are labeled by $+$, $-$, and $\times$.[6] We let perm denote the problem of computing the permanent of matrices over the integers. Recall also the *Polynomial Identity Testing* (ZEROP) problem in which the input consists of a polynomial represented by an arithmetic circuit computing it and we have to decide if it is the identically zero polynomial (see Example 20.1 and Section 7.2.3). The problem ZEROP is in $\mathbf{coRP} \subseteq \mathbf{BPP}$ and we will show that if it is in $\mathbf{P}$ then some super-polynomial circuit lower bounds hold:

Theorem 20.17 *(Derandomization implies lower bounds* [KI03]*)*
If $\mathsf{ZEROP} \in \mathbf{P}$ *then either* $\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ *or* $\mathsf{perm} \notin \mathbf{AlgP}_{/\mathbf{poly}}$.

The Theorem is known to be true even if its hypothesis is relaxed to $\mathsf{ZEROP} \in \cap_{\delta>0}\mathbf{NTIME}(2^{n^\delta})$. Thus, even a derandomization of **BPP** to subexponential non-deterministic time would still imply super-polynomial circuit lower bounds. The proof of Theorem 20.17 relies on many results described earlier in the book.(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.) Our first ingredient is the following lemma:

Lemma 20.18 *(*[BFL90],[BFNW93]*)* $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{EXP} = \mathbf{MA}$. ♢

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 8.10).

PROOF OF LEMMA 20.18: Suppose $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$. By Meyer's Theorem (Theorem 6.20), in this case **EXP** collapses to the second level $\mathbf{\Sigma}^p_2$ of the polynomial hierarchy. Hence under our assumptions $\mathbf{\Sigma}^p_2 = \mathbf{PH} = \mathbf{PSPACE} = \mathbf{IP} = \mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$. Thus every $L \in \mathbf{EXP}$ has an interactive proof, and furthermore, since our assumption implies that $\mathbf{EXP} = \mathbf{PSPACE}$, we can just use the interactive proof for TQBF, for which the prover is a **PSPACE** machine and (given that we assume $\mathbf{PSPACE} \subseteq \mathbf{P}_{/\mathbf{poly}}$) can be replaced by a polynomial size circuit family $\{C_n\}$. Now we see that the interactive proof can actually be carried out in one round: given an input x of length n, Merlin will send Arthur a polynomial size circuit C, which is supposed to be circuit C_n for the prover's strategy for L. Then Arthur simulates the interactive proof for L, using C as the prover and tossing coins to simulate the verifier. Note that if the input is not in the language, then *no* prover has a decent chance of convincing the verifier, and in particular this holds for the 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 second lemma connects the complexity of identity testing and the permanent to the power of the class $\mathbf{P}^{\mathsf{perm}}$:

Lemma 20.19 *If* $\mathsf{ZEROP} \in \mathbf{P}$ *and* $\mathsf{perm} \in \mathbf{AlgP}_{/\mathbf{poly}}$ *then* $\mathbf{P}^{\mathsf{perm}} \subseteq \mathbf{NP}$. ♢

[6] The results below extend also to circuits that are allowed to work over the rational or real numbers and use division.

PROOF OF LEMMA 20.19: 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.

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 of Theorem 20.17 is the following lemma, which is interesting in its own right:

Lemma 20.20 *([IKW01])* $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{NEXP} = \mathbf{EXP}$. ♢

PROOF: We prove the contrapositive. That is, we assume that $\mathbf{NEXP} \neq \mathbf{EXP}$ and will prove thsat $\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$. Let $L \in \mathbf{NEXP} \setminus \mathbf{EXP}$ (such a language exists under our assumption). 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, say, time $2^{|x|^{10c}}$.

We now consider the following approach to try to solve L in exponential deterministic time. 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 $L \notin \mathbf{EXP}$, M_D does not solve L and hence 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 M_D can only make one-sided errors). 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 20.7), we can use such a string y to obtain an ℓ^D-pseudorandom generator. This method is called the "easy witness" method [Kab00], because it shows that unless the input x has a witness/certificate y (i.e., string satisfying $R(x,y) = 1$) that is "easy" in the sense that it can be computed by a small circuit, then any certificate for x can be used for derandomization.

Now, if $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ and then by Lemma 20.18 $\mathbf{EXP} \subseteq \mathbf{MA}$. That is, every language in **EXP** 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 for some constant D. 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 poly$(n^D)2^{n^{10c}}$ 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 an absolute constant $c > 0$ such that *every* language in **EXP** can be decided on infinitely many inputs by an $\mathbf{NTIME}(2^{n^c})$ time algorithm using n bits of advice, and hence (since we assume $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$) by a size $n^{c'}$ circuit family for an absolute constant c'. But using standard diagonalization we can easily come up with a language in $\mathbf{DTIME}(2^{O(n^{c'})}) \subseteq \mathbf{EXP}$ that cannot be computed by such a circuit family on almost every input. ■

It might seem that Lemma 20.20 should have an easier proof that goes along the lines of the proof of Lemma 20.18 ($\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{EXP} = \mathbf{MA}$) 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 20.17.

Proof of Theorem 20.17: For contradiction's sake, assume that the following are all true:

$$\mathsf{ZEROP} \in \mathbf{P} \tag{14}$$

$$\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}, \tag{15}$$

$$\mathsf{perm} \in \mathbf{AlgP}_{/\mathbf{poly}}. \tag{16}$$

Statement (15) together with Lemmas 20.18 and 20.20 imply that $\mathbf{NEXP} = \mathbf{EXP} = \mathbf{MA}$. Now recall that $\mathbf{MA} \subseteq \mathbf{PH}$, and that by Toda's Theorem (Theorem 17.14) $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$. Recall also that by Valiant's Theorem (Theorem 17.11) perm is $\#\mathbf{P}$-complete. Thus, under our assumptions

$$\mathbf{NEXP} \subseteq \mathbf{P}^{\mathsf{perm}}. \tag{17}$$

Since we assume that $\mathsf{ZEROP} \in \mathbf{P}$, Lemma 20.19 together with statements (16) and (17) implies that $\mathbf{NEXP} \subseteq \mathbf{NP}$, contradicting the Nondeterministic Time Hierarchy Theorem (Theorem 3.2). Thus the three statements (14), (15) and (14) cannot be simultaneously true. ■

What have we learned?

- Under the assumption of certain circuit lower bounds, there exist pseudorandom generator that can derandomize every probabilistic algorithm.
- In particular, if we make the reasonable assumption that there exists a function in **E** with exponentially large average-case circuit complexity, then $\mathbf{BPP} = \mathbf{P}$.
- Proving that $\mathbf{BPP} = \mathbf{P}$ will require to prove at least some type of circuit lower bounds.

Chapter notes and history

As mentioned in the notes to Chapter 9, pseudorandom generators were first studied in the context of cryptography, by Shamir [Sha81], Blum-Micali [BM82], and Yao [Yao82a]. In particular Yao was the first to point out their potential uses for derandomizing **BPP**. He showed that if secure pseudorandom generators exist then **BPP** can be partially derandomized, specifically, $\mathbf{BPP} \subseteq \cap_{\epsilon>0}\mathbf{DTIME}(2^{n^\epsilon})$. In their seminal paper [NW88], Nisan and Wigderson showed that such derandomization is possible under significantly weaker complexity assumptions, and that under some plausible assumptions it may even be possible to achieve full derandomization of **BPP**,

namely, to show $\mathbf{BPP} = \mathbf{P}$. Since then a large body of work by was devoted to improving the derandomization and weakening the assumptions (see also the notes to Chapter 19). In particular it was shown that *worst-case* hardness assumptions suffice for derandomization (see Chapter 19 and its notes). A central goal of this line of work was achieved by Impagliazzo and Wigderson [IW97], who showed that if $\mathbf{E}$ has a function with exponential circuit complexity then $\mathbf{BPP} = \mathbf{P}$.

A pseudorandom generator with optimal dependence on the hardness assumptions (Theorem 20.6) was given by Umans [Uma03] (see Remark 20.8). Interestingly, this pseudorandom generator is based directly on *worst-case* (as opposed to *average-case*) hardness (and indeed uses the local-decoding techniques originating from the works on hardness amplification). Umans' construction, which uses the Reed-Muller code described in Chapter 19, is based on a previous paper of Shaltiel and Umans [SU01] that constructed a *hitting set generator* (a relaxation of a pseudorandom generator) with the same parameters. Andreev, Clementi, and Rolim [ACR96] showed that such hitting set generators suffice for the application of derandomizing **BPP** (see [GVW00] for a simpler proof).

Impagliazzo and Wigderson [IW98] gave the first derandomization result based on the *uniform* hardness of a function in **EXP** (i.e., Theorem 20.16), a result that gave hope that perhaps the proof of $\mathbf{BPP} = \mathbf{P}$ (or at least $\mathbf{BPP} \neq \mathbf{EXP}$) will not have to wait for progress on circuit lower bounds. Alas, Impagliazzo, Kabanets and Wigderson [IKW01] showed that derandomizing **MA** (or equivalently, the promise-problem version of **BPP**) would imply lower bounds for **NEXP**, while Kabanets and Impagliazzo [KI03] proved Theorem 20.17. That is, they showed that some circuit lower bounds would follow even from derandomizing **BPP**.

Exercises

20.1 Verify Corollary 20.4.

20.2 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 20.2, save for the computational efficiency condition. H458

20.3 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 $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n/10}$.

20.4 Prove that if there exists a an $S(\ell)$-pseudorandom generator then there exists a function $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$. H458

20.5 Prove Theorem 20.10.

20.6 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 $\mathbf{MA} = \mathbf{NP}$. H458

20.7 We define an *oracle Boolean circuit* to be a Boolean circuit that have special gates with unbounded fan-in 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 $\mathsf{H}_{\mathsf{avg}}{}^O(f)$ to be the function that maps every $n \in \mathbb{N}$ to the largest S such that $\Pr_{x \in_R \{0,1\}^n}[C^O(x) = f(x)] \leq 1/2 + 1/S$. Prove that if there exists $f \in \mathbf{E}$ and $\epsilon > 0$ with $\mathsf{H}_{\mathsf{avg}}{}^{\mathsf{3SAT}}(f) \geq 2^{\epsilon n}$ then $\mathbf{AM} = \mathbf{NP}$.

20.8 Prove that if $\mathbf{EXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ then the conclusions of Theorem 20.16 hold. H458

20.9 Let $G : \{0,1\}^* \to \{0,1\}^*$ be an $S(\ell)$-length candidate pseudorandom generator that *fails* to derandomize a particular **BPP** algorithm A on the average case. That is, letting $L \in \mathbf{BPP}$ be the language such that $\Pr[A(x) = L(x)] \geq 2/3$, it holds that for every sufficiently large n, with probability at least $1/n$ over the choice of $x \in_R \{0,1\}^n$, $\Pr[A(x; G(U_{\ell(n)})) = L(x)] \leq 1/2$ (we let $\ell(n)$ be such that $S(\ell(n)) = m(n)$ where $m(n)$ denotes the length of random tape used by A on inputs of length n). Prove that there exists a probabilistic polynomial-time algorithm D that on input 1^n outputs a circuit D_n such that with probability at least $1/(2n)$ over the randomness of D,

$$\| \mathsf{E}[D_n(G(U_{\ell(n)})] - \mathsf{E}[D_n(U_{m(n)})] \| > 0.1 \,.$$

H458

20.10 (van Melkebeek 2000, see [IKW01]) Prove that if $\mathbf{NEXP} = \mathbf{MA}$ then $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$.