Chapter 7

The Bright Side of Hardness

So saying she donned her beautiful, glittering golden–Ambrosial sandals, which carry her flying like the wind over the vast land and sea; she grasped the redoubtable bronze-shod spear, so stout and sturdy and strong, wherewith she quells the ranks of heroes who have displeased her, the [bright-eyed] daughter of her mighty father.

Homer, Odyssey, 1:96–101

The existence of natural computational problems that are (or seem to be) infeasible to solve is usually perceived as bad news, because it means that we cannot do things we wish to do. But these bad news have a positive side, because hard problem can be "put to work" to our benefit, most notably in cryptography.

It seems that utilizing hard problems requires the ability to efficiently generate hard instances, which is not guaranteed by the notion of worst-case hardness. In other words, we refer to the gap between "occasional" hardness (e.g., worst-case hardness or mild average-case hardness) and "typical" hardness (with respect to some tractable distribution). Much of the current chapter is devoted to bridging this gap, which is known by the term *hardness amplification*. The actual applications of typical hardness are presented in Chapter 8 and Appendix C.

Summary: We consider two conjectures that are related to $\mathcal{P} \neq \mathcal{NP}$. The first conjecture is that there are problems that are solvable in exponential-time (i.e., in $\mathcal{E}$) but are not solvable by (non-uniform) families of small (say polynomial-size) circuits. We show that this worst-case conjecture can be transformed into an average-case hardness result; specifically, we obtain predicates that are strongly "inapproximable" by small circuits. Such predicates are used towards derandomizing $\mathcal{BPP}$ in a non-trivial manner (see Section 8.3).

The second conjecture is that there are problems in NP (i.e., search problems in $\mathcal{PC}$) for which it is easy to generate (solved) instances that

are typically hard to solve (for a party that did not generate these instances). This conjecture is captured in the formulation of *one-way functions*, which are functions that are easy to evaluate but hard to invert (in an average-case sense). We show that functions that are hard to invert in a relatively mild average-case sense yield functions that are hard to invert in a strong average-case sense, and that the latter yield predicates that are very hard to approximate (called *hard-core predicates*). Such predicates are useful for the construction of general-purpose pseudorandom generators (see Section 8.2) as well as for a host of cryptographic applications (see Appendix C).

In the rest of this chapter, the actual order of presentation of the two aforementioned conjectures and their consequences is reversed: We start (in Section 7.1) with the study of one-way functions, and only later (in Section 7.2) turn to the study of problems in $\mathcal{E}$ that are hard for small circuits.

Teaching note: We list several reasons for preferring the aforementioned order of presentation. First, we mention the great conceptual appeal of one-way functions and the fact that they have very practical applications. Second, hardness amplification in the context of one-way functions is technically simpler than the amplification of hardness in the context of $\mathcal{E}$. (In fact, Section 7.2 is the most technical text in this book.) Third, some of the techniques that are shared by both treatments seem easier to understand first in the context of one-way functions. Last, the current order facilitates the possibility of teaching hardness amplification only in one incarnation, where the context of one-way functions is recommended as the incarnation of choice (for the aforementioned reasons).

If you wish to teach hardness amplification and pseudorandomness in the two aforementioned incarnations, then we suggest following the order of the current text. That is, first teach hardness amplification in its two incarnations, and only next teach pseudorandomness in the corresponding incarnations.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1). In particular, standard conventions regarding random variables (presented in Appendix D.1.1) and various "laws of large numbers" (presented in Appendix D.1.2) will be extensively used.

7.1 One-Way Functions

Loosely speaking, one-way functions are functions that are easy to evaluate but hard (on the average) to invert. Thus, in assuming that one-way functions exist, we are postulating the existence of *efficient processes* (i.e., the computation of the function in the forward direction) *that are hard to reverse*. Analogous phenomena in daily life are known to us in abundance (e.g., the lighting of a match). Thus, the assumption that one-way functions exist is a complexity theoretic analogue of our daily experience.

One-way functions can also be thought of as efficient ways for generating "puzzles" that are infeasible to solve; that is, the puzzle is a random image of the function and a solution is a corresponding preimage. Furthermore, the person generating the puzzle knows a solution to it and can efficiently verify the validity of (possibly other) solutions to the puzzle. In fact, as explained in Section 7.1.1, every mechanism for generating such puzzles can be converted to a one-way function.

The reader may note that when presented in terms of generating hard puzzles, one-way functions have a clear cryptographic flavor. Indeed, one-way functions are central to cryptography, but we shall not explore this aspect here (and rather refer the reader to Appendix C). Similarly, one-way functions are closely related to (general-purpose) pseudorandom generators, but this connection will be explored in Section 8.2. Instead, in the current section, we will focus on one-way functions *per se*.

Teaching note: While we recommend including a basic treatment of pseudorandomness within a course on complexity theory, we do not recommend doing so with respect to cryptography. The reason is that cryptography is far more complex than pseudorandomness (e.g., compare the definition of secure encryption to the the definition of pseudorandom generators). The extra complexity is due to conceptual richness, which is something good, except that some of these conceptual issues are central to cryptography but not to complexity theory. Thus, teaching cryptography in the context of a course on complexity theory is likely to either overload the course with material that is not central to complexity theory or cause a superficial and misleading treatment of cryptography. We are not sure as to which of these two possibilities is worse. Still, for the benefit of the interested reader, we have included an overview of the foundations of cryptography as an appendix to the main text (see Appendix C).

7.1.1 Generating hard instances and one-way functions

Let us start by examining the prophecy, made in the preface to this chapter, by which intractable problems can be used to our benefit. The basic idea is that intractable problems offer a way of generating an obstacle that stands in the way of our opponents and thus protects our interests. These opponents may be either real (e.g., in the context of cryptography) or imaginary (e.g., in the context of derandomization), but in both cases we wish to prevent them from seeing something or doing something. Hard obstacles seems useful towards this goal.

Let us assume that $\mathcal{P} \neq \mathcal{NP}$ or even that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$. Can we use this assumption to our benefit? Not really: The $\mathcal{NP} \not\subseteq \mathcal{BPP}$ assumption refers to the worst-case complexity of problems, while benefiting from hard problems seems to require the ability to generate hard instances. In particular, the generated instances should be typically hard and not merely occasionally hard; that is, we seek average-case hardness and not merely worst-case hardness.

Taking a short digression, we mention that in Section 7.2 we shall see that worst-case hardness (of $\mathcal{NP}$ or even $\mathcal{E}$) can be transformed into average-case hardness of $\mathcal{E}$. Such a transformation is not known for $\mathcal{NP}$ itself, and in some applications (e.g., in cryptography) we do need the hard-on-the-average problem to be in $\mathcal{NP}$.

In this case, we currently need to assume that, for some problem in $\mathcal{NP}$, it is the case that hard instances are easy to generate (and not merely exist). That is, we assume that $\mathcal{NP}$ is "hard on the average" *with respect to a distribution that is efficiently sampleable.* This assumption will be further discussed in Section 10.2.

However, for the aforementioned applications (e.g., in cryptography) this assumption does not seem to suffice either: we know how to utilize such "hard on the average" problems *only when we can efficiently generate hard instances coupled with adequate solutions.*[1] That is, we assume that, for some search problem in $\mathcal{PC}$ (resp., decision problem in $\mathcal{NP}$), we can efficiently generate instance-solution pairs (resp., yes-instances coupled with corresponding NP-witnesses) such that the instance is hard to solve (resp., hard to verify as belonging to the set). Needless to say, the hardness assumption refers to a person that does not get the solution (resp., witness). Thus, we can efficiently generate hard "puzzles" coupled with solutions, and so we may present to others hard puzzles for which we know a solution.

Let us formulate the foregoing discussion. Referring to Definition 2.3, we consider a relation R in $\mathcal{PC}$ (i.e., R is polynomially bounded and membership in R can be determined in polynomial-time), and assume that there exists a probabilistic polynomial-time algorithm G that satisfies the following two conditions:

1. On input 1^n, algorithm G always generates a pair in R such that the first element has length n. That is, $\mathsf{Pr}[G(1^n) \in R \cap (\{0,1\}^n \times \{0,1\}^*)] = 1$.

2. It is typically infeasible to find solutions to instances that are generated by G; that is, when only given the first element of $G(1^n)$, it is infeasible to find an adequate solution. Formally, denoting the first element of $G(1^n)$ by $G_1(1^n)$, for every probabilistic polynomial-time (solver) algorithm S, it holds that $\mathsf{Pr}[(G_1(1^n), S(G_1(1^n)) \in R] = \mu(n)$, where μ vanishes faster than any polynomial fraction (i.e., for every positive polynomial p and all sufficiently large n it is the case that $\mu(n) < 1/p(n)$).

We call G a generator of solved intractable instances for R. We will show that such a generator exists if and only if one-way functions exist, where one-way functions are functions that are easy to evaluate but hard (on the average) to invert. That is, a function $f:\{0,1\}^* \to \{0,1\}^*$ is called one-way if there is an efficient algorithm that on input x outputs $f(x)$, whereas any feasible algorithm that tries to find a preimage of $f(x)$ under f may succeed only with negligible probability (where the probability is taken uniformly over the choices of x and the algorithm's coin tosses). Associating feasible computations with probabilistic polynomial-time algorithms and negligible functions with functions that vanish faster than any polynomial fraction, we obtain the following definition.

Definition 7.1 (one-way functions): *A function* $f:\{0,1\}^* \to \{0,1\}^*$ *is called* one-way *if the following two conditions hold:*

[1]We wish to stress the difference between the two gaps discussed here. Our feeling is that the non-usefulness of worst-case hardness (*per se*) is far more intuitive than the non-usefulness of average-case hardness that does not correspond to an efficient generation of "solved" instances.

1. Easy to evaluate: *There exist a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0,1\}^*$.*

2. Hard to invert: *For every probabilistic polynomial-time algorithm A', every polynomial p, and all sufficiently large n,*

$$\mathsf{Pr}_{x\in\{0,1\}^n}[A'(f(x),1^n) \in f^{-1}(f(x))] \;<\; \frac{1}{p(n)} \tag{7.1}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses of algorithm A'.

Algorithm A' is given the auxiliary input 1^n so as to allow it to run in time polynomial in the length of x, which is important in case f drastically shrinks its input (e.g., $|f(x)| = O(\log|x|)$). Typically (and, in fact, without loss of generality, see Exercise 7.1), f is length preserving, in which case the auxiliary input 1^n is redundant. Note that A' is not required to output a specific preimage of $f(x)$; any preimage (i.e., element in the set $f^{-1}(f(x))$) will do. (Indeed, in case f is 1-1, the string x is the only preimage of $f(x)$ under f; but in general there may be other preimages.) It is required that algorithm A' fails (to find a preimage) with overwhelming probability, when the probability is also taken over the input distribution. That is, f is "typically" hard to invert, not merely hard to invert in some ("rare") cases.

Proposition 7.2 *The following two conditions are equivalent:*

1. *There exists a generator of solved intractable instances for some $R \in \mathcal{NP}$.*

2. *There exist one-way functions.*

Proof Sketch: Suppose that G is such a generator of solved intractable instances for some $R \in \mathcal{NP}$, and suppose that on input 1^n it tosses $\ell(n)$ coins. For simplicity, we assume that $\ell(n) = n$, and consider the function $g(r) = G_1(1^{|r|}, r)$, where $G(1^n, r)$ denotes the output of G on input 1^n when using coins r (and G_1 is as in the foregoing discussion). Then g must be one-way, because an algorithm that inverts g on input $x = g(r)$ obtains r' such that $G_1(1^n, r') = x$ and $G(1^n, r')$ must be in R (which means that the second element of $G(1^n, r')$ is a solution to x). In case $\ell(n) \neq n$ (and assuming without loss of generality that $\ell(n) \geq n$), we define $g(r) = G_1(1^n, s)$ where n is the largest integer such that $\ell(n) \leq |r|$ and s is the $\ell(n)$-bit long prefix of r.

Suppose, on the other hand, that f is a one-way function (and that f is length preserving). Consider $G(1^n)$ that uniformly selects $r \in \{0,1\}^n$ and outputs $(f(r), r)$, and let $R \stackrel{\rm def}{=} \{(f(x), x) : x \in \{0,1\}^*\}$. Then R is in $\mathcal{PC}$ and G is a generator of solved intractable instances for R, because any solver of R (on instances generated by G) is effectively inverting f on $f(U_n)$. $\square$

Comments. Several candidates one-way functions and variation on the basic definition appear in Appendix C.2.1. Here, for the sake of future discussions, we define a stronger version of one-way functions, which refers to the infeasibility of inverting the function by non-uniform circuits of polynomial-size. We seize the opportunity and use an alternative technical formulation, which is based on the probabilistic conventions in Appendix D.1.1.[2]

Definition 7.3 (one-way functions, non-uniformly hard): *A one-way function* $f:\{0,1\}^* \to \{0,1\}^*$ *is said to be* non-uniformly hard to invert *if for every family of polynomial-size circuits* $\{C_n\}$*, every polynomial* p*, and all sufficiently large* n*,*

$$\mathsf{Pr}[C_n(f(U_n),1^n) \in f^{-1}(f(U_n))] \;<\; \frac{1}{p(n)}$$

We note that if a function is infeasible to invert by polynomial-size circuits then it is hard to invert by probabilistic polynomial-time algorithms; that is, non-uniformity (more than) compensates for lack of randomness. See Exercise 7.2.

7.1.2 Amplification of Weak One-Way Functions

In the forgoing discussion we have interpreted "hardness on the average" in a very strong sense. Specifically, we required that any feasible algorithm fails to solve the problem (e.g., invert the one-way function) *almost always* (i.e., *except with negligible probability*). This interpretation is indeed the one that is suitable for various applications. Still, a weaker interpretation of hardness on the average, which is also appealing, only requires that any feasible algorithm fails to solve the problem *often enough* (i.e., *with noticeable probability*). The main thrust of the current section is showing that the mild form of hardness on the average can be transformed into the strong form discussed in Section 7.1.1. Let us first define the mild form of hardness on the average, using the framework of one-way functions. Specifically, we define weak one-way functions.

Definition 7.4 (weak one-way functions): *A function* $f:\{0,1\}^* \to \{0,1\}^*$ *is called* weakly one-way *if the following two conditions hold:*

1. Easy to evaluate: *As in Definition 7.1.*
2. Weakly hard to invert: *There exists a positive polynomial* p *such that for every probabilistic polynomial-time algorithm* A' *and all sufficiently large* n*,*

$$\mathsf{Pr}_{x\in\{0,1\}^n}[A'(f(x),1^n) \notin f^{-1}(f(x))] \;>\; \frac{1}{p(n)} \tag{7.2}$$

   where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ *and all the possible outcomes of the internal coin tosses of algorithm* A'*. In such a case, we say that* f *is* $1/p$-one-way.

[2] Specifically, letting U_n denote a random variable uniformly distributed in $\{0,1\}^n$, we may write Eq. (7.1) as $\mathsf{Pr}[A'(f(U_n),1^n) \in f^{-1}(f(U_n))] \;<\; 1/p(n)$, recalling that both occurrences of U_n refer to the same sample.

Here we require that algorithm A' fails (to find an f-preimage for a random f-image) with noticeable probability, rather than with overwhelmingly high probability (as in Definition 7.1). For clarity, we will occasionally refer to one-way functions as in Definition 7.1 by the term strong one-way functions.

We note that, assuming that one-way functions exist at all, there exists weak one-way functions that are not strongly one-way (see Exercise 7.3). Still, any weak one-way function can be transformed into a strong one-way function. This is indeed the main result of the current section.

Theorem 7.5 (amplification of one-way functions): *The existence of weak one-way functions implies the existence of strong one-way functions.*

Proof Sketch: The construction itself is straightforward. We just parse the argument to the new function into sufficiently many blocks, and apply the weak one-way function on the individual blocks. That is, suppose that f is $1/p$-one-way, for some polynomial p, and consider the following function

$$\begin{aligned} F(x_1, ..., x_t) &= (f(x_1), ..., f(x_t)) \\ &\quad \text{where } t \stackrel{\text{def}}{=} n \cdot p(n) \text{ and } x_1, ..., x_t \in \{0,1\}^n. \end{aligned} \tag{7.3}$$

(Indeed F should be extended to strings of length outside $\{n^2 \cdot p(n) : n \in \mathbb{N}\}$ and this extension must be hard to invert on all preimage lengths.)[3]

We warn that the hardness of inverting the resulting function F is not established by mere "combinatorics" (i.e., considering, for any $S \subset \{0,1\}^n$, the relative volume of S^t in $(\{0,1\}^n)^t$, where S represents the set of f-preimages that are mapped by f to an image that is "easy to invert"). Specifically, one may *not* assume that the potential inverting algorithm works independently on each block. Indeed this assumption seems reasonable, but we do not know if nothing is lost by this restriction. (In fact, proving that nothing is lost by this restriction is a formidable research project.) In general, we should not make assumptions regarding the class of all efficient algorithms (as underlying the definition of one-way functions), unless we can actually prove that nothing is lost by such assumptions.

The hardness of inverting the resulting function F is proved via a so called "reducibility argument" (which is used to prove all conditional results in the area). By a reducibility argument we actually mean a reduction, but one that is analyzed with respect to average case complexity. Specifically, we show that any algorithm that inverts the resulting function F with non-negligible success probability can be used to construct an algorithm that inverts the original function f with success probability that violates the hypothesis (regarding f). In other words, we reduce the task of "strongly inverting" f (i.e., violating its weak one-wayness) to the task of "weakly inverting" F (i.e., violating its strong one-wayness). In particular, on input $y = f(x)$, the reduction invokes the F-inverter (polynomially) many times, each time feeding it with a sequence of random f-images that contains y at a

[3]One simple extension is defining $F(x)$ to equal $F(x_1, ..., x_{n \cdot p(n)})$, where n is the largest integer satisfying $n^2 p(n) \le |x|$ and x_i is the i^{th} consecutive n-bit long string in x (i.e., $x = x_1 \cdots x_{n \cdot p(n)} x'$, where $x_1, ..., x_{n \cdot p(n)} \in \{0,1\}^n$).

random location. (Indeed such a sequence corresponds to a random image of F.) Details follow.

Suppose towards the contradiction that F is not strongly one-way; that is, there exists a probabilistic polynomial-time algorithm B' and a polynomial $q(\cdot)$ so that for infinitely many m's

$$\mathsf{Pr}[B'(F(U_m)) \in F^{-1}(F(U_m))] > \frac{1}{q(m)} \tag{7.4}$$

Focusing on such a generic m and assuming (see Footnote 3) that $m = n^2p(n)$, we present the following probabilistic polynomial-time algorithm, A', for inverting f. On input y and 1^n (where supposedly $y = f(x)$ for some $x \in \{0,1\}^n$), algorithm A' proceeds by applying the following probabilistic procedure, denoted I, on input y for $t'(n)$ times, where $t'(\cdot)$ is a polynomial that depends on the polynomials p and q (specifically, we set $t'(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2p(n))$).

Procedure I (on input y and 1^n):

`For` $i = 1$ `to` $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$ `do begin`

(1) Select uniformly and independently a sequence of strings $x_1, ..., x_{t(n)} \in \{0,1\}^n$.

(2) Compute $(z_1, ..., z_{t(n)}) \leftarrow B'(f(x_1), ..., f(x_{i-1}), y, f(x_{i+1}), ..., f(x_{t(n)}))$

(Note that y is placed in the i^{th} position instead of $f(x_i)$.)

(3) If $f(z_i) = y$ then halt and output z_i.

(This is considered a *success*).

`end`

Using Eq. (7.4), we now present a lower bound on the success probability of algorithm A', deriving a contradiction to the theorem's hypothesis. To this end we define a set, denoted S_n, that contains all n-bit strings on which the procedure I succeeds with probability greater than $n/t'(n)$. (The probability is taken only over the coin tosses of procedure I). Namely,

$$S_n \stackrel{\text{def}}{=} \left\{ x \in \{0,1\}^n : \mathsf{Pr}[I(f(x)) \in f^{-1}(f(x))] > \frac{n}{t'(n)} \right\}$$

In the next two claims we shall show that S_n contains all but at most a $1/2p(n)$ fraction of the strings of length n, and that for each string $x \in S_n$ algorithm A' inverts f on $f(x)$ with probability exponentially close to 1. It will follow that A' inverts f on $f(U_n)$ with probability greater than $1 - (1/p(n))$, in contradiction to the theorem's hypothesis.

Claim 7.5.1: For every $x \in S_n$

$$\mathsf{Pr}\left[A'(f(x)) \in f^{-1}(f(x))\right] > 1 - 2^{-n}$$

This claim follows directly from the definitions of S_n and A'.

Claim 7.5.2:

$$|S_n| > \left(1 - \frac{1}{2p(n)}\right) \cdot 2^n$$

The rest of the proof is devoted to establishing this claim, and indeed combining Claims 7.5.1 and 7.5.2, the theorem follows.

The key observation is that, for every $i \in [t(n)]$ and every $x_i \in \{0,1\}^n \setminus S_n$, it holds that

$$\begin{aligned}\mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)})) \,\middle|\, U_n^{(i)} = x_i\right] \\ \leq \mathsf{Pr}\left[I(f(x_i)) \in f^{-1}(f(x_i))\right] \leq \frac{n}{t'(n)}\end{aligned}$$

where $U_n^{(1)}, ..., U_n^{(n\cdot p(n))}$ denote the n-bit long blocks in the random variable $U_{n^2p(n)}$. It follows that

$$\begin{aligned}\xi &\stackrel{\text{def}}{=} \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)})) \wedge \left(\exists i \text{ s.t. } U_n^{(i)} \in \{0,1\}^n \setminus S_n\right)\right] \\ &\leq \sum_{i=1}^{t(n)} \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)})) \wedge U_n^{(i)} \in \{0,1\}^n \setminus S_n\right] \\ &\leq t(n) \cdot \frac{n}{t'(n)} = \frac{1}{2q(n^2p(n))}\end{aligned}$$

where the equality is due to $t'(n) = 2n^2 \cdot p(n) \cdot q(n^2p(n))$ and $t(n) = n \cdot p(n)$. On the other hand, using Eq. (7.4), we have

$$\begin{aligned}\xi &\geq \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)}))\right] - \mathsf{Pr}\left[(\forall i)\, U_n^{(i)} \in S_n\right] \\ &\geq \frac{1}{q(n^2p(n))} - \mathsf{Pr}\left[U_n \in S_n\right]^{t(n)} .\end{aligned}$$

Using $t(n) = n \cdot p(n)$, we get $\mathsf{Pr}[U_n \in S_n] > (1/2q(n^2p(n)))^{1/(n\cdot p(n))}$, which implies $\mathsf{Pr}[U_n \in S_n] > 1 - (1/2p(n))$ for sufficiently large n. Claim 7.5.2 follows, and so does the theorem. □

Digest. Let us recall the structure of the proof of Theorem 7.5. Given a weak one-way function f, we first constructed a polynomial-time computable function F with the intention of later proving that F is strongly one-way. To prove that F is strongly one-way, we used a *reducibility argument*. The argument transforms efficient algorithms that supposedly contradict the strong one-wayness of F into efficient algorithms that contradict the hypothesis that f is weakly one-way. Hence F must be strongly one-way. We stress that our algorithmic transformation, which is in fact a randomized Cook reduction, makes no implicit or explicit assumptions about the structure of the prospective algorithms for inverting F. Such assumptions (e.g., the "natural" assumption that the inverter of F works independently on each block) cannot be justified (at least not at our current state of understanding of the nature of efficient computations).

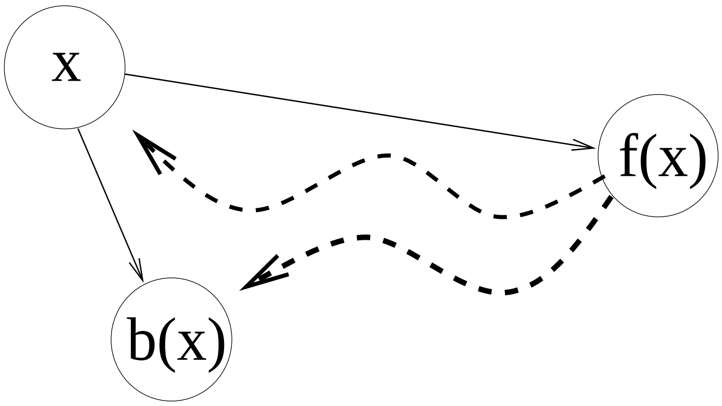
We use the term a *reducibility argument*, rather than just saying a reduction so as to emphasize that we do *not* refer here to standard (worst-case complexity) reductions. Let us clarify the distinction: In both cases we refer to *reducing* the task of solving one problem to the task of solving another problem; that is, we use a procedure solving the second task in order to construct a procedure that solves the first task. However, in standard reductions one assumes that the second task has a perfect procedure solving it on all instances (i.e., on the worst-case), and constructs such a procedure for the first task. Thus, the reduction may invoke the given procedure (for the second task) on very "non-typical" instances. This cannot be allowed in our reducibility arguments. Here, we are given a procedure that solves the second task *with certain probability with respect to a certain distribution*. Thus, in employing a reducibility argument, we cannot invoke this procedure on any instance. Instead, we must consider the probability distribution, on instances of the second task, induced by our reduction. In our case (as in many cases) the latter distribution equals the distribution to which the hypothesis (regarding solvability of the second task) refers, but in general these distributions need only be "sufficiently close" in an adequate sense (which depends on the analysis). In any case, a careful consideration of the distribution induced by the reducibility argument is due. (Indeed, the same issue arises in the context of reductions among "distributional problems" considered in Section 10.2.)

An information theoretic analogue. Theorem 7.5 (or rather its proof) has a natural information theoretic (or "probabilistic") analogue that refers to the amplification of the success probability by repeated experiments: If some event occurs with probability p in a single experiment, then the event will occur with very high probability (i.e., $1-e^{-n}$) when the experiment is repeated n/p times. The analogy is to evaluating the function F at a random input, where each block of this input may be viewed as an attempt to hit the noticeable "hard region" of f. The reader is probably convinced at this stage that the proof of Theorem 7.5 is much more complex than the proof of the information theoretic analogue. In the information theoretic context the repeated experiments are independent by definition, whereas in the computational context no such independence can be guaranteed. (Indeed, the independence assumption corresponds to the naive argument discussed at the beginning of the proof of Theorem 7.5.) Another indication to the difference between the two settings follows. In the information theoretic setting, the probability that the event did not occur in any of the repeated trials decreases exponentially with the number of repetitions. In contrast, in the computational setting we can only reach an unspecified negligible bound on the inverting probabilities of polynomial-time algorithms. Furthermore, for all we know, it may be the case that F can be efficiently inverted on $F(U_{n^2p(n)})$ with success probability that is sub-exponentially decreasing (e.g., with probability $2^{-(\log_2 n)^3}$), whereas the analogous information theoretic bound is exponentially decreasing (i.e., e^{-n}).

7.1.3 Hard-Core Predicates

One-way functions *per se* suffice for one central application: the construction of secure signature schemes (see Appendix C.6). For other applications, one relies not merely on the infeasibility of fully recovering the preimage of a one-way function, but rather on the infeasibility of meaningfully guessing bits in the preimage. The latter notion is captured by the definition of a hard-core predicate.

Recall that saying that a function f is one-way means that given a typical y (in the range of f) it is infeasible to find a preimage of y under f. This does not mean that it is infeasible to find partial information about the preimage(s) of y under f. Specifically, it may be easy to retrieve half of the bits of the preimage (e.g., given a one-way function f consider the function f' defined by $f'(x,r) \stackrel{\rm def}{=} (f(x),r)$, for every $|x|=|r|$). We note that hiding partial information (about the function's preimage) plays an important role in more advanced constructs (e.g., pseudorandom generators and secure encryption). With this motivation in mind, we will show that essentially any one-way function hides specific partial information about its preimage, where this partial information is easy to compute from the preimage itself. This partial information can be considered as a "hard core" of the difficulty of inverting f. Loosely speaking, a *polynomial-time computable* (Boolean) predicate b, is called a hard-core of a function f if no feasible algorithm, given $f(x)$, can guess $b(x)$ with success probability that is non-negligibly better than one half.



The solid arrows depict easily computable transformation while the dashed arrows depict infeasible transformations.

Figure 7.1: The hard-core of a one-way function – an illustration.

Definition 7.6 (hard-core predicates): *A polynomial-time computable predicate* $b : \{0,1\}^* \to \{0,1\}$ *is called a* hard-core *of a function* f *if for every probabilistic polynomial-time algorithm* A'*, every positive polynomial* $p(\cdot)$*, and all sufficiently large* n*'s*

$$\Pr\left[A'(f(x))=b(x)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ *and all the possible outcomes of the internal coin tosses of algorithm* A'*.*

Note that for every $b : \{0,1\}^* \to \{0,1\}$ and $f : \{0,1\}^* \to \{0,1\}^*$, there exist obvious algorithms that guess $b(x)$ from $f(x)$ with success probability at least one half (e.g., the algorithm that, obliviously of its input, outputs a uniformly chosen bit). Also, if b is a hard-core predicate (of any function) then it follows that b is almost unbiased (i.e., for a uniformly chosen x, the difference $|\mathsf{Pr}[b(x)=0] - \mathsf{Pr}[b(x)=1]|$ must be a negligible function in n).

Since b itself is polynomial-time computable, the failure of efficient algorithms to approximate $b(x)$ from $f(x)$ (with success probability that is non-negligibly higher than one half) must be due either to an information loss of f (i.e., f not being one-to-one) or to the difficulty of inverting f. For example, for $\sigma \in \{0,1\}$ and $x' \in \{0,1\}^*$, the predicate $b(\sigma x') = \sigma$ is a hard-core of the function $f(\sigma x') \stackrel{\text{def}}{=} 0x'$. Hence, in this case the fact that b is a hard-core of the function f is due to the fact that f loses information (specifically, the first bit: σ). On the other hand, in the case that f loses no information (i.e., f is one-to-one) a hard-core for f may exist only if f is hard to invert. In general, the interesting case is when being a hard-core is a computational phenomenon rather than an information theoretic one (which is due to "information loss" of f). It turns out that any one-way function has a modified version that possesses a hard-core predicate.

Theorem 7.7 (a generic hard-core predicate): *For any one-way function f, the inner-product mod 2 of x and r, denoted $b(x,r)$, is a hard-core of $f'(x,r) = (f(x),r)$.*

In other words, Theorem 7.7 asserts that, given $f(x)$ and a random subset $S \subseteq [|x|]$, it is infeasible to guess $\oplus_{i\in S} x_i$ significantly better than with probability $1/2$, where $x = x_1 \cdots x_n$ is uniformly distributed in $\{0,1\}^n$.

Proof Sketch: The proof is by a so-called "reducibility argument" (see Section 7.1.2). Specifically, we reduce the task of inverting f to the task of predicting the hard-core of f', while making sure that the reduction (when applied to input distributed as in the inverting task) generates a distribution as in the definition of the predicting task. Thus, a contradiction to the claim that b is a hard-core of f' yields a contradiction to the hypothesis that f is hard to invert. We stress that this argument is far more complex than analyzing the corresponding "probabilistic" situation (i.e., the distribution of $(r, b(X,r))$, where $r \in \{0,1\}^n$ is uniformly distributed and X is a random variable with super-logarithmic min-entropy (which represents the "effective" knowledge of x, when given $f(x)$)).[4]

Our starting point is a probabilistic polynomial-time algorithm B that satisfies, for some polynomial p and infinitely many n's, $\mathsf{Pr}[B(f(X_n),U_n) = b(X_n,U_n)] > (1/2) + (1/p(n))$, where X_n and U_n are uniformly and independently distributed over $\{0,1\}^n$. Using a simple averaging argument, we focus on a $\varepsilon \stackrel{\text{def}}{=} 1/2p(n)$

[4]The min-entropy of X is defined as $\min_v\{\log_2(1/\mathsf{Pr}[X=v])\}$; that is, if X has min-entropy m then $\max_v\{\mathsf{Pr}[X=v]\} = 2^{-m}$. The Leftover Hashing Lemma (see Appendix D.2) implies that, in this case, $\mathsf{Pr}[b(X,U_n)=1|U_n] = \frac{1}{2} \pm 2^{-\Omega(m)}$, where U_n denotes the uniform distribution over $\{0,1\}^n$.

fraction of the x's for which $\Pr[B(f(x),U_n) = b(x,U_n)] > (1/2)+\varepsilon$ holds. We will show how to use B in order to invert f, on input $f(x)$, provided that x is in this good set (which has density ε).

As a warm-up, suppose for a moment that, for the aforementioned x's, algorithm B succeeds with probability p such that $p > \frac{3}{4} + 1/\text{poly}(|x|)$ rather than $p > \frac{1}{2} + 1/\text{poly}(|x|)$. In this case, retrieving x from $f(x)$ is quite easy: To retrieve the i^{th} bit of x, denoted x_i, we randomly select $r \in \{0,1\}^{|x|}$, and obtain $B(f(x),r)$ and $B(f(x),r\oplus e^i)$, where $e^i = 0^{i-1}10^{|x|-i}$ and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. A key observation underlying the foregoing scheme as well as the rest of the proof is that $b(x,r\oplus s) = b(x,r) \oplus b(x,s)$, which can be readily verified by writing $b(x,y) = \sum_{i=1}^{n} x_i y_i \bmod 2$ and noting that addition modulo 2 of bits corresponds to their XOR. Now, note that if both $B(f(x),r) = b(x,r)$ and $B(f(x),r\oplus e^i) = b(x,r\oplus e^i)$ hold, then $B(f(x),r) \oplus B(f(x),r\oplus e^i)$ equals $b(x,r) \oplus b(x,r\oplus e^i) = b(x,e^i) = x_i$. The probability that both $B(f(x),r) = b(x,r)$ and $B(f(x),r\oplus e^i) = b(x,r\oplus e^i)$ hold, for a random r, is at least $1 - 2\cdot(1-p) > \frac{1}{2} + \frac{1}{\text{poly}(|x|)}$. Hence, repeating the foregoing procedure sufficiently many times (using independent random choices of such r's) and ruling by majority, we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $p > \frac{3}{4}+\frac{1}{\text{poly}(|x|)}$, whereas we only know that $p > \frac{1}{2}+\varepsilon$ for $\varepsilon = 1/\text{poly}(|x|)$.

The problem with the foregoing procedure is that it doubles the original error probability of algorithm B on inputs of the form $(f(x),\cdot)$. Under the unrealistic (foregoing) assumption that B's average error on such inputs is non-negligibly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where B's error is exactly $\frac{1}{4}$) the foregoing procedure is unlikely to invert f. Note that the *average* error probability of B (for a fixed $f(x)$, when the average is taken over a random r) can not be decreased by repeating B several times (e.g., for every x, it may be that B always answer correctly on three quarters of the pairs $(f(x),r)$, and always err on the remaining quarter). What is required is an *alternative way of using* the algorithm B, a way that does not double the original error probability of B.

The key idea is generating the r's in a way that allows applying algorithm B only once per each r (and i), instead of twice. Specifically, we will invoke B on $(f(x),r\oplus e^i)$ in order to obtain a "guess" for $b(x,r\oplus e^i)$, and obtain $b(x,r)$ in a different way (which does not involve using B). The good news is that the error probability is no longer doubled, since we only use B to get a "guess" of $b(x,r\oplus e^i)$. The bad news is that we still need to know $b(x,r)$, and it is not clear how we can know $b(x,r)$ without applying B. The answer is that we can guess $b(x,r)$ by ourselves. This is fine if we only need to guess $b(x,r)$ for one r (or logarithmically in $|x|$ many r's), but the problem is that we need to know (and hence guess) the value of $b(x,r)$ for polynomially many r's. The obvious way of guessing these $b(x,r)$'s yields an exponentially small success probability. Instead, we generate these polynomially many r's such that, on one hand they are "sufficiently random" whereas, on the other hand, we can guess all the $b(x,r)$'s with noticeable success probability.[5] Specifically, generating the r's in a specific *pairwise independent* manner will satisfy both these (conflicting) requirements. We stress that in case we are successful (in our guesses for all the $b(x,r)$'s), we can retrieve x with high probability. Hence, we retrieve x with noticeable probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x,r)$'s are guessed) is indeed in place. To generate $m = \text{poly}(|x|)$ many r's, we uniformly (and independently) select $\ell \stackrel{\text{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^{|x|}$. Let us denote these strings by $s^1,...,s^\ell$. We then guess $b(x,s^1)$ through $b(x,s^\ell)$. Let us denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^ℓ. Hence, the probability that all our guesses for the $b(x,s^i)$'s are correct is $2^{-\ell} = \frac{1}{\text{poly}(|x|)}$. The different r's correspond to the different *non-empty* subsets of $\{1,2,...,\ell\}$. Specifically, for every such subset J, we let $r^J \stackrel{\text{def}}{=} \oplus_{j\in J} s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^{|x|}$; see Exercise 7.5. The key observation is that $b(x,r^J) = b(x,\oplus_{j\in J} s^j) = \oplus_{j\in J} b(x,s^j)$. Hence, our guess for $b(x,r^J)$ is $\oplus_{j\in J}\sigma^j$, and with noticeable probability all our guesses are correct. Wrapping-up everything, we obtain the following procedure, where $\varepsilon = 1/\text{poly}(n)$ represents a lower-bound on the advantage of B in guessing $b(x,\cdot)$ for an ε fraction of the x's (i.e., for these good x's it holds that $\Pr[B(f(x),U_n) = b(x,U_n)] > \frac{1}{2}+\varepsilon$).

Inverting procedure (on input $y = f(x)$ and parameters n and ε):
Set $\ell = \log_2(n/\varepsilon^2) + O(1)$.
(1) Select uniformly and independently $s^1,...,s^\ell \in \{0,1\}^n$.
Select uniformly and independently $\sigma^1,...,\sigma^\ell \in \{0,1\}$.
(2) For every non-empty $J \subseteq [\ell]$, compute $r^J = \oplus_{j\in J} s^j$ and $\rho^J = \oplus_{j\in J}\sigma^j$.
(3) For $i = 1,...,n$ determine the bit z_i according to the majority vote of the $(2^\ell - 1)$-long sequence of bits $(\rho^J \oplus B(f(x), r^J \oplus e^i))_{\emptyset\neq J\subseteq[\ell]}$.
(4) Output $z_1 \cdots z_n$.

Note that the "voting scheme" employed in Step 3 uses pairwise independent samples (i.e., the r^J's), but works essentially as well as it would have worked with independent samples (i.e., the independent r's).[6] That is, for every i and J, it holds that $\Pr_{s^1,...,s^\ell}[B(f(x),r^J\oplus e^i) = b(x,r^J\oplus e^i)] > (1/2)+\varepsilon$, where $r^J = \oplus_{j\in J} s^j$, and (for every fixed i) the events corresponding to different J's are pairwise independent. It follows that *if for every $j \in [\ell]$ it holds that $\sigma^j = b(x,s^j)$*, then for every i and J we have

$$\Pr_{s^1,...,s^\ell}[\rho^J \oplus B(f(x),r^J\oplus e^i) = b(x,e^i)] \tag{7.5}$$

[5] Alternatively, we can try all polynomially many possible guesses. In such a case, we shall output a list of candidates that, with high probability, contains x. (See Exercise 7.6.)

[6] Our focus here is on the accuracy of the approximation obtained by the sample, and not so much on the error probability. We wish to approximate $\Pr[b(x,r) \oplus B(f(x), r\oplus e^i) = 1]$ up to an additive term of ε, because such an approximation allows to correctly determine $b(x,e^i)$. A pairwise independent sample of $O(t/\varepsilon^2)$ points allows for an approximation of a value in $[0,1]$ up to an additive term of ε with error probability $1/t$, whereas a totally random sample of the same size yields error probability $\exp(-t)$. Since we can afford setting $t = \text{poly}(n)$ and having error probability $1/2n$, the difference in the error probability between the two approximation schemes is not important here. For a wider perspective see Appendix D.1.2 and D.3.

$$= \Pr_{s^1,\ldots,s^\ell}[B(f(x), r^J \oplus e^i) = b(x, r^J \oplus e^i)] \; > \; \frac{1}{2} + \varepsilon$$

where the equality is due to $\rho^J = \oplus_{j\in J}\sigma^j = b(x, r^J) = b(x, r^J \oplus e^i) \oplus b(x, e^i)$. Note that Eq. (7.5) refers to the correctness of a single vote for $b(x, e^i)$. Using $m = 2^\ell - 1 = O(n/\varepsilon^2)$ and noting that these (Boolean) votes are pairwise independent, we infer that the probability that the majority of these votes is wrong is upper-bounded by $1/2n$. Using a union bound on all i's, we infer that with probability at least $1/2$, all majority votes are correct and thus x is retrieved correctly. Recall that the foregoing is conditioned on $\sigma^j = b(x, s^j)$ for every $j \in [\ell]$, which in turn holds with probability $2^{-\ell} = (m+1)^{-1} = \Omega(\varepsilon^2/n) = 1/\mathrm{poly}(n)$. Thus, x is retrieved correctly with probability $1/\mathrm{poly}(n)$, and the theorem follows. □

Digest. Looking at the proof of Theorem 7.7, we note that it actually refers to an arbitrary black-box $B_x(\cdot)$ that approximates $b(x,\cdot)$; specifically, in the case of Theorem 7.7 we used $B_x(r) \stackrel{\mathrm{def}}{=} B(f(x), r)$. In particular, the proof does not use the fact that we can verify the correctness of the preimage recovered by the described process. Thus, the proof actually establishes *the existence of a* $\mathrm{poly}(n/\varepsilon)$*-time oracle machine that, for every* $x \in \{0,1\}^n$*, given oracle access to any* $B_x : \{0,1\}^n \to \{0,1\}$ *satisfying*

$$\Pr_{r\in\{0,1\}^n}[B_x(r) = b(x,r)] \geq \frac{1}{2} + \varepsilon \tag{7.6}$$

outputs x *with probability at least* $\mathrm{poly}(\varepsilon/n)$. Specifically, x is output with probability at least $p \stackrel{\mathrm{def}}{=} \Omega(\varepsilon^2/n)$. Noting that x is merely a string for which Eq. (7.6) holds, it follows that the number of strings that satisfy Eq. (7.6) is at most $1/p$. Furthermore, by iterating the foregoing procedure for $\widetilde{O}(1/p)$ times we can obtain all these strings (see Exercise 7.7).

Theorem 7.8 (Theorem 7.7, revisited): *There exists a probabilistic oracle machine that, given parameters* n, ε *and oracle access to any function* $B : \{0,1\}^n \to \{0,1\}$*, halts after* $\mathrm{poly}(n/\varepsilon)$ *steps and with probability at least* $1/2$ *outputs a list of all strings* $x \in \{0,1\}^n$ *that satisfy*

$$\Pr_{r\in\{0,1\}^n}[B(r) = b(x,r)] \geq \frac{1}{2} + \varepsilon,$$

where $b(x,r)$ *denotes the inner-product mod 2 of* x *and* r.

This machine can be modified such that, with high probability, its output list does not include any string x such that $\Pr_{r\in\{0,1\}^n}[B(r) = b(x,r)] < \frac{1}{2} + \frac{\varepsilon}{2}$.

Theorem 7.8 means that if given some information about x it is hard to recover x, then given the same information and a random r it is hard to predict $b(x,r)$. This assertion is proved by the counter-positive (see Exercise 7.14). Indeed, the foregoing statement is in the spirit of Theorem 7.7 itself, except that it refers to any "information about x" (rather than to the value $f(x)$). To demonstrate the point, let us rephrase the foregoing statement as follows: *for every randomized process* Π*, if given* s *it is hard to obtain* $\Pi(s)$ *then given* s *and a random* r *it is hard to predict* $b(\Pi(s), r)$.[7]

A coding theory perspective. Theorem 7.8 can be viewed as a list decoding procedure for the Hadamard Code, where the Hadamard encoding of a string $x \in \{0,1\}^n$ is the 2^n-bit long string containing $b(x, r)$ for every $r \in \{0,1\}^n$. In contrast to *standard decoding* in which the task is recovering the unique information that is encoded in the codeword that is closest to the given string, in list decoding the task is recovering all strings having encoding that is at a specified distance from the given string.[8] We mention that list decoding is applicable and valuable in the case that the specified distance does not allow for unique decoding (i.e., the specified distance is greater than half the distance of the code).

Applications of hard-core predicates. Turning back to hard-core predicates, we mention that they play a central role in the construction of general-purpose pseudorandom generators (see Section 8.2), commitment schemes and zero-knowledge proofs (see Sections 9.2.2 and C.4.3), and encryption schemes (see Appendix C.5).

7.1.4 Reflections on hardness amplification

Let us take notice that something truly amazing happens in Theorems 7.5 and 7.7. We are not talking merely of using an assumption to derive some conclusion; this is common practice in Mathematics and Science (and was indeed done several times in previous chapters, starting with Theorem 2.28). The thing that is special about Theorems 7.5 and 7.7 (and we shall see more of this in Section 7.2 as well as in Sections 8.2 and 8.3) is that a relatively mild intractability assumption is shown to imply a stronger intractability result.

This strengthening of an intractability phenomenon (a.k.a hardness amplification) takes place while we admit that we do not understand the intractability phenomenon (because we do not understand the nature of efficient computation). Nevertheless, hardness amplification is enabled by the use of the counter-positive, which in this case is called a reducibility argument. At this point things look less miraculous: a reducibility argument calls for the design of a procedure (i.e., a reduction) and a probabilistic analysis of its behavior. The design and analysis of such procedures may not be easy, but it is certainly within the standard expertise of computer science. The fact that hardness amplification is achieved via this counter-positive is best represented in the statement of Theorem 7.8.

---

[7] Indeed, Theorem 7.7 is obtained as a special case by letting $\Pi(s)$ be uniformly distributed in $f^{-1}(s)$.

[8] Further discussion of error-correcting codes and list-decoding is provided in Appendix E.1.

7.2 Hard Problems in E

As in Section 7.1, we start with the assumption $\mathcal{P} \neq \mathcal{NP}$ and seek to use it to our benefit. Again, we shall actually use a seemingly stronger assumption; here the strengthening is in requiring *worst-case* hardness with respect to *non-uniform* models of computation (rather than average-case hardness with respect to the standard uniform model). Specifically, we shall assume that $\mathcal{NP}$ cannot be solved by (non-uniform) families of polynomial-size circuits; that is, $\mathcal{NP}$ is not contained in $\mathcal{P}/\text{poly}$ (even not infinitely often).

Our goal is to transform this worst-case assumption into an average-case condition, which is useful for our applications. Since the transformation will not yield a problem in $\mathcal{NP}$ but rather one in $\mathcal{E}$, we might as well take the seemingly weaker assumption by which $\mathcal{E}$ is not contained in $\mathcal{P}/\text{poly}$ (see Exercise 7.9). That is, our starting point is actually that *there exists an exponential-time solvable decision problem such that any family of polynomial-size circuit fails to solve it correctly on all but finitely many input lengths.*[9]

A different perspective on our assumption is provided by the fact that $\mathcal{E}$ contains problems that cannot be solved in polynomial-time (cf.. Section 4.2.1). The current assumption goes beyond this fact by postulating the failure of non-uniform polynomial-time machines rather than the failure of (uniform) polynomial-time machines.

Recall that our goal is to obtain a predicate (i.e., a decision problem) that is computable in exponential-time but is inapproximable by polynomial-size circuits. For sake of later developments, we formulate a general notion of inapproximability.

Definition 7.9 (inapproximability, a general formulation): *We say that* $f : \{0,1\}^* \to \{0,1\}$ *is* (S,ρ)-inapproximable *if for every family of* S*-size circuits* $\{C_n\}_{n\in\mathbb{N}}$ *and all sufficiently large* n *it holds that*

$$\mathsf{Pr}[C_n(U_n) \neq f(U_n)] \geq \frac{\rho(n)}{2} \tag{7.7}$$

We say that f *is* T-inapproximable *if it is* $(T, 1-(1/T))$*-inapproximable.*

We chose the specific form of Eq. (7.7) such that the "level of inapproximability" represented by the parameter ρ will range in $(0,1)$ and increase with the value of ρ. Specifically, (almost-everywhere) *worst-case* hardness for circuits of size S is represented by (S,ρ)-inapproximability with $\rho(n) = 2^{-n+1}$ (i.e., in this case $\mathsf{Pr}[C(U_n) \neq f(U_n)] \geq 2^{-n}$ for every circuit C_n of size $S(n)$). On the other hand, no predicate can be (S,ρ)-inapproximable for $\rho(n) = 1 - 2^{-n}$ even with $S(n) = O(n)$ (i.e., $\mathsf{Pr}[C(U_n) = f(U_n)] \geq 0.5 + 2^{-n-1}$ holds for some linear-size circuit; see Exercise 7.10).

We note that Eq. (7.7) can be interpreted as an upper-bound on the *correlation* of each adequate circuit with f (i.e., Eq. (7.7) is equivalent to $\mathsf{E}[\chi(C(U_n), f(U_n))] \leq$

[9]Note that our starting point is actually stronger than assuming the existence of a function f in $\mathcal{E} \setminus \mathcal{P}/\text{poly}$. Such an assumption would mean that any family of polynomial-size circuit fails to compute f correctly on infinitely many input lengths, whereas our starting point postulates failures on all but finitely many lengths.

$1 - \rho(n)$, where $\chi(\sigma, \tau) = 1$ if $\sigma = \tau$ and $\chi(\sigma, \tau) = -1$ otherwise).[10] Thus, T-inapproximability means that no family of size T circuits can correlate f better than $1/T$.

We note that the existence of a non-uniformly hard one-way function (as in Definition 7.3) implies the existence of an exponential-time computable predicate that is T-inapproximable for every polynomial T. (For details see Exercise 7.24.) However, our goal in this section is to establish this conclusion under a seemingly weaker assumption.

On almost everywhere hardness. We highlight the fact that both our assumptions and conclusions refer to *almost everywhere* hardness. For example, our starting point is not merely that $\mathcal{E}$ is not contained in $\mathcal{P}/\text{poly}$ (or in other circuit size classes to be discussed), but rather that this is the case almost everywhere. Note that by saying that f has circuit complexity exceeding S, we merely mean that *there are infinitely many n's* such that no circuit of size $S(n)$ can compute f correctly on all inputs of length n. In contrast, by saying that f has circuit complexity exceeding S almost everywhere, we mean that *for all but finite many n's* no circuit of size $S(n)$ can computes f correctly on all inputs of length n. (Indeed, it is not known whether an "infinitely often" type of hardness implies a corresponding "almost everywhere" hardness.)

The class $\mathcal{E}$. Recall that $\mathcal{E}$ denote the class of exponential-time solvable decision problems (equivalently, exponential-time computable Boolean predicates); that is, $\mathcal{E} = \cup_\varepsilon \text{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{\varepsilon n}$.

The rest of this section. We start (in Section 7.2.1) with a treatment of assumptions and hardness amplification regarding polynomial-size circuits, which suffice for non-trivial derandomization of $\mathcal{BPP}$. We then turn (in Section 7.2.2) to assumptions and hardness amplification regarding exponential-size circuits, which yield a "full" derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$). In fact, both sections contain material that is applicable to various other circuit-size bounds, but the motivational focus is as stated.

Teaching note: Section 7.2.2 is advanced material, which is best left for independent reading. Furthermore, for one of the central results (i.e., Lemma 7.23) only an outline is provided and the interested reader is referred to the original paper [127].

7.2.1 Amplification wrt polynomial-size circuits

Our goal here is to prove the following result.

Theorem 7.10 *Suppose that for every polynomial p there exists a problem in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than p. Then there exist polynomial-inapproximable Boolean functions in $\mathcal{E}$; that is, for every polynomial p there exists a p-inapproximable Boolean function in $\mathcal{E}$.*

[10] Indeed, $\mathsf{E}[\chi(X,Y)] = \mathsf{Pr}[X=Y] - \mathsf{Pr}[X \neq Y] = 1 - 2\mathsf{Pr}[X \neq Y]$.

Theorem 7.10 is used towards deriving a meaningful derandomization of $\mathcal{BPP}$ under the aforementioned assumption (see Part 2 of Theorem 8.19). We present two proofs of Theorem 7.10. The first proof proceeds in two steps:

1. Starting from the worst-case hypothesis, we first establish some mild level of average-case hardness (i.e., a mild level of inapproximability). Specifically, we show that for every polynomial p there exists a problem in $\mathcal{E}$ that is (p,ε)-inapproximable for $\varepsilon(n) = 1/n^3$.

2. Using the foregoing mild level of inapproximability, we obtain the desired strong level of inapproximability (i.e., p'-inapproximability for every polynomial p'). Specifically, for every two polynomials p_1 and p_2, we prove that *if the function f is $(p_1, 1/p_2)$-inapproximable, then the function $F(x_1,...,x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $t(n) = n \cdot p_2(n)$ and $x_1,...,x_{t(n)} \in \{0,1\}^n$, is p'-inapproximable for $p'(t(n) \cdot n) = p_1(n)^{\Omega(1)}/\mathrm{poly}(t(n))$.* This claim is known as Yao's XOR Lemma and its proof is far more complex than the proof of its information theoretic analogue (discussed at the beginning of §7.2.1.2).

The second proof of Theorem 7.10 consists of showing that the construction employed in the first step, when composed with Theorem 7.8, actually yields the desired end result. This proof will uncover a connection between hardness amplification and coding theory. Our presentation will thus proceed in three corresponding steps (presented in §7.2.1.1-7.2.1.3, and schematically depicted in Figure 7.2).

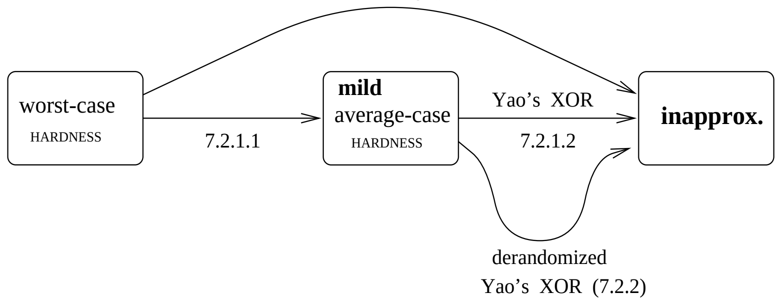


Figure 7.2: Proofs of hardness amplification: organization

7.2.1.1 From worst-case hardness to mild average-case hardness

The transformation of worst-case hardness into average-case hardness (even in a mild sense) is indeed remarkable. Note that worst-case hardness may be due to a relatively small number of instances, whereas even mild forms of average-case hardness refer to a very large number of possible instances.[11] In other words, we should transform hardness that may occur on a negligible fraction of the instances

[11] Indeed, worst-case hardness with respect to polynomial-size circuits cannot be due to a polynomial number of instances, because a polynomial number of instances can be hard-wired into

into hardness that occurs on a noticeable fraction of the instances. Intuitively, we should "spread" the hardness of few instances (of the original problem) over all (or most) instances (of the transformed problem). The counter-positive view is that computing the value of typical instances of the transformed problem should enable solving the original problem on every instance.

The aforementioned transformation is based on the *self-correction paradigm*, to be reviewed first. The paradigm refers to functions g that can be evaluated at any desired point by using the value of g at a few random points, where each of these points is uniformly distributed in the function's domain (but indeed the points are not independently distributed). The key observation is that if $g(x)$ can be reconstructed based on the value of g at t such random points, then such a reconstruction can tolerate a $1/3t$ fraction of errors (regarding the values of g). Thus, if we can correctly obtain the value of g on all but at most a $1/3t$ fraction of its domain, then we can probabilistically recover the correct value of g at any point with very high probability. It follows that if no probabilistic polynomial-time algorithm can correctly compute g *in the worst-case sense*, then every probabilistic polynomial-time algorithm must fail to correctly compute g *on more than a* $1/3t$ *fraction of its domain*.

The archetypical example of a self-correctable function is any m-variate polynomial of individual degree d over a finite field F such that $|F| > dm + 1$. The value of such a polynomial at any desired point x can be recovered based on the values of $dm + 1$ points (other than x) that reside on a random line that passes through x. Note that each of these points is uniformly distributed in F^m, which is the function's domain. (For details, see Exercise 7.11.)

Recall that we are given an arbitrary function $f \in \mathcal{E}$ that is hard to compute in the worst-case. Needless to say, this function is not necessarily self-correctable (based on relatively few points), but it can be extended into such a function. Specifically, we extend $f : [N] \to \{0,1\}$ (viewed as $f : [N^{1/m}]^m \to \{0,1\}$) to an m-variate polynomial of individual degree d over a finite field F such that $|F| > dm+1$ and $(d+1)^m = N$. Intuitively, in terms of worst-case complexity, the extended function is at least as hard as f, which means that it is hard (in the worst-case). The point is that the extended function is self-correctable and thus its worst-case hardness implies that it must be at least mildly hard in the average-case. Details follow.

Construction 7.11 (multi-variate extension)[12]: *For any function $f_n : \{0,1\}^n \to \{0,1\}$, a finite field F, a set $H \subset F$ and an integer m such that $|H|^m = 2^n$ and $|F| > (|H| - 1)m + 1$, we consider the function $\hat{f}_n : F^m \to F$ defined as the m-variate polynomial of individual degree $|H|-1$ that extends $f_n : H^m \to \{0,1\}$. That*

such circuits. Still, for all we know, worst-case hardness may be due to a small super-polynomial number of instances (e.g., $n^{\log_2 n}$ instances). In contrast, even mild forms of average-case hardness must be due to an exponential number of instances (i.e., $2^n/\mathrm{poly}(n)$ instances).

[12]The algebraic fact underlying this construction is that for any function $f : H^m \to F$ there exists a unique m-variate polynomial $\hat{f} : F^m \to F$ of individual degree $|H| - 1$ such that for every $x \in H^m$ it holds that $\hat{f}(x) = f(x)$. This polynomial is called a multi-variate polynomial extension of f, and it can be found in $\mathrm{poly}(|H|^m \log |F|)$-time. For details, see Exercise 7.12.

is, we identify $\{0,1\}^n$ with H^m, and define $\hat{f}_n$ as the unique m-variate polynomial of individual degree $|H|-1$ that satisfies $\hat{f}_n(x) = f_n(x)$ for every $x \in H^m$, where we view $\{0,1\}$ as a subset of F.

Note that $\hat{f}_n$ can be evaluated at any desired point, by evaluating f_n on its entire domain, and determining the unique m-variate polynomial of individual degree $|H|-1$ that agrees with f_n on H^m (see Exercise 7.12). Thus, for $f : \{0,1\}^* \to \{0,1\}$ in $\mathcal{E}$, the corresponding $\hat{f}$ (defined by separately extending the restriction of f to each input length) is also in $\mathcal{E}$. For the sake of preserving various complexity measures, we wish to have $|F^m| = \mathrm{poly}(2^n)$, which leads to setting $m = n/\log_2 n$ (yielding $|H| = n$ and $|F| = \mathrm{poly}(n)$). In particular, in this case $\hat{f}_n$ is defined over strings of length $O(n)$. The mild average-case hardness of $\hat{f}$ follows by the forgoing discussion. In fact, we state and prove a more general result.

Theorem 7.12 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S. Then, there exists an exponential-time computable function $\hat{f} : \{0,1\}^* \to \{0,1\}^*$ such that $|\hat{f}(x)| \le |x|$ and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = S(n'/O(1))/\mathrm{poly}(n')$ it holds that $\mathsf{Pr}[C'_{n'}(U_{n'}) \neq \hat{f}(U_{n'})] > (1/n')^2$. Furthermore, $\hat{f}$ does not depend on S.*

Theorem 7.12 seems to complete the first step of the proof of Theorem 7.10, except that we desire a Boolean function rather than a function that merely does not stretch its input. The extra step of obtaining a Boolean function that is $(\mathrm{poly}(n), n^{-3})$-inapproximable is taken in Exercise 7.13.[13] Essentially, if $\hat{f}$ is hard to compute on a noticeable fraction of its inputs then the Boolean predicate that on input (x, i) returns the i^{th} bit of $\hat{f}(x)$ must be mildly inapproximable.

Proof Sketch: Given f as in the hypothesis and for every $n \in \mathbb{N}$, we consider the restriction of f to $\{0,1\}^n$, denoted f_n, and apply Construction 7.11 to it, while using $m = n/\log n$, $|H| = n$ and $n^2 < |F| = \mathrm{poly}(n)$. Recall that the resulting function $\hat{f}_n$ maps strings of length $n' = \log_2 |F^m| = O(n)$ to strings of length $\log_2 |F| = O(\log n)$. Following the foregoing discussion, we shall show that circuits that approximate $\hat{f}_n$ too well yield circuits that compute f_n correctly on each input. Using the hypothesis regarding the size of the latter, we shall derive a lower-bound on the size of the former. The actual (reducibility) argument proceeds as follows. We fix an arbitrary circuit $C'_{n'}$ that satisfies

$$\mathsf{Pr}[C'_{n'}(U_{n'}) = \hat{f}_n(U_{n'})] \ge 1 - (1/n')^2 > 1 - (1/3t), \tag{7.8}$$

where $t \stackrel{\text{def}}{=} (|H|-1)m + 1 = o(n^2)$ exceeds the total degree of $\hat{f}_n$. Using the self-correction feature of $\hat{f}_n$, we observe that by making t oracle calls to $C'_{n'}$ we can probabilistically recover the value of ($\hat{f}_n$ and thus of) f_n on each input, with probability at least 2/3. Using error-reduction and (non-uniform) derandomization as in

[13] A quantitatively stronger bound can be obtained by noting that the proof of Theorem 7.12 actually establishes an error lower-bound of $\Omega((\log n')/(n')^2)$ and that $|\hat{f}(x)| = O(\log |x|)$.

the proof of Theorem 6.3,[14] we obtain a circuit of size $n^3 \cdot |C'_{n'}|$ that computes f_n. By the hypothesis $n^3 \cdot |C'_{n'}| > S(n)$, and so $|C'_{n'}| > S(n'/O(1))/\text{poly}(n')$. Recalling that $C'_{n'}$ is an arbitrary circuit that satisfies Eq. (7.8), the theorem follows. □

Digest. The proof of Theorem 7.12 is actually a worst-case to average-case reduction. That is, the proof consists of a self-correction procedure that allows for the evaluation of f at any desired n-bit long point, using oracle calls to any circuit that computes $\hat{f}$ correctly on a $1-(1/n')^2$ fraction of the n'-bit long inputs. We recall that if $f \in \mathcal{E}$ then $\hat{f} \in \mathcal{E}$, but we do not know how to preserve the complexity of f in case it is in $\mathcal{NP}$. (Various indications to the difficulty of a worst-case to average-case reduction for $\mathcal{NP}$ are known; see, e.g., [42].)

We mention that the ideas underlying the proof of Theorem 7.12 have been applied in a large variety of settings. For example, we shall see applications of the self-correction paradigm in §9.3.2.1 and in §9.3.2.2. Furthermore, in §9.3.2.2 we shall re-encounter the very same multi-variate extension used in the proof of Theorem 7.12.

7.2.1.2 Yao's XOR Lemma

Having obtained a mildly inapproximable predicate, we wish to obtain a strongly inapproximable one. The information theoretic context provides an appealing suggestion: Suppose that X is a Boolean random variable (representing the mild inapproximability of the aforementioned predicate) that equals 1 with probability ε. Then XORing the outcome of n/ε independent samples of X yields a bit that equals 1 with probability $0.5 \pm \exp(-\Omega(n))$. It is tempting to think that the same should happen in the computational setting. That is, if f is hard to approximate correctly with probability exceeding $1-\varepsilon$ then XORing the output of f on n/ε non-overlapping parts of the input should yield a predicate that is hard to approximate correctly with probability that is non-negligibly higher than $1/2$. The latter assertion turns out to be correct, but (even more than in Section 7.1.2) the proof of the computational phenomenon is considerably more complex than the analysis of the information theoretic analogue.

Theorem 7.13 (Yao's XOR Lemma): *There exist a universal constant $c > 0$ such that the following holds. If, for some polynomials p_1 and p_2, the Boolean function f is $(p_1, 1/p_2)$-inapproximable, then the function $F(x_1, ..., x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $t(n) = n \cdot p_2(n)$ and $x_1, ..., x_{t(n)} \in \{0,1\}^n$, is p'-inapproximable for $p'(t(n) \cdot n) = p_1(n)^c / t(n)^{1/c}$. Furthermore, the claim holds also if the polynomials p_1 and p_2 are replaced by any integer functions.*

[14] First, we apply the foregoing probabilistic procedure $O(n)$ times and take a majority vote. This yields a probabilistic procedure that, on input $x \in \{0,1\}^n$, invokes $C'_{n'}$ for $o(n^3)$ times and computes $f_n(x)$ correctly with probability greater than $1-2^{-n}$. Finally, we just fix a sequence of random choices that is good for all 2^n possible inputs, and obtain a circuit of size $n^3 \cdot |C'_{n'}|$ that computes f_n correctly on every n-bit input.

Combining Theorem 7.12 (and Exercise 7.13), and Theorem 7.13, we obtain a proof of Theorem 7.10. (Recall that an alternative proof is presented in §7.2.1.3.)

We note that proving Theorem 7.13 seems more difficult than proving Theorem 7.5 (i.e., the amplification of one-way functions), due to two issues. Firstly, unlike in Theorem 7.5, the computational problems are not in $\mathcal{PC}$ and thus we cannot efficiently recognize correct solutions to them. Secondly, unlike in Theorem 7.5, solutions to instances of the transformed problem do not correspond of the concatenation of solutions for the original instances, but are rather a function of the latter that losses almost all the information about the latter. The proof of Theorem 7.13 presented next deals with each of these two difficulties separately.

Several different proofs of Theorem 7.13 are known. As just stated, the proof that we present is conceptually appealing because it deal separately with two unrelated difficulties. Furthermore, this proof benefits most from the material already presented in Section 7.1. The proof proceeds in two steps:

1. First we prove that the corresponding "direct product" function $P(x_1, ..., x_{t(n)}) = (f(x_1), ..., f(x_{t(n)}))$ is difficult to compute in a strong average-case sense.
2. Next we establish the desired result by an application of Theorem 7.8.

Thus, given Theorem 7.8, our main focus is on the first step, which is of independent interest (and is thus generalized from Boolean functions to arbitrary ones).

Theorem 7.14 (The Direct Product Lemma): *Let p_1 and p_2 be two polynomials, and suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is such that for every family of p_1-size circuits, $\{C_n\}_{n\in\mathbb{N}}$, and all sufficiently large $n \in \mathbb{N}$, it holds that $\Pr[C_n(U_n) \neq f(U_n)] > 1/p_2(n)$. Let $P(x_1, ..., x_{t(n)}) = (f(x_1), ..., f(x_{t(n)}))$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p_2(n)$. Then, for any $\varepsilon' : \mathbb{N} \to [0,1]$, setting p' such that $p'(t(n)\cdot n) = p_1(n)/\mathrm{poly}(t(n)/\varepsilon'(t(n)\cdot n))$, it holds that every family of p'-size circuits, $\{C'_m\}_{m\in\mathbb{N}}$, satisfies $\Pr[C'_m(U_m) = P(U_m)] < \varepsilon'(m)$. Furthermore, the claim holds also if the polynomials p_1 and p_2 are replaced by any integer functions.*

In particular, for an adequate constant $c > 0$, selecting $\varepsilon'(t(n)\cdot n) = p_1(n)^{-c}$, we obtain $p'(t(n)\cdot n) = p_1(n)^c/t(n)^{1/c}$, and so $\varepsilon'(m) \le 1/p'(m)$.

Deriving Theorem 7.13 from Theorem 7.14. Theorem 7.13 follows from Theorem 7.14 by considering the function $P'(x_1, ..., x_{t(n)}, r) = b(f(x_1)\cdots f(x_{t(n)}), r)$, where f is a Boolean function, $r \in \{0,1\}^{t(n)}$, and $b(y,r)$ is the inner-product modulo 2 of the $t(n)$-bit long strings y and r. Note that, for the corresponding P, we have $P'(x_1, ..., x_{t(n)}, r) \equiv b(P(x_1, ..., x_{t(n)}), r)$, whereas $F(x_1, ..., x_{t(n)}) = P'(x_1, ..., x_{t(n)}, 1^{t(n)})$. Intuitively, the inapproximability of P' should follow from the strong average-case hardness of P (via Theorem 7.8), whereas it should be possible to reduce the approximation of P' to the approximation of F (and thus derive the desired inapproximability of F). Indeed, this intuition does not fail, but detailing the argument seems a bit cumbersome (and so we only provide the clues here). Assuming that f is $(p_1, 1/p_2)$-inapproximable, we first apply Theorem 7.14 (with $\varepsilon'(t(n)\cdot n) = p_1(n)^{-c}$) and then apply Theorem 7.8 (see Exercise 7.14), inferring that P' is p'-inapproximable for $p'(t(n) \cdot n) = p_1(n)^{\Omega(1)}/\mathrm{poly}(t(n))$. The less obvious part of the argument is reducing the approximation of P' to the approximation of F. The key observation is that

$$P'(x_1, ..., x_{t(n)}, r) = F(z_1, ..., z_{t(n)}) \oplus \bigoplus_{i:r_i=0} f(z_i) \qquad (7.9)$$

where $z_i = x_i$ if $r_i = 1$ and is an arbitrary n-bit long string otherwise. Now, if somebody provides us with samples of the distribution $(U_n, f(U_n))$, then we can use these samples in the role of the pairs $(z_i, f(z_i))$ for the indices i that satisfy $r_i = 0$. Considering a best choice of such samples (i.e., one for which we obtain the best approximation of P'), we obtain a circuit that approximates P' (by using a circuit that approximates F and the said choices of samples). (The details are left for Exercise 7.17.) Theorem 7.13 follows.

Proving Theorem 7.14. Note that Theorem 7.14 is closely related to Theorem 7.5; see Exercise 7.20 for details. This suggests employing an analogous proof strategy; that is, converting circuits that violate the theorem's conclusion into circuits that violate the theorem's hypothesis. We note, however, that things were much simpler in the context of Theorem 7.5: there we could (efficiently) check whether or not a value contained in the output of the circuit that solves the direct-product problem constitutes a correct answer for the corresponding instance of the basic problem. Lacking such an ability in the current context, we shall have to use such values more carefully. Loosely speaking, we shall take a weighted majority vote among various answers, where the weights reflect our confidence in the correctness of the various answers.

We establish Theorem 7.14 by applying the following lemma that provides quantitative bounds on the feasibility of computing the direct product of two functions. In this lemma, $\{Y_m\}_{m\in\mathbb{N}}$ and $\{Z_m\}_{m\in\mathbb{N}}$ are independent probability ensembles such that $Y_m, Z_m \in \{0,1\}^m$, and $X_n = (Y_{\ell(n)}, Z_{n-\ell(n)})$ for some function $\ell : \mathbb{N} \to \mathbb{N}$. The lemma refers to the success probability of computing the direct product function $F : \{0,1\}^* \to \{0,1\}^*$ defined by $F(yz) = (F_1(y), F_2(z))$, where $|y| = \ell(|yz|)$, when given bounds on the success probability of computing F_1 and F_2 (separately). Needless to say, these probability bounds refer to circuits of certain sizes. We stress that *the lemma is not symmetric with respect to the two functions*: *it guarantees a stronger* (and in fact lossless) *preservation of circuit sizes for one of the functions* (which is arbitrarily chosen to be F_1).

Lemma 7.15 (Direct Product, a quantitative two argument version): *For $\{Y_m\}$, $\{Z_m\}$, F_1, F_2, ℓ, $\{X_n\}$ and F as in the foregoing, let $\rho_1(\cdot)$ be an upper-bound on the success probability of $s_1(\cdot)$-size circuits in computing F_1 over $\{Y_m\}$. That is, for every such circuit family $\{C_m\}$*

$$\Pr[C_m(Y_m) = F_1(Y_m)] \leq \rho_1(m).$$

Likewise, suppose that $\rho_2(\cdot)$ is an upper-bound on the probability that $s_2(\cdot)$-size circuits compute F_2 over $\{Z_m\}$. Then, for every function $\varepsilon : \mathbb{N} \to \mathbb{R}$, the function

ρ defined as

$$\rho(n) \stackrel{\text{def}}{=} \rho_1(\ell(n)) \cdot \rho_2(n - \ell(n)) + \varepsilon(n)$$

is an upper-bound on the probability that families of $s(\cdot)$-size circuits correctly compute F over $\{X_n\}$, where

$$s(n) \stackrel{\text{def}}{=} \min\left\{ s_1(\ell(n)) \ , \ \frac{s_2(n - \ell(n))}{\text{poly}(n/\varepsilon(n))} \right\}.$$

Theorem 7.14 is derived from Lemma 7.15 by using a *careful induction*, which capitalizes on the highly quantitative form of Lemma 7.15 and in particular on the fact that no loss is incurred for one of the two functions that are used. We first detail this argument, and next establish Lemma 7.15 itself.

Deriving Theorem 7.14 from Lemma 7.15. We write $P(x_1, x_2, ..., x_{t(n)})$ as $P^{(t(n))}(x_1, x_2, ..., x_{t(n)})$, where $P^{(i)}(x_1, ..., x_i) = (f(x_1), ..., f(x_i))$ and $P^{(i)}(x_1, ..., x_i) \equiv (P^{(i-1)}(x_1, ..., x_{i-1}), f(x_i))$. For any function ε, *we shall prove by induction on i that circuits of size $s(n) = p_1(n)/\text{poly}(t(n)/\varepsilon(n))$ cannot compute $P^{(i)}(U_{i \cdot n})$ with success probability greater than* $(1-(1/p_2(n))^i + (i-1)\cdot\varepsilon(n)$, where p_1 and p_2 are as in Theorem 7.14. Thus, no $s(n)$-size circuit can compute $P^{(t(n))}(U_{t(n)\cdot n})$ with success probability greater than $(1-(1/p_2(n))^{t(n)} + (t(n)-1)\cdot\varepsilon(n) = \exp(-n) + (t(n) - 1)\cdot\varepsilon(n)$. Recalling that this is established for any function ε, Theorem 7.14 follows (by using $\varepsilon(n) = \varepsilon'(t(n)\cdot n)/t(n)$, and observing that the setting $s(n) = p'(t(n)\cdot n)$ satisfies $s(n) = p_1(n)/\text{poly}(t(n)/\varepsilon(n))$).

Turning to the induction itself, we first note that its basis (i.e., $i = 1$) is guaranteed by the theorem's hypothesis (i.e., the hypothesis of Theorem 7.14 regarding f). The induction step (i.e., from i to $i+1$) will be proved by using Lemma 7.15 with $F_1 = P^{(i)}$ and $F_2 = f$, along with the parameter setting $\rho_1^{(i)}(i \cdot n) = (1-(1/p_2(n))^i + (i-1)\cdot\varepsilon(n)$, $s_1^{(i)}(i\cdot n) = s(n)$, $\rho_2^{(i)}(n) = 1 - (1/p_2(n))$ and $s_2^{(i)}(n) = \text{poly}(n/\varepsilon(n)) \cdot s(n) = p_1(n)$. Details follow.

Note that the induction hypothesis (regarding $P^{(i)}$) implies that F_1 satisfies the hypothesis of Lemma 7.15 (w.r.t size $s_1^{(i)}$ and success rate $\rho_1^{(i)}$), whereas the theorem's hypothesis regarding f implies that F_2 satisfies the hypothesis of Lemma 7.15 (w.r.t size $s_2^{(i)}$ and success rate $\rho_2^{(i)}$). Thus, $F = P^{(i+1)}$ satisfies the lemma's conclusion with respect to circuits of size $\min(s_1^{(i)}(i\cdot n), s_2^{(i)}(n)/\text{poly}(n/\varepsilon(n))) = s(n)$ and success rate $\rho_1^{(i)}(i\cdot n)\cdot\rho_2^{(i)}(n) + \varepsilon(n)$ which is upper-bounded by $(1-(1/p_2(n))^{i+1} + i\cdot\varepsilon(n)$. This completes the induction step.

We stress the fact that we used induction for a non-constant number of steps, and that this was enabled by the highly quantitative form of the inductive claim and the small loss incurred by the inductive step. Specifically, the size bound did not decrease during the induction (although we could afford a small additive loss in each step, but not a constant factor loss). Likewise, the success rate suffered an additive increase of $\varepsilon(n)$ in each step, which was accommodated by the inductive claim. Thus, *assuming the correctness of Lemma 7.15, we have established Theorem 7.14.* $\square$

Proof of Lemma 7.15: Proceeding (as usual) by the contrapositive, we consider a family of $s(\cdot)$-size circuits $\{C_n\}_{n\in\mathbb{N}}$ that violates the lemma's conclusion; that is, $\Pr[C_n(X_n) = F(X_n)] > \rho(n)$. We will show how to use such circuits in order to obtain either circuits that violate the lemma's hypothesis regarding F_1 or circuits that violate the lemma's hypothesis regarding F_2. Towards this end, it is instructive to write the success probability of C_n in a conditional form, while denoting the $i^{\rm th}$ output of $C_n(x)$ by $C_n(x)_i$ (i.e., $C_n(x) = (C_n(x)_1, C_n(x)_2)$):

$$\begin{aligned}
&\Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)}) = F(Y_{\ell(n)}, Z_{n-\ell(n)})] \\
&\quad = \Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_1 = F_1(Y_{\ell(n)})] \\
&\qquad \cdot \Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_2 = F_2(Z_{n-\ell(n)}) \mid C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_1 = F_1(Y_{\ell(n)})].
\end{aligned}$$

The basic idea is that if the first factor is greater than $\rho_1(\ell(n))$ then we immediately derive a circuit (i.e., $C'_n(y) = C_n(y, Z_{n-\ell(n)})_1$) contradicting the lemma's hypothesis regarding F_1, whereas if the second factor is significantly greater than $\rho_2(n - \ell(n))$ then we can obtain a circuit contradicting the lemma's hypothesis regarding F_2. The treatment of the latter case is indeed not obvious. The idea is that a sufficiently large sample of $(Y_{\ell(n)}, F_1(Y_{\ell(n)}))$, which may be hard-wired into the circuit, allows using the conditional probability space (in such a circuit) towards an attempt to approximate F_2. That is, on input z, we select uniformly a string y satisfying $C_n(y,z)_1 = F_1(y)$ (from the aforementioned sample), and output $C_n(y,z)_2$. For a fixed z, sampling of the conditional space (i.e., y's satisfying $C_n(y,z)_1 = F_1(y)$) is possible provided that $\Pr[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ holds with noticeable probability. The last caveat motivates a separate treatment of z's having a noticeable value of $\Pr[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ and of the rest of z's (which are essentially ignored). Details follow.

Let us first simplify the notations by fixing a generic n and using the abbreviations $C = C_n$, $\varepsilon = \varepsilon(n)$, $\ell = \ell(n)$, $Y = Y_\ell$, and $Z = Y_{n-\ell}$. We call z good if $\Pr[C(Y,z)_1 = F_1(Y)] \ge \varepsilon/2$ and let G be the set of good z's. Next, rather than considering the event $C(Y,Z) = F(Y,Z)$, we consider the combined event $C(Y,Z) = F(Y,Z) \wedge Z \in G$, which occurs with almost the same probability (up to an additive error term of $\varepsilon/2$). This is the case because, for any $z \notin G$, it holds that

$$\Pr[C(Y,z) = F(Y,z)] \le \Pr[C(Y,z)_1 = F_1(Y)] < \varepsilon/2$$

and thus z's that are not good do not contribute much to $\Pr[C(Y,Z) = F(Y,Z)]$; that is, $\Pr[C(Y,Z) = F(Y,Z) \wedge Z \in G]$ is lower-bounded by $\Pr[C(Y,Z) = F(Y,Z)] - \varepsilon/2$. Using $\Pr[C(Y,z) = F(Y,z)] > \rho(n) = \rho_1(\ell) \cdot \rho_2(n-\ell) + \varepsilon$, we have

$$\Pr[C(Y,Z) = F(Y,Z) \wedge Z \in G] > \rho_1(\ell) \cdot \rho_2(n-\ell) + \frac{\varepsilon}{2}. \tag{7.10}$$

We proceed according to the forgoing outline, first showing that if $\Pr[C(Y,Z)_1 = F_1(Y)] > \rho_1(\ell)$ then we immediately derive circuits violating the hypothesis concerning F_1. Actually, we prove something stronger (which we will actually need for the other case).

Claim 7.15.1: For every z, it holds that $\Pr[C(Y,z)_1 = F_1(Y)] \le \rho_1(\ell)$.

Proof: Otherwise, using any $z \in \{0,1\}^{n-\ell}$ that satisfies $\Pr[C(Y,z)_1 = F_1(Y)] > \rho_1(\ell)$, we obtain a circuit $C'(y) \stackrel{\rm def}{=} C(y,z)_1$ that contradicts the lemma's hypothesis concerning F_1. $\square$

Using Claim 7.15.1, we show how to obtain a circuit that violates the lemma's hypothesis concerning F_2, and doing so we complete the proof of the lemma.

Claim 7.15.2: There exists a circuit C'' of size $s_2(n-\ell)$ such that

$$\begin{aligned}\Pr[C''(Z)=F_2(Z)] \;&\geq\; \frac{\Pr[C(Y,Z)=F(Y,Z) \wedge Z\in G]}{\rho_1(\ell)} - \frac{\varepsilon}{2}\\ &>\; \rho_2(n-\ell)\end{aligned}$$

Proof: The second inequality is due to Eq. (7.10), and thus we focus on establishing the first inequality. We construct the circuit C'' as suggested in the foregoing outline. Specifically, we take a poly(n/ε)-large sample, denoted S, from the distribution $(Y, F_1(Y))$ and let $C''(z) \stackrel{\rm def}{=} C(y,z)_2$, where (y,v) is a uniformly selected among the elements of S for which $C(y,z)_1 = v$ holds. Details follow.

Let m be a sufficiently large number that is upper-bounded by a polynomial in n/ε, and consider a random sequence of m pairs, generated by taking m independent samples from the distribution $(Y, F_1(Y))$. We stress that we do not assume here that such a sample, denoted S, can be produced by an efficient (uniform) algorithm (but, jumping ahead, we remark that such a sequence can be fixed non-uniformly). For each $z \in G \subseteq \{0,1\}^{n-\ell}$, we denote by S_z the set of pairs $(y,v) \in S$ for which $C(y,z)_1 = v$. Note that S_z is a random sample of the residual probability space defined by $(Y, F_1(Y))$ conditioned on $C(Y,z)_1 = F_1(Y)$. Also, with overwhelmingly high probability, $|S_z| = \Omega(n/\varepsilon^2)$, because $z \in G$ implies $\Pr[C(Y,z)_1 = F_1(Y)] \geq \varepsilon/2$ and $m = \Omega(n/\varepsilon^3)$.[15] Thus, for each $z \in G$, with overwhelming probability (taken over the choices of S), the sample S_z provides a good approximation to the conditional probability space.[16] In particular, with probability greater than $1 - 2^{-n}$, it holds that

$$\frac{|\{(y,v) \in S_z : C(y,z)_2 = F_2(z)\}|}{|S_z|} \geq \Pr[C(Y,z)_2 = F_2(z) \,|\, C(Y,z)_1 = F_1(Y)] - \frac{\varepsilon}{2}. \tag{7.11}$$

Thus, with positive probability, Eq. (7.11) holds for all $z \in G \subseteq \{0,1\}^{n-\ell}$. The circuit C'' computing F_2 is now defined as follows. The circuit will contain a set $S = \{(y_i, v_i) : i = 1, ..., m\}$ (i.e., S is "hard-wired" into the circuit C'') such that the following two conditions hold:

1. For every $i \in [m]$ it holds that $v_i = F_1(y_i)$.
2. For each good z the set $S_z = \{(y,v) \in S : C(y,z)_1 = v\}$ satisfies Eq. (7.11).

   (In particular, S_z is not empty for any good z.)

[15] Note that the expected size of S_z is $m \cdot \varepsilon/2 = \Omega(n/\varepsilon^2)$. Using Chernoff Bound, we get $\Pr_S[|S_z| < m\varepsilon/4] = \exp(-\Omega(n/\varepsilon^2)) < 2^{-n}$.

[16] For $T_z = \{y : C(y,z)_1 = F_1(y)\}$, we are interested in a sample S' of T_z such that $|\{y \in S' : C(y,z)_2 = F_2(z)\}|/|S'|$ approximates $\Pr[C(Y,z)_2 = F_2(z) \,|\, Y \in T_z]$ up-to an additive term of $\varepsilon/2$. Using Chernoff Bound again, we note that a random $S' \subset T_z$ of size $\Omega(n/\varepsilon^2)$ provides such an approximation with probability greater than $1 - 2^{-n}$.

On input z, the circuit C'' first determines the set S_z, by running C for m times and checking, for each $i = 1, ..., m$, whether or not $C(y_i, z) = v_i$. In case S_z is empty, the circuit returns an arbitrary value. Otherwise, the circuit selects uniformly a pair $(y, v) \in S_z$ and outputs $C(y, z)_2$. (The latter random choice can be eliminated by an averaging argument; see Exercise 7.16.) Using the definition of C'' and Eq. (7.11), we have:

$$\begin{aligned}
\mathsf{Pr}[C''(Z)=F_2(Z)] &\geq \sum_{z\in G} \mathsf{Pr}[Z=z] \cdot \mathsf{Pr}[C''(z)=F_2(z)] \\
&= \sum_{z\in G} \mathsf{Pr}[Z=z] \cdot \frac{|\{(y,v) \in S_z : C(y,z)_2=F_2(z)\}|}{|S_z|} \\
&\geq \sum_{z\in G} \mathsf{Pr}[Z=z] \cdot \left(\mathsf{Pr}[C(Y,z)_2=F_2(z) \mid C(Y,z)_1=F_1(Y)] - \frac{\varepsilon}{2}\right) \\
&= \sum_{z\in G} \mathsf{Pr}[Z=z] \cdot \left(\frac{\mathsf{Pr}[C(Y,z)_2=F_2(z) \wedge C(Y,z)_1=F_1(Y)]}{\mathsf{Pr}[C(Y,z)_1=F_1(Y)]} - \frac{\varepsilon}{2}\right)
\end{aligned}$$

Next, using Claim 7.15.1, we have:

$$\begin{aligned}
\mathsf{Pr}[C''(Z)=F_2(Z)] &\geq \left(\sum_{z\in G} \mathsf{Pr}[Z=z] \cdot \frac{\mathsf{Pr}[C(Y,z)=F(Y,z)]}{\rho_1(\ell)}\right) - \frac{\varepsilon}{2} \\
&= \frac{\mathsf{Pr}[C(Y,Z)=F(Y,Z) \wedge Z\in G]}{\rho_1(\ell)} - \frac{\varepsilon}{2}
\end{aligned}$$

Finally, using Eq. (7.10), the claim follows. □

This completes the proof of the lemma. ■

Comments. Firstly, we wish to call attention to the care with which an inductive argument needs to be carried out in the computational setting, especially when a non-constant number of inductive steps is concerned. Indeed, our inductive proof of Theorem 7.14 involves invoking a quantitative lemma (i.e., Lemma 7.15) that allows to keep track of the relevant quantities (e.g., success probability and circuit size) throughout the induction process. Secondly, we mention that Lemma 7.15 (as well as Theorem 7.14) has a uniform complexity version that assumes that one can efficiently sample the distribution $(Y_{\ell(n)}, F_1(Y_{\ell(n)}))$ (resp., $(U_n, f(U_n))$). For details see [101]. Indeed, a good lesson from the proof of Lemma 7.15 is that non-uniform circuits can "effectively sample" any distribution. Lastly, we mention that Theorem 7.5 (the amplification of one-way functions) and Theorem 7.13 (Yao's XOR Lemma) also have (tight) quantitative versions (see, e.g., [90, Sec. 2.3.2] and [101, Sec. 3], respectively).

7.2.1.3 List decoding and hardness amplification

Recall that Theorem 7.10 was proved in §7.2.1.1-7.2.1.2, by first constructing a mildly inapproximable predicate via Construction 7.11, and then amplifying its hardness via Yao's XOR Lemma. In this subsection we show that the construction used in the first step (i.e., Construction 7.11) actually yields a strongly inapproximable predicate. Thus, we provide an alternative proof of Theorem 7.10. Specifically, we show that a strongly inapproximable predicate (as asserted in Theorem 7.10) can be obtained by combining Construction 7.11 (with a suitable choice of parameters) and the inner-product construction (of Theorem 7.8). The main ingredient of this argument is captured by the following result.

Proposition 7.16 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S, and let $\varepsilon : \mathbb{N} \to [0,1]$ satisfying $\varepsilon(n) > 2^{-n}$. Let f_n be the restriction of f to $\{0,1\}^n$, and let $\hat{f}_n$ be the function obtained from f_n when applying Construction 7.11*[17] *with $|H| = n/\varepsilon(n)$ and $|F| = |H|^3$. Then, the function $\hat{f} : \{0,1\}^* \to \{0,1\}^*$, defined by $\hat{f}(x) = \hat{f}_{|x|/3}(x)$, is computable in exponential-time and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = \mathrm{poly}(\varepsilon(n'/3)/n') \cdot S(n'/3)$ it holds that $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < \varepsilon'(n') \stackrel{\mathrm{def}}{=} \varepsilon(n'/3)$.*

Before turning to the proof of Proposition 7.16, let us describe how it yields an alternative proof of Theorem 7.10. Firstly, for some $\gamma > 0$, Proposition 7.16 yields an exponential-time computable function $\hat{f}$ such that $|\hat{f}(x)| \leq |x|$ and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = S(n'/3)^{\gamma}/\mathrm{poly}(n')$ it holds that $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < 1/S'(n')$. Combining this with Theorem 7.8 (cf. Exercise 7.14), we infer that $P(x,r) = b(\hat{f}(x), r)$, where $|r| = |\hat{f}(x)| \leq |x|$, is S''-inapproximable for $S''(n'') = S'(n''/2)^{\Omega(1)}/\mathrm{poly}(n'')$. In particular, for every polynomial p, we obtain a p-inapproximable predicate in $\mathcal{E}$ by applying the foregoing with $S(n) = \mathrm{poly}(n, p(n))$. Thus, Theorem 7.10 follows.

> **Teaching note:** The following material is very advanced and is best left for independent reading. Furthermore, its understanding requires being comfortable with basic notions of error-correcting codes (as presented in Appendix E.1).

Proposition 7.16 is proven by observing that the transformation of f_n to $\hat{f}_n$ constitutes a "good" code (see §E.1.1.4) and that any such code provides a worst-case to (strongly) average-case reduction. We start by defining the class of codes that suffices for the latter reduction, while noting that the code underlying the mapping $f_n \mapsto \hat{f}_n$ is actually stronger than needed.

Definition 7.17 (efficient codes supporting implicit decoding): *For fixed functions $q, \ell : \mathbb{N} \to \mathbb{N}$ and $\alpha : \mathbb{N} \to [0,1]$, the mapping $\Gamma : \{0,1\}^* \to \{0,1\}^*$ is said to be* efficient and supports implicit decoding *with parameters q, ℓ, α if it satisfies the following two conditions:*

[17] Recall that in Construction 7.11 we have $|H|^m = 2^n$, which may yield a non-integer m if we insist on $|H| = n/\varepsilon(n)$. This problem was avoided in the proof of Theorem 7.12 (where $|H| = n$ was used), but is more acute in the current context because of ε (e.g., we may have $\varepsilon(n) = 2^{-2n/7}$). Thus, we should either relax the requirement $|H|^m = 2^n$ (e.g., allow $2^n \leq |H|^m < 2^{2n}$) or relax the requirement $|H| = n/\varepsilon(n)$. However, for the sake of simplicity, we ignore this issue in the presentation.

1. Encoding (or efficiency): *The mapping Γ is polynomial-time computable.*

   It is instructive to view Γ as mapping N-bit long strings to sequences of length $\ell(N)$ over $[q(N)]$, and to view each (codeword) $\Gamma(x) \in [q(|x|)]^{\ell(|x|)}$ *as a mapping from $[\ell(|x|)]$ to $[q(|x|)]$.*

2. Decoding (in implicit form): *There exists a polynomial p such that the following holds. For every $w : [\ell(N)] \to [q(N)]$ and every $x \in \{0,1\}^N$ such that $\Gamma(x)$ is $(1-\alpha(N))$-close to w, there exists an oracle-aided*[18] *circuit C of size $p((\log N)/\alpha(N))$ such that, for every $i \in [N]$, it holds that $C^w(i)$ equals the i^{th} bit of x.*

The encoding condition implies that ℓ is polynomially bounded. The decoding condition refers to any Γ-codeword that agrees with the oracle $w : [\ell(N)] \to [q(N)]$ on an $\alpha(N)$ fraction of the $\ell(N)$ coordinates, *where $\alpha(N)$ may be very small.* We highlight the non-triviality of the decoding condition: There are N bits of information in x, while the size of the circuit C is only $p((\log N)/\alpha(N))$ and yet C should be able to recover any desired entry of x by making queries to w, which may be a highly corrupted version of $\Gamma(x)$. Needless to say, the number of queries made by C is upper-bounded by its size (i.e.,$p((\log N)/\alpha(N))$). On the other hand, the decoding condition does not refer to the complexity of obtaining the aforementioned oracle-aided circuits.

Let us relate the transformation of f_n to $\hat{f}_n$, which underlies Proposition 7.16, to Definition 7.17. We view f_n as a binary string of length $N = 2^n$ (representing the truth-table of $f_n : H^m \to \{0,1\}$) and analogously view $\hat{f}_n : F^m \to F$ as an element of $F^{|F|^m} = F^{N^3}$ (or as a mapping from $[N^3]$ to $[|F|]$).[19] Recall that the transformation of f_n to $\hat{f}_n$ is efficient. We mention that *this transformation also supports implicit decoding with parameters q, ℓ, α such that $\ell(N) = N^3$, $\alpha(N) = \varepsilon(n)$, and $q(N) = (n/\varepsilon(n))^3$*, where $N = 2^n$. The latter fact is highly non-trivial, but establishing it is beyond the scope of the current text (and the interested reader is referred to [217]).

We mention that the transformation of f_n to $\hat{f}_n$ enjoys additional features, which are not required in Definition 7.17 and will not be used in the current context. For example, there are at most $O(1/\alpha(2^n)^2)$ codewords (i.e., $\hat{f}_n$'s) that are $(1-\alpha(2^n))$-close to any fixed $w : [\ell(2^n)] \to [q(2^n)]$, and the corresponding oracle-aided circuits can be constructed in probabilistic $p(n/\alpha(2^n))$-time.[20] These results are

[18] Oracle-aided circuits are defined analogously to oracle Turing machines. Alternatively, we may consider here oracle machines that take advice such that both the advice length and the machine's running time are upper-bounded by $p((\log N)/\alpha(N))$. The relevant oracles may be viewed either as blocks of binary strings that encode sequences over $[q(N)]$ or as sequences over $[q(N)]$. Indeed, in the latter case we consider non-binary oracles, which return elements in $[q(N)]$.

[19] Recall that $N = 2^n = |H|^m$ and $|F| = |H|^3$. Hence, $|F|^m = N^3$.

[20] The construction may yield also oracle-aided circuits that compute the decoding of codewords that are almost $(1-\alpha(2^n))$-close to w. That is, there exists a probabilistic $p(n/\alpha(2^n))$-time algorithm that outputs a list of circuits that, with high probability, contains an oracle-aided circuit for the decoding of each codeword that is $(1-\alpha(2^n))$-close to w. Furthermore, with high probability, the list contains only circuits that decode codewords that are $(1-\alpha(2^n)/2)$-close to w.

termed "list decoding with implicit representations" (and we refer the interested reader again to [217]).

Our focus is on showing that efficient codes that supports implicit decoding suffice for worst-case to (strongly) average-case reductions. We state and prove a general result, noting that in the special case of Proposition 7.16 $g_n = \hat{f}_n$ (and $\ell(2^n) = 2^{3n}$).

Theorem 7.18 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S, and let $\varepsilon : \mathbb{N} \to [0,1]$. Consider a polynomial $\ell : \mathbb{N} \to \mathbb{N}$ such that $n \mapsto \log_2 \ell(2^n)$ is a 1-1 map of the integers, and let $m(n) = \log_2 \ell(2^n)$; e.g., if $\ell(N) = N^3$ then $m(n) = 3n$. Suppose that the mapping $\Gamma : \{0,1\}^* \to \{0,1\}^*$ is efficient and supports implicit decoding with parameters q, ℓ, α such that $\alpha(N) = \varepsilon(\lfloor \log_2 N \rfloor)$. Define $g_n : [\ell(2^n)] \to [q(2^n)]$ such that $g_n(i)$ equals the i^{th} element of $\Gamma(\langle f_n \rangle) \in [q(2^n)]^{\ell(2^n)}$, where $\langle f_n \rangle$ denotes the 2^n-bit long description of the truth-table of f_n. Then, the function $g : \{0,1\}^* \to \{0,1\}^*$, defined by $g(z) = g_{m^{-1}(|z|)}(z)$, is computable in exponential-time and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = \text{poly}(\varepsilon(m^{-1}(n'))/n') \cdot S(m^{-1}(n'))$ it holds that $\mathsf{Pr}[C'_{n'}(U_{n'}) = g(U_{n'})] < \varepsilon'(n') \stackrel{\text{def}}{=} \varepsilon(m^{-1}(n'))$.*

Proof Sketch: First note that we can generate the truth-table of f_n in exponential-time, and by the encoding condition of Γ it follows that g_n can be evaluated in exponential-time. The average-case hardness of g is established via a reducibility argument as follows. We consider a circuit $C' = C'_{n'}$ of size S' such that $\mathsf{Pr}[C'_{n'}(U_{n'}) = g(U_{n'})] < \varepsilon'(n')$, let $n = m^{-1}(n')$, and recall that $\varepsilon'(n') = \varepsilon(n) = \alpha(2^n)$. Then, $C' : \{0,1\}^{n'} \to \{0,1\}$ (viewed as a function) is $(1 - \alpha(2^n))$-close to the function g_n, which in turn equals $\Gamma(\langle f_n \rangle)$. The decoding condition of Γ asserts that we can recover each bit of $\langle f_n \rangle$ (i.e., evaluate f_n) by an oracle-aided circuit D of size $p(n/\alpha(2^n))$ that uses (the function) C' as an oracle. Combining (the circuit C') with the oracle-aided circuit D, we obtain a (standard) circuit of size $p(n/\alpha(2^n)) \cdot S'(n') < S(n)$ that computes f_n. The theorem follows (i.e., the violation of the conclusion regarding g implies the violation of the hypothesis regarding f). $\square$

Advanced comment. For simplicity, we formulated Definition 7.17 in a crude manner that suffices for the proving Proposition 7.16, where $q(N) = ((\log_2 N)/\alpha(N))^3$. The issue is the existence of codes that satisfy Definition 7.17: In general, such codes may exist only when using a more careful formulation of the decoding condition that refers to codewords that are $(1 - ((1/q(N)) + \alpha(N)))$-close to the oracle $w : [\ell(N)] \to [q(N)]$ rather than being $(1 - \alpha(N))$-close to it.[21] Needless to say, the difference is insignificant in the case that $\alpha(N) \gg 1/q(N)$ (as in Proposition 7.16),

[21] Note that this is the "right" formulation, because in the case that $\alpha(N) < 1/q(N)$ it seems impossible to satisfy the decoding condition (as stated in Definition 7.17). Specifically, a random $\ell(N)$-sequence over $[q(N)]$ is expected to be $(1 - (1/q(N)))$-close to any fixed codeword, and with overwhelmingly high probability it will be $(1 - ((1 - o(1))/q(N)))$-close to almost all the codewords, provided $\ell(N) \gg q(N)^2$. But in case $N > \text{poly}(q(N))$, we cannot hope to recover almost all N-bit long strings based on $\text{poly}(q(N) \log N)$ bits of advice (per each of them).

but it is significant in case we care about binary codes (i.e., $q(N) = 2$, or codes over other small alphabets). We mention that Theorem 7.18 can be adapted to this context (of $q(N) = 2$), and directly yields strongly inapproximable predicates. For details, see Exercise 7.21.

7.2.2 Amplification wrt exponential-size circuits

For the purpose of stronger derandomization of $\mathcal{BPP}$, we start with a stronger assumption regarding the worst-case circuit complexity of $\mathcal{E}$ and turn it to a stronger inapproximability result.

Theorem 7.19 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having almost-everywhere exponential circuit complexity; that is, there exists a constant $b > 0$ such that, for all but finitely many n's, any circuit that correctly computes f on $\{0,1\}^n$ has size at least $2^{b \cdot n}$. Then, for some constant $c > 0$ and $T(n) \stackrel{\text{def}}{=} 2^{c \cdot n}$, there exists a T-inapproximable Boolean function in $\mathcal{E}$.*

Theorem 7.19 can be used for deriving a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$) under the aforementioned assumption (see Part 1 of Theorem 8.19).

Theorem 7.19 follows as a special case of Proposition 7.16 (combined with Theorem 7.8; see Exercise 7.22). An alternative proof, which uses different ideas that are of independent interest, will be briefly reviewed next. The starting point of the latter proof is a mildly inapproximable predicate, as provided by Theorem 7.12. However, here we cannot afford to apply Yao's XOR Lemma (i.e., Theorem 7.13), because the latter relates the size of circuits that *strongly* fail to approximate a predicate defined over $\text{poly}(n)$-bit long strings to the size of circuits that fail to *mildly* approximate a predicate defined over n-bit long strings. That is, Yao's XOR Lemma asserts that if $f : \{0,1\}^n \to \{0,1\}$ is mildly inapproximable by S_f-size circuits then $F : \{0,1\}^{\text{poly}(n)} \to \{0,1\}$ is strongly inapproximable by S_F-size circuits, where $S_F(\text{poly}(n))$ is polynomially related to $S_f(n)$. In particular, $S_F(\text{poly}(n)) < S_f(n)$ seems inherent in this reasoning. For the case of polynomial lower-bounds, this is good enough (i.e., if S_f can be an arbitrarily large polynomial then so can S_F), but for $S_f(n) = \exp(\Omega(n))$ we cannot obtain $S_F(m) = \exp(\Omega(m))$ (but rather only obtain $S_F(m) = \exp(m^{\Omega(1)})$).

The source of trouble is that amplification of inapproximability was achieved by taking a polynomial number of independent instances. Indeed, we cannot hope to amplify hardness without applying f on many instances, but these instances need not be independent. Thus, the idea is to define $F(r) = \oplus_{i=1}^{\text{poly}(n)} f(x_i)$, where $x_1, ..., x_{\text{poly}(n)} \in \{0,1\}^n$ are generated from r and still $|r| = O(n)$. That is, we seek a "derandomized" version of Yao's XOR Lemma. In other words, we seek a "pseudorandom generator" of a type appropriate for expanding r to dependent x_i's such that the XOR of the $f(x_i)$'s is as inapproximable as it would have been for independent x_i's.[22]

[22] Indeed, this falls within the general paradigm discussed in Section 8.1. Furthermore, this suggestion provides another perspective on the connection between randomness and computational difficulty, which is the focus of much discussion in Chapter 8 (see, e.g., §8.2.7.2).

Teaching note: In continuation to Footnote 22, we note that there is a strong connection between the rest of this section and Chapter 8. On top of the aforementioned conceptual aspect, we will use technical tools from Chapter 8 towards establishing the derandomized version of the XOR Lemma. These tools include pairwise independence generators (see Section 8.5.1), random walks on expanders (see Section 8.5.3), and the Nisan-Wigderson Construction (Construction 8.17). Indeed, recall that Section 7.2.2 is advanced material, which is best left for independent reading.

The pivot of the proof is the notion of a hard region of a Boolean function. Loosely speaking, S is a hard region of a Boolean function f if f is *strongly inapproximable on a random input in* S; that is, for every (relatively) small circuit C_n, it holds that $\Pr[C_n(U_n) = f(U_n)|U_n \in S] \approx 1/2$. By definition, $\{0,1\}^*$ is a hard region of any *strongly* inapproximable predicate. As we shall see, any *mildly* inapproximable predicate has a hard region of density related to its inapproximability parameter. Loosely speaking, hardness amplification will proceed via methods for generating related instances that hit the hard region with sufficiently high probability. But, first let us study the notion of a hard region.

7.2.2.1 Hard regions

We actually generalize the notion of hard regions to arbitrary distributions. The important special case of uniform distributions (on n-bit long strings) is obtained from Definition 7.20 by letting X_n equal U_n (i.e., the uniform distribution over $\{0,1\}^n$). In general, we only assume that $X_n \in \{0,1\}^n$.

Definition 7.20 (hard region relative to arbitrary distribution): *Let* $f:\{0,1\}^* \to \{0,1\}$ *be a Boolean predicate,* $\{X_n\}_{n\in\mathbb{N}}$ *be a probability ensemble,* $s:\mathbb{N}\to\mathbb{N}$ *and* $\varepsilon:\mathbb{N}\to[0,1]$.

- *We say that a set* S *is a* hard region of f relative to $\{X_n\}_{n\in\mathbb{N}}$ *with respect to* $s(\cdot)$*-size circuits and advantage* $\varepsilon(\cdot)$ *if for every* n *and every circuit* C_n *of size at most* $s(n)$*, it holds that*

$$\Pr[C_n(X_n)=f(X_n)|X_n\in S] \le \frac{1}{2}+\varepsilon(n).$$

- *We say that* f *has a* hard region of density $\rho(\cdot)$ *relative to* $\{X_n\}_{n\in\mathbb{N}}$ (with respect to $s(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$) *if there exists a set* S *that is a hard region of* f *relative to* $\{X_n\}_{n\in\mathbb{N}}$ (with respect to the foregoing parameters) *such that* $\Pr[X_n\in S_n] \ge \rho(n)$.

Note that a Boolean function f is $(s, 1-2\varepsilon)$-inapproximable if and only if $\{0,1\}^*$ is a hard region of f relative to $\{U_n\}_{n\in\mathbb{N}}$ with respect to $s(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$. Thus, *strongly* inapproximable predicates (e.g., S-inapproximable predicates for super-polynomial S) have a hard region of density 1 (with respect to a negligible advantage).[23] But this trivial observation does not provide hard regions

[23]Likewise, *mildly* inapproximable predicates have a hard region of density 1 with respect to an advantage that is noticeably smaller than 1/2.

(with respect to a small (i.e., close to zero) advantage) for *mildly* inapproximable predicates. Providing such hard regions is the contents of the following theorem.

Theorem 7.21 (hard regions for mildly inapproximable predicates): *Let $f:\{0,1\}^* \to \{0,1\}$ be a Boolean predicate, $\{X_n\}_{n\in\mathbb{N}}$ be a probability ensemble, $s:\mathbb{N}\to\mathbb{N}$, and $\rho:\mathbb{N}\to[0,1]$ such that $\rho(n) > 1/\mathrm{poly}(n)$. Suppose that, for every circuit C_n of size at most $s(n)$, it holds that $\mathsf{Pr}[C_n(X_n)=f(X_n)] \leq 1-\rho(n)$. Then, for every $\varepsilon:\mathbb{N}\to[0,1]$, the function f has a hard region of density $\rho'(\cdot)$ relative to $\{X_n\}_{n\in\mathbb{N}}$ with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$, where $\rho'(n) \stackrel{\mathrm{def}}{=} (1-o(1))\cdot\rho(n)$ and $s'(n) \stackrel{\mathrm{def}}{=} s(n)/\mathrm{poly}(n/\varepsilon(n))$.*

In particular, if f is $(s, 2\rho)$-inapproximable then f has a hard region of density $\rho'(\cdot) \approx \rho(\cdot)$ relative to the uniform distribution (with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$).

Proof Sketch:[24] The proof proceeds by first establishing that $\{X_n\}$ is "related" to (or rather "dominates") an ensemble $\{Y_n\}$ such that f is strongly inapproximable on $\{Y_n\}$, and next showing that this implies the claimed hard region. Indeed, this notion of "related ensembles" plays a central role in the proof.

For $\rho:\mathbb{N}\to[0,1]$, we say that $\{X_n\}$ ρ-dominates $\{Y_n\}$ if for every x it holds that $\mathsf{Pr}[X_n=x] \geq \rho(n)\cdot\mathsf{Pr}[Y_n=x]$. In this case we also say that $\{Y_n\}$ is ρ-dominated by $\{X_n\}$. We say that $\{Y_n\}$ is critically ρ-dominated by $\{X_n\}$ if for every x either $\mathsf{Pr}[Y_n=x] = (1/\rho(n))\cdot\mathsf{Pr}[X_n=x]$ or $\mathsf{Pr}[Y_n=x] = 0$.[25]

The notions of domination and critical domination play a central role in the proof, which consists of two parts. In the first part (Claim 7.21.1), we prove that, for $\{X_n\}$ and ρ as in the theorem's hypothesis, there exists a ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Y_n\}$. In the second part (Claim 7.21.2), we prove that the existence of such a dominated ensemble implies the existence of an ensemble $\{Z_n\}$ that is *critically* ρ'-dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Z_n\}$. Finally, we note that such a critically dominated ensemble yields a hard region of f relative to $\{X_n\}$, and the theorem follows.

Claim 7.21.1: Under the hypothesis of the theorem it holds that there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that, for every $s'(n)$-size circuit C_n, it holds that

$$\mathsf{Pr}[C_n(Y_n)=f(Y_n)] \leq \frac{1}{2} + \frac{\varepsilon(n)}{2}. \tag{7.12}$$

Proof: We start by assuming, towards the contradiction, that for every distribution Y_n that is ρ-dominated by X_n there exists a $s'(n)$-size circuits C_n such that $\mathsf{Pr}[C_n(Y_n) = f(Y_n)] > 0.5 + \varepsilon'(n)$, where $\varepsilon'(n) = \varepsilon(n)/2$. One key observation is that there is a correspondence between the set of all distributions that are

[24] See details in [101, Apdx. A].

[25] Actually, we should allow one point of exception; that is, relax the requirement by saying that for at most one string $x \in \{0,1\}^n$ it holds that $0 < \mathsf{Pr}[Y_n=x] < \mathsf{Pr}[X_n=x]/\rho(n)$. This point has little effect on the proof, and is ignored in our presentation.

each ρ-dominated by X_n and the set of all the convex combinations of critically ρ-dominated (by X_n) distributions; that is, each ρ-dominated distribution is a convex combinations of critically ρ-dominated distributions and vice versa (cf., a special case in §D.4.1.1). Thus, considering an enumeration $Y_n^{(1)}, ..., Y_n^{(t)}$ of the critically ρ-dominated (by X_n) distributions, we conclude that for every distribution π on $[t]$ there exists a $s'(n)$-size circuits C_n such that

$$\sum_{i=1}^{t} \pi(i) \cdot \Pr[C_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n). \tag{7.13}$$

Now, consider a finite game between two players, where the first player selects a critically ρ-dominated (by X_n) distribution, and the second player selects a $s'(n)$-size circuit and obtains a payoff as determined by the corresponding success probability; that is, if the first player selects the i^{th} critically dominated distribution and the second player selects the circuit C then the payoff equals $\Pr[C(Y_n^{(i)}) = f(Y_n^{(i)})]$. Eq. (7.13) may be interpreted as saying that for any randomized strategy for the first player there exists a deterministic strategy for the second player yielding average payoff greater than $0.5+\varepsilon'(n)$. The Min-Max Principle (cf. von Neumann [233]) asserts that in such a case there exists a randomized strategy for the second player that yields average payoff greater than $0.5 + \varepsilon'(n)$ no matter what strategy is employed by the first player. This means that there exists a distribution, denoted D_n, on $s'(n)$-size circuits such that for every i it holds that

$$\Pr[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n), \tag{7.14}$$

where the probability refers both to the choice of the circuit D_n and to the random variable Y_n. Let $B_n = \{x : \Pr[D_n(x) = f(x)] \leq 0.5 + \varepsilon'(n)\}$. Then, $\Pr[X_n \in B_n] < \rho(n)$, because otherwise we reach a contradiction to Eq. (7.14) by defining Y_n such that $\Pr[Y_n = x] = \Pr[X_n = x]/\Pr[X_n \in B_n]$ if $x \in B_n$ and $\Pr[Y_n = x] = 0$ otherwise.[26] By employing standard amplification to D_n, we obtain a distribution D'_n over $\text{poly}(n/\varepsilon'(n)) \cdot s'(n)$-size circuits such that for every $x \in \{0,1\}^n \setminus B_n$ it holds that $\Pr[D'_n(x) = f(x)] > 1 - 2^{-n}$. It follows that there exists a $s(n)$-sized circuit C_n such that $C_n(x) = f(x)$ for every $x \in \{0,1\}^n \setminus B_n$, which implies that $\Pr[C_n(X_n) = f(X_n)] \geq \Pr[X_n \in \{0,1\}^n \setminus B_n] > 1 - \rho(n)$, in contradiction to the theorem's hypothesis. The claim follows. □

We next show that the conclusion of Claim 7.21.1 (which was stated for ensembles that are ρ-dominated by $\{X_n\}$) essentially holds also when allowing only critically ρ-dominated (by $\{X_n\}$) ensembles. The following precise statement involves some loss in the domination parameter ρ (as well as in the advantage ε).

Claim 7.21.2: If there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that for every $s'(n)$-size circuit C_n it holds that $\Pr[C_n(Y_n) =$

[26] Note that Y_n is ρ-dominated by X_n, whereas by the hypothesis $\Pr[D_n(Y_n) = f(Y_n)] \leq 0.5 + \varepsilon'(n)$. Using the fact that any ρ-dominated distribution is a convex combination of critically ρ-dominated distributions, it follows that $\Pr[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] \leq 0.5 + \varepsilon'(n)$ holds for some critically ρ-dominated $Y_n^{(i)}$.

$f(Y_n)] \leq 0.5 + (\varepsilon(n)/2)$, then there exists a probability ensemble $\{Z_n\}$ that is critically ρ'-dominated by $\{X_n\}$ such that for every $s'(n)$-size circuit C_n it holds that $\Pr[C_n(Z_n) = f(Z_n)] \leq 0.5 + \varepsilon(n)$.

In other words, Claim 7.21.2 asserts that the function f has a hard region of density $\rho'(\cdot)$ relative to $\{X_n\}$ with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$, thus establishing the theorem. The proof of Claim 7.21.2 uses the Probabilistic Method (cf. [10]). Specifically, we select a set S_n at random by including each n-bit long string x with probability

$$p(x) \stackrel{\text{def}}{=} \frac{\rho(n) \cdot \Pr[Y_n = x]}{\Pr[X_n = x]} \leq 1 \tag{7.15}$$

independently of the choice of all other strings. It can be shown that, with high probability over the choice of S_n, it holds that $\Pr[X_n \in S_n] \approx \rho(n)$ and that $\Pr[C_n(X_n) = f(X_n) | X_n \in S_n] < 0.5 + \varepsilon(n)$ for every circuit C_n of size $s'(n)$. The latter assertion is proved by a union bound on all relevant circuits, while showing that for each such circuit C_n, with probability $1 - \exp(-s'(n)^2)$ over the choice of S_n, it holds that $|\Pr[C_n(X_n) = f(X_n)|X_n \in S_n] - \Pr[C_n(Y_n) = f(Y_n)]| < \varepsilon(n)/2$. For details, see [101, Apdx. A]. (This completes the proof of the theorem.) □

7.2.2.2 Hardness amplification via hard regions

Before showing how to use the notion of a hard region in order to prove a derandomized version of Yao's XOR Lemma, we show how to use it in order to prove the original version of Yao's XOR Lemma (i.e., Theorem 7.13).

An alternative proof of Yao's XOR Lemma. Let f, p_1, and p_2 be as in Theorem 7.13. Then, by Theorem 7.21, for $\rho'(n) = 1/3p_2(n)$ and $s'(n) = p_1(n)^{\Omega(1)}/\text{poly}(n)$, the function f has a hard region S of density ρ' (relative to $\{U_n\}$) with respect to $s'(\cdot)$-size circuits and advantage $1/s'(\cdot)$. Thus, for $t(n) = n \cdot p_2(n)$ and F as in Theorem 7.13, with probability at least $1 - (1 - \rho'(n))^{t(n)} = 1 - \exp(-\Omega(n))$, one of the $t(n)$ random (n-bit long) blocks of F resides in S (i.e., the hard region of f). Intuitively, this suffices for establishing the strong inapproximability of F. Indeed, suppose towards the contradiction that a small (i.e., $p'(t(n) \cdot n)$-size) circuit C_n can approximate F (over $U_{t(n)\cdot n}$) with advantage $\varepsilon(n) + \exp(-\Omega(n))$, where $\varepsilon(n) > 1/s'(n)$. Then, the $\varepsilon(n)$ term must be due to $t(n) \cdot n$-bit long inputs that contain a block in S. Using an averaging argument, we can first fix the index of this block and then the contents of the other blocks, and infer the following: for some $i \in [t(n)]$ and $x_1, ..., x_{t(n)} \in \{0,1\}^n$ it holds that

$$\Pr[C_n(x', U_n, x'') = F(x', U_n, x'') \,|\, U_n \in S] \;\geq\; \frac{1}{2} + \varepsilon(n)$$

where $x' = (x_1, ..., x_{i-1}) \in \{0,1\}^{(i-1)\cdot n}$ and $x'' = (x_{i+1}, ..., x_{t(n)}) \in \{0,1\}^{(t(n)-i)\cdot n}$. Hard-wiring $i \in [t(n)]$, $x' = (x_1, ..., x_{i-1})$ and $x'' = (x_{i+1}, ..., x_{t(n)})$ as well as $\sigma \stackrel{\text{def}}{=} \oplus_{j \neq i} f(x_j)$ in C_n, we obtain a contradiction to the (established) fact that

S is a hard region of f (by using the circuit $C'_n(z) = C_n(x', z, x'') \oplus \sigma$). Thus, Theorem 7.13 follows (for any $p'(t(n) \cdot n) \le s'(n) - 1$).

Derandomized versions of Yao's XOR Lemma. We first show how to use the notion of a hard region in order to amplify very mild inapproximability to a constant level of inapproximability. Recall that our goal is to obtain such an amplification while applying the given function on many (related) instances, where each instance has length that is linearly related to the length of the input of the resulting function. Indeed, these related instances are produced by applying an adequate "pseudorandom generator" (see Chapter 8). The following amplification utilizes a pairwise independence generator (see Section 8.5.1), denoted G, that stretches $2n$-bit long seeds to sequences of n strings, each of length n.

Lemma 7.22 (derandomized XOR Lemma up to constant inapproximability): *Suppose that $f : \{0,1\}^* \to \{0,1\}$ is (T, ρ)-inapproximable, for $\rho(n) > 1/\mathrm{poly}(n)$, and assume for simplicity that $\rho(n) \le 1/n$. Let b denote the inner-product mod 2 predicate, and G be the aforementioned pairwise independence generator. Then $F_1(s, r) = b(f(x_1) \cdots f(x_n), r)$, where $|r| = n = |s|/2$ and $(x_1, ..., x_n) = G(s)$, is (T', ρ')-inapproximable for $T'(n') = T(n'/3)/\mathrm{poly}(n')$ and $\rho'(n') = \Omega(n' \cdot \rho(n'/3))$.*

Needless to say, if $f \in \mathcal{E}$ then $F_1 \in \mathcal{E}$. By applying Lemma 7.22 for a constant number of times, we may transform an $(T, 1/\mathrm{poly})$-inapproximable predicate into an $(T'', \Omega(1))$-inapproximable one, where $T''(n'') = T(n''/O(1))/\mathrm{poly}(n'')$.

Proof Sketch: As in the foregoing proof (of the original version of Yao's XOR Lemma), we first apply Theorem 7.21. This time we set the parameters so to infer that, for $\alpha(n) = \rho(n)/3$ and $t'(n) = T(n)/\mathrm{poly}(n)$, the function f has a hard region S of density α (relative to $\{U_n\}$) with respect to $t'(\cdot)$-size circuits and *advantage 0.01*. Next, as in §7.2.1.2, we shall consider the corresponding (derandomized) direct product problem; that is, the function $P_1(s) = (f(x_1), ..., f(x_n))$, where $|s| = 2n$ and $(x_1, ..., x_n) = G(s)$. We will first show that P_1 is hard to compute on an $\Omega(n \cdot \alpha(n))$ fraction of the domain, and the quantified inapproximality of F_1 will follow.

One key observation is that, by Exercise 7.23, with probability at least $\beta(n) \stackrel{\text{def}}{=} n \cdot \alpha(n)/2$, at least one of the n strings output by $G(U_{2n})$ resides in S. Intuitively, we expect every $t'(n)$-sized circuit to fail in computing $P_1(U_{2n})$ with probability at least $0.49\beta(n)$, because with probability $\beta(n)$ the sequence $G(U_{2n})$ contains an element in the hard region of f (and in this case the value can be guessed correctly with probability at most 0.51). The actual proof relies on a reducibility argument, which is less straightforward than the argument used in the non-derandomized case.

For technical reasons[27], we use the condition $\alpha(n) < 1/2n$ (which is guaranteed by the hypothesis that $\rho(n) \le 1/n$ and our setting of $\alpha(n) = \rho(n)/3$). In this case Exercise 7.23 implies that, with probability at least $\beta(n) \stackrel{\text{def}}{=} 0.75 \cdot n \cdot \alpha(n)$, at least one of the n strings output by $G(U_{2n})$ resides in S. We shall show that

[27] The following argument will rely on the fact that $\beta(n) - \gamma(n) > 0.51n \cdot \alpha(n)$, where $\gamma(n) = \Omega(\beta(n))$.

every $(t'(n) - \text{poly}(n))$-sized circuit fails in computing P_1 with probability at least $\gamma(n) = 0.3\beta(n)$. As usual, the claim is proved by a reducibility argument. Let $G(s)_i$ denote the i^{th} string in the sequence $G(s)$ (i.e., $G(s) = (G(s)_1, ..., G(s)_n)$), and note that given i and x we can efficiently sample $G_i^{-1}(x) \stackrel{\text{def}}{=} \{s \in \{0,1\}^{2n} : G(s)_i = x\}$. Given a circuit C_n that computes $P_1(U_{2n})$ correctly with probability $1 - \gamma(n)$, we consider the circuit C'_n that, on input x, uniformly selects $i \in [n]$ and $s \in G_i^{-1}(x)$, and outputs the i^{th} bit in $C_n(s)$. Then, by the construction (of C'_n) and the hypothesis regarding C_n, it holds that

$$\begin{aligned}
\mathsf{Pr}[C'_n(U_n)=f(U_n)|U_n\in S] &\geq \sum_{i=1}^{n} \frac{1}{n} \cdot \mathsf{Pr}[C_n(U_{2n})=P_1(U_{2n})|G(U_{2n})_i \in S] \\
&\geq \frac{\mathsf{Pr}[C_n(U_{2n})=P_1(U_{2n}) \wedge \exists i\, G_i(U_{2n})_i \in S]}{n \cdot \max_i\{\mathsf{Pr}[G(U_{2n})_i \in S]\}} \\
&\geq \frac{(1-\gamma(n)) - (1-\beta(n))}{n \cdot \alpha(n)} \\
&= \frac{0.7\beta(n)}{n \cdot \alpha(n)} \;>\; 0.52\,.
\end{aligned}$$

This contradicts the fact that S is a hard region of f with respect to $t'(\cdot)$-size circuits and advantage 0.01. Thus, we have established that every $(t'(n) - \text{poly}(n))$-sized circuit fails in computing P_1 with probability at least $\gamma(n) = 0.3\beta(n)$.

Having established the hardness of P_1, we now infer the mild inapproximability of F_1, where $F_1(s,r) = b(P_1(s),r)$. It suffices to employ the simple (warm-up) case discussed at the beginning of the proof of Theorem 7.7 (where the predictor errs with probability less than 1/4, rather than the full-fledged result that refers to prediction error that is only smaller than 1/2). Denoting by $\eta_C(s) = \mathsf{Pr}_{r\in\{0,1\}^n}[C(s,r) \neq b(P_1(s),r)]$ the prediction error of the circuit C, we recall that if $\eta_C(s) \leq 0.24$ then C can be used to recover $P_1(s)$. Thus, for circuits C of size $T'(3n) = t'(n)/\text{poly}(n)$ it must hold that $\mathsf{Pr}_s[\eta_C(s) > 0.24] \geq \gamma(n)$. It follows that $\mathsf{E}_s[\eta_C(s)] > 0.24\gamma(n)$, which means that every $T'(3n)$-sized circuits fails to compute $(s,r) \mapsto b(P_1(s),r)$ with probability at least $\delta(|s|+|r|) \stackrel{\text{def}}{=} 0.24 \cdot \gamma(|r|)$. This means that F_1 is $(T', 2\delta)$-inapproximable, and the lemma follows (when noting that $\delta(n') = \Omega(n' \cdot \alpha(n'/3))$). □

The next lemma offers an amplification of constant inapproximability to strong inapproximability. Indeed, combining Theorem 7.12 with Lemmas 7.22 and 7.23, yields Theorem 7.19 (as a special case).

Lemma 7.23 (derandomized XOR Lemma starting with constant inapproximability): *Suppose that $f : \{0,1\}^* \to \{0,1\}$ is (T,ρ)-inapproximable, for some constant ρ, and let b denote the inner-product mod 2 predicate. Then there exists an exponential-time computable function G such that $F_2(s,r) = b(f(x_1)\cdots f(x_n), r)$, where $(x_1, ..., x_n) = G(s)$ and $n = \Omega(|s|) = |r| = |x_1| = \cdots = |x_n|$, is T'-inapproximable for $T'(n') = T(n'/O(1))^{\Omega(1)}/\text{poly}(n')$.*

Again, if $f \in \mathcal{E}$ then $F_2 \in \mathcal{E}$.

Proof Outline:[28] As in the proof of Lemma 7.22, we start by establishing a hard region of density $\rho/3$ for f (this time with respect to circuits of size $T(n)^{\Omega(1)}/\text{poly}(n)$ and advantage $T(n)^{-\Omega(1)}$), and focus on the analysis of the (derandomized) direct product problem corresponding to computing the function $P_2(s) = (f(x_1), ..., f(x_n))$, where $|s| = O(n)$ and $(x_1, ..., x_n) = G(s)$. The "generator" G is defined such that $G(s's'') = G_1(s') \oplus G_2(s'')$, where $|s'| = |s''|$, $|G_1(s')| = |G_2(s'')|$, and the following conditions hold:

1. G_1 is the Expander Random Walk Generator discussed in Section 8.5.3. It can be shown that $G_1(U_{O(n)})$ outputs a sequence of n strings such that for any set S of density ρ, with probability $1 - \exp(-\Omega(\rho n))$, at least $\Omega(\rho n)$ of the strings hit S. Note that this property is inherited by G, provided $|G_1(s')| = |G_2(s'')|$ for any $|s'| = |s''|$. It follows that, with probability $1 - \exp(-\Omega(\rho n))$, a constant fraction of the x_i's in the definition of P_2 hit the hard region of f.

   It is tempting to say that small circuits cannot compute P_2 better than with probability $\exp(-\Omega(\rho n))$, but this is clear only in the case that the x_i's that hit the hard region are distributed independently (and uniformly) in it, which is hardly the case here. Indeed, G_2 is used to handle this problem.

2. G_2 is the "set projection" system underlying Construction 8.17; specifically, $G_2(s) = (s_{S_1}, ..., s_{S_n})$, where each S_i is an n-subset of $[|s|]$ and the S_i's have pairwise intersections of size at most $n/O(1)$.[29] An analysis as in the proof of Theorem 8.18 can be employed for showing that the dependency among the x_i's does not help for computing a particular $f(x_i)$ when given x_i as well as all the other $f(x_j)$'s. (Note that this property of G_2 is inherited by G.)

The actual analysis of the construction (via a guessing game presented in [127, Sec. 3]), links the success probability of computing P_2 to the advantage of guessing f on its hard region. The interested reader is referred to [127]. □

Digest. Both Lemmas 7.22 and 7.23 are proved by first establishing corresponding derandomized versions of the "direct product" lemma (Theorem 7.14); in fact, the core of these proofs is proving adequate derandomized "direct product" lemmas. We call the reader's attention to the seemingly crucial role of this step (especially in the proof of Lemma 7.23): We cannot treat the values $f(x_1), ...f(x_n)$ as if they were independent (at least not for the generator G as postulated in these lemmas), and so we seek to avoid analyzing the probability of correctly computing the XOR of *all these values*. In contrast, we have established that it is very hard to correctly compute all n values, and thus *XORing a random subset of these values* yields a strongly inapproximable predicate. (Note that the argument used in Exercise 7.17

[28] For details, see [127].

[29] Recall that s_S denotes the projection of s on coordinates $S \subseteq [|s|]$; that is, for $s = \sigma_1 \cdots \sigma_k$ and $S = \{i_j : j = 1, ..., n\}$, we have $s_S = \sigma_{i_1} \cdots \sigma_{i_n}$.

fails here, because the x_i's are not independent, which is the reason that we XOR a random subset of these values rather than all of them.)

Chapter Notes

The notion of a one-way function was suggested by Diffie and Hellman [65]. The notion of weak one-way functions as well as the amplification of one-way functions (i.e., Theorem 7.5) were suggested by Yao [237]. A proof of Theorem 7.5 has first appeared in [86].

The concept of hard-core predicates was suggested by Blum and Micali [39]. They also proved that a particular predicate constitutes a hard-core for the "DLP function" (i.e., exponentiation in a finite field), provided that the latter function is one-way. The generic hard-core predicate (of Theorem 7.7) was suggested by Levin, and proven as such by Goldreich and Levin [98]. The proof presented here was suggested by Rackoff. We comment that the original proof has its own merits (cf., e.g., [104]).

The construction of canonical derandomizers (see Section 8.3) and, specifically, the Nisan-Wigderson framework (i.e., Construction 8.17) has been the driving force behind the study of inapproximable predicates in $\mathcal{E}$. Theorem 7.10 is due to [21], whereas Theorem 7.19 is due to [127]. Both results rely heavily of variants of Yao's XOR Lemma, to be reviewed next.

Like several other fundamental insights[30] attributed to Yao's paper [237], Yao's XOR Lemma (i.e., Theorem 7.13) is not even stated in [237] but is rather due to Yao's oral presentations of his work. The first published proof of Yao's XOR Lemma was given by Levin (see [101, Sec. 3]). The proof presented in §7.2.1.2 is due to Goldreich, Nisan and Wigderson [101, Sec. 5]. For a recent (but brief) review of other proofs of Yao's XOR Lemma (as well as of variants of it), the interested reader is referred to [222].

The notion of a hard region and its applications to proving the original version of Yao's XOR Lemma are due to Impagliazzo [125] (see also [101, Sec. 4]). The first derandomization of Yao's XOR Lemma (i.e., Lemma 7.22) also originates in [125], while the second derandomization (i.e., Lemma 7.23) as well as Theorem 7.19 are due to Impagliazzo and Wigderson [127].

The worst-case to average-case reduction (i.e., §7.2.1.1, yielding Theorem 7.12) is due to [21]. This reduction follows the self-correction paradigm of Blum, Luby and Rubinfeld [40], which was first employed in the context of a (strict)[31] worst-case to average-case reduction by Lipton [156].[32]

[30] Most notably, the equivalence of pseudorandomness and unpredictability (see Section 8.2.5).

[31] Earlier uses of the self-correction paradigm referred to "two argument problems" and consisted of fixing one argument and randomizing the other (see, e.g., [107]); consider, for example, the decision problem in which given (N, r) the task is to determine whether $x^2 \equiv r \pmod N$ has an integer solution, and the randomized process mapping (N, r) to (N, r'), where $r' = r \cdot \omega^2 \bmod N$ and ω is uniformly distibuted in $[N]$. Loosely speaking, such a process yields a reduction from worst-case complexity to "mixed worst/average-case" complexity (or from "mixed average/worst-case" to pure average-case).

[32] An earlier use of the self-correction paradigm for a strict worst-case to average-case reduction

The connection between list decoding and hardness amplification (i.e., §7.2.1.3), yielding alternative proofs of Theorems 7.10 and 7.19, is due to Sudan, Trevisan, and Vadhan [217].

Hardness amplification for $\mathcal{NP}$ has been the subject of recent attention: An amplification of mild inapproximability to strong inapproximability is provided in [120], and an indication to the impossibility of a worst-case to average-case reductions (at least non-adaptive ones) is provided in [42].

Exercises

Exercise 7.1 Prove that if one way-functions exist then there exists one-way functions that are length preserving (i.e., $|f(x)| = |x|$ for every $x \in \{0,1\}^n$).

Guideline: Clearly, for some polynomial p, it holds that $|f(x)| < p(|x|)$ for all x. Assume, without loss of generality that $n \mapsto p(n)$ is 1-1 and increasing, and let $p^{-1}(m) = n$ if $p(n) \leq m < p(n+1)$. Define $f'(z) = f(x)01^{|z|-|f(x)|-1}$, where x is the $p^{-1}(|z|)$-bit long prefix of z.

Exercise 7.2 Prove that if a function f is hard to invert in the sense of Definition 7.3 then it is hard to invert in the sense of Definition 7.1.

Guideline: Consider a sequence of internal coin tosses that maximizes the probability in Eq. (7.1).

Exercise 7.3 Assuming the existence of one-way functions, prove that there exists a weak one-way function that is not strongly one-way.

Exercise 7.4 (a universal one-way function (by L. Levin)) Using the notion of a universal machine, present a polynomial-time computable function that is hard to invert (in the sense of Definition 7.1) if and only if there exist one-way functions.

Guideline: Consider the function F that parses its input into a pair (M, x) and emulates $|x|^3$ steps of M on input x. Note that if there exists a one-way function that can be evaluated in cubic time then F is a weak one-way function. Using padding, prove that there exists a one-way function that can be evaluated in cubic time if and only if there exist one-way functions.

Exercise 7.5 For $\ell > 1$, prove that the following $2^\ell - 1$ samples are pairwise independent and uniformly distributed in $\{0,1\}^n$. The samples are generated by uniformly and independently selecting ℓ strings in $\{0,1\}^n$. Denoting these strings by $s^1, ..., s^\ell$, we generate $2^\ell - 1$ samples corresponding to the different *non-empty* subsets of $\{1, 2, ..., \ell\}$ such that for subset J we let $r^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$.

appears in [18], but it refers to very low complexity classes. Specifically, this reduction refers to the `parity` function and is computable in $\mathcal{AC}^0$ (implying that `parity` cannot be approximated in $\mathcal{AC}^0$, since it cannot be computed in that class (see [82, 238, 114])). The reduction (randomly) maps $x \in \{0,1\}^n$, viewed as a sequence $(x_1, x_2, x_3, ..., x_n)$, to the sequence $x' = (x_1 \oplus r_1, r_1 \oplus x_2 \oplus r_2, r_2 \oplus x_3 \oplus r_3, ..., r_{n-1} \oplus x_n \oplus r_n)$, where $r_1, ..., r_n \in \{0,1\}$ are uniformly and independently distributed. Note that x' is uniformly distributed in $\{0,1\}^n$ and that $\texttt{parity}(x) = \texttt{parity}(x') \oplus r_n$.

Guideline: For $J \neq J'$, it holds that $r^J \oplus r^{J'} = \oplus_{j \in K} s^j$, where K denotes the symmetric difference of J and J'. See related material in Section 8.5.1.

Exercise 7.6 (a variant on the proof of Theorem 7.7) Provide a detailed presentation of the alternative procedure outlined in Footnote 5. That is, prove that for every $x \in \{0,1\}^n$, given oracle access to any $B_x : \{0,1\}^n \to \{0,1\}$ that satisfies Eq. (7.6), this procedure makes $\text{poly}(n/\varepsilon)$ steps and outputs a list of strings that, with probability at least $1/2$, contains x.

Exercise 7.7 (proving Theorem 7.8) Recall that the proof of Theorem 7.7 establishes the existence of a $\text{poly}(n/\varepsilon)$-time oracle machine M such that, for every $B : \{0,1\}^n \to \{0,1\}$ and every $x \in \{0,1\}^n$ that satisfy $\mathsf{Pr}_r[B(r) = b(x,r)] \geq \frac{1}{2} + \varepsilon$, it holds that $\mathsf{Pr}[M^B(n,\varepsilon) = x] = \Omega(\varepsilon^2/n)$. Show that this implies Theorem 7.8. (Indeed, an alternative proof can be derived by adapting Exercise 7.6.)

Guideline: Apply a "coupon collector" argument.

Exercise 7.8 A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a universal hard-core predicate if for every one-way function f, the predicate b is a hard-core of f. Note that the predicate presented in Theorem 7.7 is "almost universal" (i.e., for every one-way function f, that predicate is a hard-core of $f'(x,r) = (f(x),r)$, where $|x| = |r|$). Prove that there exist no universal hard-core predicate.

Guideline: Let b be a candidate universal hard-core predicate, and let f be an arbitrary one-way function. Then consider the function $f'(x) = (f(x), b(x))$.

Exercise 7.9 Prove that if $\mathcal{NP}$ is not contained in $\mathcal{P}/\text{poly}$ then neither is $\mathcal{E}$. Furthermore, for every $S : \mathbb{N} \to \mathbb{N}$, if some problem in $\mathcal{NP}$ does not have circuits of size S then for some constant $\varepsilon > 0$ there exists a problem in $\mathcal{E}$ that does not have circuits of size S', where $S'(n) = S(n^\varepsilon)$. Repeat the exercise for the "almost everywhere" case.

Guideline: Although $\mathcal{NP}$ is not known to be in $\mathcal{E}$, it is the case that `SAT` is in $\mathcal{E}$, which implies that $\mathcal{NP}$ is reducible to a problem in $\mathcal{E}$. For the "almost everywhere" case, address the fact that the said reduction may not preserve the length of the input.

Exercise 7.10 For every function $f : \{0,1\}^n \to \{0,1\}$, present a linear-size circuit C_n such that $\mathsf{Pr}[C(U_n) = f(U_n)] \geq 0.5 + 2^{-n}$. Furthermore, for every $t \leq 2^{n-1}$, present a circuit C_n of size $O(t \cdot n)$ such that $\mathsf{Pr}[C(U_n) = f(U_n)] \geq 0.5 + t \cdot 2^{-n}$. Warning: you may not assume that $\mathsf{Pr}[f(U_n) = 1] = 0.5$.

Exercise 7.11 (self-correction of low-degree polynomials) Let d, m be integers, and F be a finite field of cardinality greater than $t \stackrel{\text{def}}{=} dm + 1$. Let $p : F^m \to F$ be a polynomial of individual degree d, and $\alpha_1, ..., \alpha_t$ be t distinct non-zero elements of F.

1. Show that, for every $x, y \in F^m$, the value of $p(x)$ can be efficiently computed from the values of $p(x + \alpha_1 y), ..., p(x + \alpha_t y)$, where x and y are viewed as m-ary vectors over F.

2. Show that, for every $x \in F^m$ and $\alpha \in F \setminus \{0\}$, if we uniformly select $r \in F^m$ then the point $x + \alpha r$ is uniformly distributed in F^m.

Conclude that $p(x)$ can be recovered based on t random points, where each point is uniformly distributed in F^m.

Exercise 7.12 (low degree extension) Prove that for any $H \subset F$ and every function $f : H^m \to F$ there exists an m-variate polynomial $\hat{f} : F^m \to F$ of individual degree $|H| - 1$ such that for every $x \in H^m$ it holds that $\hat{f}(x) = f(x)$.

Guideline: Define $\hat{f}(x) = \sum_{a \in H^m} \delta_a(x) \cdot f(a)$, where δ_a is an m-variate of individual degree $|H| - 1$ such that $\delta_a(a) = 1$ whereas $\delta_a(x) = 0$ for every $x \in H^m \setminus \{a\}$. Specifically, $\delta_{a_1,...,a_m}(x_1, ..., x_m) = \prod_{i=1}^m \prod_{b \in H \setminus \{a_i\}} ((x_i - b)/(a_i - b))$.

Exercise 7.13 Suppose that $\hat{f}$ and S' are as in the conclusion of Theorem 7.12. Prove that there exists a Boolean function g in $\mathcal{E}$ that is (S'', ε)-inapproximable for $S''(n' + O(\log n')) = S'(n')/n'$ and $\varepsilon(m) = 1/m^3$.

Guideline: Consider the function g defined such that $g(x, i)$ equals the i^{th} bit of $\hat{f}(x)$.

Exercise 7.14 (a generic application of Theorem 7.8) For any $\ell : \mathbb{N} \to \mathbb{N}$, let $h : \{0,1\}^* \to \{0,1\}^*$ be a function such that $|h(x)| = \ell(|x|)$ for every $x \in \{0,1\}^*$, and $\{X_n\}_{n \in \mathbb{N}}$ be a probability ensemble. Suppose that, for some $s : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0, 1]$, for every family of s-size circuits $\{C_n\}_{n \in \mathbb{N}}$ and all sufficiently large n it holds that $\Pr[C_n(X_n) = h(X_n)] \leq \varepsilon(n)$. Suppose that $s' : \mathbb{N} \to \mathbb{N}$ and $\varepsilon' : \mathbb{N} \to [0, 1]$ satisfy $s'(n + \ell(n)) \leq s(n)/\text{poly}(n/\varepsilon'(n + \ell(n)))$ and $\varepsilon'(n + \ell(n)) \geq 2\varepsilon(n)$. Show that Theorem 7.8 implies that for every family of s'-size circuits $\{C'_{n'}\}_{n' \in \mathbb{N}}$ and all sufficiently large $n' = n + \ell(n)$ it holds that

$$\Pr[C'_{n+\ell(n)}(X_n, U_{\ell(n)}) = b(h(X_n), U_{\ell(n)})] \leq \frac{1}{2} + \varepsilon'(n + \ell(n)),$$

where $b(y, r)$ denotes the inner-product mod 2 of y and r. Note that if X_n is uniform over $\{0,1\}^n$ then the predicate $h'(x, r) = b(h(x), r)$, where $|r| = |h(x)|$, is $(s', 1 - 2\varepsilon')$-inapproximable. Conclude that, in this case, if $\varepsilon(n) = 1/s(n)$ and $s'(n + \ell(n)) = s(n)^{\Omega(1)}/\text{poly}(n)$, then h' is s'-inapproximable.

Exercise 7.15 (reversing Exercise 7.14 (by Viola and Wigderson)) Let $\ell : \mathbb{N} \to \mathbb{N}$, $h : \{0,1\}^* \to \{0,1\}^*$, $\{X_n\}_{n \in \mathbb{N}}$, and b be as in Exercise 7.14. Let $H(x, r) = b(h(x), r)$ and recall that in Exercise 7.14 we reduced guessing h to approximating H. Present a reduction in the opposite direction. That is, show that if H is $(s, 1-\varepsilon)$-inapproximable (over $\{X_n\}_{n \in \mathbb{N}}$) then every s'-size circuit succeeds in computing h (over $\{X_n\}_{n \in \mathbb{N}}$) with probability at most ε, where $s'(n) = s(n) - O(\ell(n))$.

Guideline: As usual, start by assuming the existence of a s'-size circuit that computes h with success probability exceeding ε. Consider two correlated random variables X and Y, each distributed over $\{0,1\}^{\ell(n)}$, where X represents the value of $h(U_n)$ and Y represents the circuit's guess for this value. Prove that, for a uniformly distributed $r \in \{0,1\}^{\ell(n)}$, it holds that $\Pr[b(X, r) = b(Y, r)] = (1 + p)/2$, where $p \stackrel{\text{def}}{=} \Pr[X = Y]$.

Exercise 7.16 (derandomization via averaging arguments) Let $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ be a circuit, which may represent a "probabilistic circuit" that processes the first input using a sequence of choices that are given as a second input. Let X and Z be two independent random variables distributed over $\{0,1\}^n$ and $\{0,1\}^m$, respectively, and let χ be a Boolean predicate (which may represent a success event regarding the behavior of C). Prove that there exists a string $z \in \{0,1\}^m$ such that for $C_z(x) \stackrel{\text{def}}{=} C(x,z)$ it holds that $\Pr[\chi(X, C_z(X)) = 1] \geq \Pr[\chi(X, C(X,Z))=1]$.

Exercise 7.17 (reducing "selective XOR" to "standard XOR") Let f be a Boolean function, and $b(y,r)$ denote the inner-product modulo 2 of the equal-length strings y and r. Suppose that $F'(x_1, ..., x_{t(n)}, r) \stackrel{\text{def}}{=} b(f(x_1) \cdots f(x_{t(n)}), r)$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $r \in \{0,1\}^{t(n)}$, is T'-inapproximable. Assuming that $n \mapsto t(n) \cdot n$ is 1-1, prove that $F(x) \stackrel{\text{def}}{=} F'(x, 1^{t'(|x|)})$, where $t'(t(n) \cdot n) = t(n)$, is T-inapproximable for $T(m) = T'(m + t'(m)) - O(m)$.

Guideline: Reduce the approximation of F' to the approximation of F. An important observation is that for any $x = (x_1, ..., x_{t(n)})$, $x' = (x'_1, ..., x'_{t(n)})$, and $r = r_1 \cdots r_{t(n)}$ such that $x'_i = x_i$ if $r_i = 1$, it holds that $F'(x,r) = F(x') \oplus \oplus_{i:r_i=0} f(x'_i)$. This suggests a non-uniform reduction of F' to F, which uses "adequate" $z_1, ..., z_{t(n)} \in \{0,1\}^n$ as well as the corresponding values $f(z_i)$'s as advice. On input $x_1, ..., x_{t(n)}, r_1 \cdots r_{t(n)}$, the reduction sets $x'_i = x_i$ if $r_i = 1$ and $x'_i = z_i$ otherwise, makes the query $x' = (x'_1, ..., x'_{t(n)})$ to F, and returns $F(x') \oplus_{i:r_i=0} f(z_i)$. Analyze this reduction in the case that $z_1, ..., z_{t(n)} \in \{0,1\}^n$ are uniformly distributed, and infer that they can be set to some fixed values (see Exercise 7.16).[33]

Exercise 7.18 (reducing "standard XOR" to "selective XOR") In continuation to Exercise 7.17, show a reduction in the opposite direction. That is, for F and F' as in Exercise 7.17, show that if F is T-inapproximable then F' is T'-inapproximable, where $T'(m + t'(m)) = \min(T(m) - O(m), \exp(t'(m)/O(1)))^{1/3}$.

Guideline: Reduce the approximation of F to the approximation of F', using the fact that for any $x = (x_1, ..., x_{t(n)})$ and $r = r_1 \cdots r_{t(n)}$ it holds that $\oplus_{i \in S_r} f(x_i) = F'(x,r)$, where $S_r = \{i \in [t(n)] : r_i = 1\}$. Note that, with probability $1 - \exp(-\Omega(t(n)))$, the set S_r contains at least $t(n)/3$ indices. Thus, the XOR of $t(n)/3$ values of f can be reduced to the selective XOR of $t(n)$ such values (by using some of the ideas used in Exercise 7.17 for handling the case that $|S_r| > t(n)/3$). The XOR of $t(n)$ values can be obtained by three XORs (of $t(n)/3$ values each), at the cost of decreasing the advantage by raising it to a power of three.

Exercise 7.19 (reducing "selective XOR" to direct product) Recall that, in §7.2.1.2, the approximation of the "selective XOR" predicate P' was reduced to

[33] That is, assume first that the reduction is given $t(n)$ samples of the distribution $(U_n, f(U_n))$, and analyze its success probability on a uniformly distributed input $(x,r) = (x_1, ..., x_{t(n)}, r_1 \cdots r_{t(n)})$. Next, apply Exercise 7.16 when X represents the distribution of the actual input (x,r), and Z represents the the distribution of the auxiliary sequence of samples.

the guessing of the value of the direct product function P. Present a reduction in the opposite direction. That is, for P and P' as in §7.2.1.2, show that if P' is T'-inapproximable then every T-size circuit succeeds in computing P with probability at most $1/T$, where $T = \Omega(T')$.

Guideline: Use Exercise 7.15.

Exercise 7.20 (Theorem 7.14 versus Theorem 7.5) Consider a generalization of Theorem 7.14 in which f and P are functions from strings to sets of strings such that $P(x_1, ..., x_t) = f(x_1) \times \cdots \times f(x_t)$.

1. Prove that if for every family of p_1-size circuits, $\{C_n\}_{n\in\mathbb{N}}$, and all sufficiently large $n \in \mathbb{N}$, it holds that $\mathsf{Pr}[C_n(U_n) \notin f(U_n)] > 1/p_2(n)$ then for every family of p'-size circuits, $\{C'_m\}_{m\in\mathbb{N}}$, it holds that $\mathsf{Pr}[C'_m(U_m) \in P(U_m)] < \varepsilon'(m)$, where ε' and p' are as in Theorem 7.14. Further generalize the claim by replacing $\{U_n\}_{n\in\mathbb{N}}$ with an arbitrary distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$, and replacing U_m by a $t(n)$-fold Cartesian product of X_n (where $m = t(n) \cdot n$).

2. Show that the foregoing generalizes both Theorem 7.14 and a non-uniform complexity version of Theorem 7.5.

Exercise 7.21 (refinement of the main theme of §7.2.1.3) Consider the following modification of Definition 7.17, in which the decoding condition refers to an agreement threshold of $(1/q(N)) + \alpha(N)$ rather than to a threshold of $\alpha(N)$. The modified definition reads as follows (where p is a fixed polynomial): *For every $w : [\ell(N)] \to [q(N)]$ and $x \in \{0,1\}^N$ such that $\Gamma(x)$ is $(1 - ((1/q(N)) + \alpha(N)))$-close to w, there exists an oracle-aided circuit C of size $p((\log N)/\alpha(N))$ such that $C^w(i)$ yields the i^{th} bit of x for every $i \in [N]$.*

1. Formulate and prove a version of Theorem 7.18 that refers to the modified definition (rather than to the original one).

   Guideline: The modified version should refer to computing $g(U_{m(n)})$ with success probability greater than $(1/q(n)) + \varepsilon(n)$ (rather than greater than $\varepsilon(n)$).

2. Prove that, when applied to binary codes (i.e., $q \equiv 2$), the version in Item 1 yields S''-inapproximable predicates, for $S''(n') = S(m^{-1}(n'))^{\Omega(1)}/\text{poly}(n')$.

3. Prove that the Hadamard Code allows implicit decoding under the modified definition (but not according to the original one).[34]

   Guideline: This is the actual contents of Theorem 7.8.

Show that if $\Gamma : \{0,1\}^N \to [q(N)]^{\ell(N)}$ is a (non-binary) code that allows implicit decoding then encoding its symbols by the Hadamard code yields a binary code $(\{0,1\}^N \to \{0,1\}^{\ell(N)\cdot 2^{\lceil \log_2 q(N) \rceil}})$ that allows implicit decoding. Note that efficient encoding is preserved only if $q(N) \leq \text{poly}(N)$.

[34]Needless to say, the Hadamard Code is not efficient (for the trivial reason that its codewords have exponential length).

Exercise 7.22 (using Proposition 7.16 to prove Theorem 7.19) Prove Theorem 7.19 by combining Proposition 7.16 and Theorem 7.8.

Guideline: Note that, for some $\gamma > 0$, Proposition 7.16 yields an exponential-time computable function $\hat{f}$ such that $|\hat{f}(x)| \leq |x|$ and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = S(n'/3)^{\gamma}/\text{poly}(n')$ it holds that $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < 1/S'(n')$. Combining this with Theorem 7.8, infer that $P(x,r) = b(\hat{f}(x), r)$, where $|r| = |\hat{f}(x)| \leq |x|$, is S''-inapproximable for $S''(n'') = S(n''/2)^{\Omega(1)}/\text{poly}(n'')$. Note that if $S(n) = 2^{\Omega(n)}$ then $S''(n'') = 2^{\Omega(n'')}$.

Exercise 7.23 Let G be a pairwise independent generator (i.e., as in Lemma 7.22), $S \subset \{0,1\}^n$ and $\alpha \stackrel{\text{def}}{=} |S|/2^n$. Prove that, with probability at least $\min(n \cdot \alpha, 1)/2$, at least one of the n strings output by $G(U_{2n})$ resides in S. Furthermore, if $\alpha \leq 1/2n$ then this probability is at least $0.75 \cdot n \cdot \alpha$.

Guideline: Using the pairwise independence property and employing the Inclusion-Exclusion formula, we lower-bound the aforementioned probability by $n \cdot \alpha - \binom{n}{2} \cdot \alpha^2$. If $\alpha \leq 1/n$ then the claim follows, otherwise we employ the same reasoning to the first $1/\alpha$ elements in the output of $G(U_{2n})$.

Exercise 7.24 (one-way functions versus inapproximable predicates) Prove that the existence of a non-uniformly hard one-way function (as in Definition 7.3) implies the existence of an exponential-time computable predicate that is T-inapproximable (as per Definition 7.9), for every polynomial T.

Guideline: Suppose first that the one-way function f is length-preserving and 1-1. Consider the hard-core predicate b guaranteed by Theorem 7.7 for $g(x,r) = (f(x), r)$, define the Boolean function h such that $h(z) = b(g^{-1}(z))$, and show that h is T-inapproximable for every polynomial T. For the general case a different approach seems needed. Specifically, given a (length preserving) one-way function f, consider the Boolean function h defined as $h(z,i,\sigma) = 1$ if and only if the i^{th} bit of the lexicographically first element in $f^{-1}(z) = \{x : f(x) = z\}$ equals σ. (In particular, if $f^{-1}(z) = \emptyset$ then $h(z,i,\sigma) = 0$ for every i and σ.)[35] Note that h is computable in exponential-time, but is not (worst-case) computable by polynomial-size circuits. Applying Theorem 7.10, we are done.

[35]Thus, h may be easy to computed in the average-case sense (e.g., if $f(x) = 0^{|x|}f'(x)$ for some one-way function f').

Chapter 8

Pseudorandom Generators

Indistinguishable things are identical.[1]

G.W. Leibniz (1646–1714)

A fresh view at the *question of randomness* has been taken by complexity theory: it has been postulated that a distribution is random (or rather pseudorandom) if it cannot be told apart from the uniform distribution by any efficient procedure. Thus, (pseudo)randomness is not an inherent property of an object, but is rather subjective to the observer.

At the extreme, this approach says that the question of whether the world is deterministic or allows for some free choice (which may be viewed as sources of randomness) is irrelevant. *What matters is how the world looks to us and to various computationally bounded devices.* That is, if some phenomenon looks random then we may just treat it as if it were random. Likewise, if we can generate sequences that cannot be told apart from the uniform distribution by any efficient procedure, then we can use these sequences in any efficient randomized application instead of the ideal coin tosses that are postulated in the design of this application.

The pivot of the foregoing approach is the notion of *computational indistinguishability*, which refers to pairs of distributions that cannot be told apart by efficient procedures. The most fundamental incarnation of this notion associates efficient procedures with polynomial-time algorithms, but other incarnations that restrict attention to other classes of distinguishing procedures also lead to important insights. Likewise, the *effective generation* of pseudorandom objects, which is of major concern, is actually a general paradigm with numerous useful incarnations (which differ in the computational complexity limitations imposed on the generation process).

[1]This is Leibniz's *Principle of Identity of Indiscernibles.* Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent. We thus believe that he would have agreed to the theme of this chapter, which asserts that *indistinguishable things should be considered as if they were identical.*

Summary: Pseudorandom generators are efficient deterministic procedures that stretch short random seeds into longer pseudorandom sequences. Thus, a generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*).

The archetypical case of pseudorandom generators refers to efficient generators that fool any feasible procedure; that is, the potential distinguisher is any probabilistic polynomial-time algorithm, which may be more complex than the generator itself (which, in turn, has time-complexity bounded by a fixed polynomial). These generators are called general-purpose, because their output can be safely used in any efficient application. Such (general-purpose) pseudorandom generators exist if and only if one-way functions exist.

For purposes of derandomization one may use pseudorandom generators that are somewhat more complex than the potential distinguisher (which represents the algorithm to be derandomized). Following this approach, suitable pseudorandom generators, which can be constructed assuming the existence of problems in $\mathcal{E}$ that have no sub-exponential size circuits, yield a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$).

It is also beneficial to consider pseudorandom generators that fool space-bounded distinguishers and generators that exhibit some limited random behavior (e.g., outputting a pair-wise independent or a small-bias sequence). Such (special-purpose) pseudorandom generators can be constructed without relying on any computational complexity assumption.

Introduction

The "question of randomness" has been puzzling thinkers for ages. Aspects of this question range from philosophical doubts regarding the existence of randomness (in the world) and reflections on the meaning of randomness (in our thinking) to technical questions regarding the measuring of randomness. Among many other things, the second half of the 20th century has witnessed the development of three theories of randomness, which address different aspects of the foregoing question.

The first theory (cf., [62]), initiated by Shannon [202], views randomness as representing *lack of information*, which in turn is modeled by a probability distribution on the possible values of the missing data. Indeed, Shannon's Information Theory is rooted in probability theory and is focused at distributions that are not perfectly random. It characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., in this case there is no redundancy at all). Thus, perfect randomness is associated with a unique distribution – the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [152, 155]), initiated by Solomonov [209], Kolmogorov [146], and Chaitin [50], views randomness as representing lack of structure, which in turn is reflected in the length of the most succinct and effective description of the object. The notion of a succinct and *effective description* refers to a process that transforms the succinct description to an explicit one. Indeed, this theory of randomness is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device; see §1.2.3.4). It measures the randomness (or complexity) of objects in terms of the shortest program (for a fixed universal machine) that generates the object.[2] Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, following Kolmogorov's approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, by definition, one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory, which is the focus of the current chapter, views randomness as an effect on an observer and thus as being relative to the *observer's abilities* (of analysis). The observer's abilities are modeled as its computational abilities (i.e., the complexity of the processes that the observer may apply), and hence this theory of randomness is rooted in complexity theory. This theory of randomness is explicitly aimed at providing a notion of randomness that, unlike the previous two notions, allows for an efficient (and deterministic) generation of random strings from shorter random seeds. The heart of this theory is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To illustrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" in one of the following four ways. In each of them Alice flips an unbiased coin and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess.
>
> In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2.
>
> In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion. Thus we believe that, also in this case, Bob wins with probability 1/2.

[2]We mention that Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable).

> The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess.
>
> In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. At the extreme, even events that are fully determined by public information may be perceived as random events by an observer that lacks the relevant information and/or the ability to process it. Our focus will be on the lack of sufficient processing power, and not on the lack of sufficient information. The lack of sufficient processing power may be due either to the formidable amount of computation required (for analyzing the event in question) or to the fact that the observer happens to be very limited.

A natural notion of pseudorandomness arises – a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This specific notion of pseudorandomness is indeed the most fundamental one, and much of this chapter is focused on it. Weaker notions of pseudorandomness arise as well – they refer to indistinguishability by weaker procedures such as space-bounded algorithms, constant-depth circuits, etc. Stretching this approach even further one may consider algorithms that are designed on purpose so not to distinguish even weaker forms of "pseudorandom" sequences from random ones (where such algorithms arise naturally when trying to convert some natural randomized algorithm into deterministic ones; see Section 8.5).

The foregoing discussion has focused at one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). A natural approach requires the generation process to be efficient, and furthermore to be fixed before the specific observer is determined. Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – these operating in (fixed) polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. In particular, this means that the distinguisher is allowed more resources than the generator. Such (general-purpose) pseudorandom generators (discussed in Section 8.2) allow to decrease the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography. The term *general-purpose* is meant to emphasize the fact that the same generator is good for all efficient applications, including those that consume more resources than the generator itself.

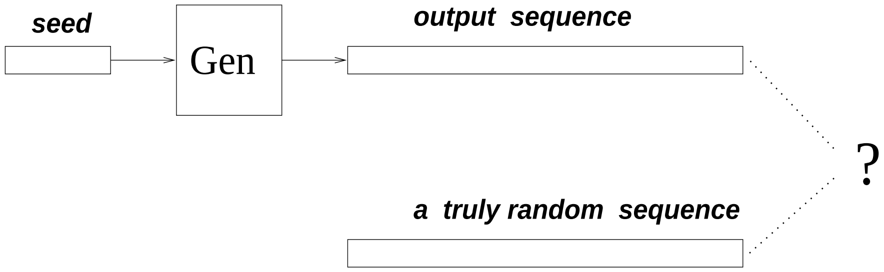


Figure 8.1: Pseudorandom generators – an illustration.

Although general-purpose pseudorandom generators are very appealing, there are important reasons for considering also the opposite relation between the complexities of the generation and distinguishing tasks; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones), where the crucial step is replacing the random input of an algorithm by a pseudorandom input, which in turn can be generated based on a much shorter random seed. In particular, when derandomizing a probabilistic polynomial-time algorithm, the observer (to be fooled by the generator) is a fixed algorithm. In this case employing a more complex generator merely means that the complexity of the derived deterministic algorithm is dominated by the complexity of the generator (rather than by the complexity of the original randomized algorithm). Needless to say, allowing the generator to use more resources than the observer that it tries to fool makes the task of designing pseudorandom generators potentially easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators. The usefulness of this approach is demonstrated in Sections 8.3 through 8.5.

We note that the goal of all types of pseudorandom generators is to allow the generation of "sufficiently random" sequences based on much shorter random seeds. Thus, pseudorandom generators offer significant saving in the randomness complexity of various applications (and in some cases eliminating randomness altogether). Saving on randomness is valuable because many applications are severely limited in their ability to generate or obtain truly random bits. Furthermore, typically, generating truly random bits is significantly more expensive than standard computation steps. Thus, randomness is a computational resource that should be considered on top of time complexity (analogously to the consideration of space complexity).

Organization. In Section 8.1 we present the general paradigm underlying the various notions of pseudorandom generators. The archetypical case of general-purpose pseudorandom generators is presented in Section 8.2. We then turn to alternative notions of pseudorandom generators: generators that suffice for the derandomization of complexity classes such as $\mathcal{BPP}$ are discussed in Section 8.3; pseudorandom generators in the domain of space-bounded computations are discussed in Section 8.4; and special-purpose generators are discussed in Section 8.5.

Teaching note: If you can afford teaching only one of the alternative notions of pseudorandom generators, then we suggest teaching the notion of general-purpose pseudorandom generators (presented in Section 8.2). This notion is more relevant to computer science at large and the technical material is relatively simpler. The chapter is organized to facilitate this option.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1). In particular, standard conventions regarding random variables (presented in Appendix D.1.1) will be extensively used. We shall also apply a couple of results from Chapter 7, but these applications will be self-contained.

8.1 The General Paradigm

Teaching note: We advocate a unified view of various notions of pseudorandom generators. That is, we view these notions as incarnations of a general abstract paradigm, to be presented in this section. A teacher that wishes to focus on one of these incarnations may still use this section as a general motivation towards the specific definitions used later. On the other hand, some students may prefer reading this section after studying one of the specific incarnations.

A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*). Let us elaborate.

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that the generator is a *deterministic algorithm* that stretches short strings, called seeds, into longer output sequences.[3] Specifically, this algorithm stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell : \mathbb{N} \to \mathbb{N}$ is called the stretch measure (or stretch function) of the generator. In some settings the specific stretch measure is immaterial (e.g., see Section 8.2.4).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that the generator "fools" some non-trivial algorithms. That is, it is required that any algorithm taken from a predetermined class of interest cannot distinguish the output produced by the generator (when the generator is fed with a uniformly chosen seed) from a uniformly chosen sequence.

[3]Indeed, the seed represents the randomness that is used in the generation of the output sequences; that is, the randomized generation process is decoupled into a deterministic algorithm and a random seed. This decoupling facilitates the study of such processes.

Thus, we consider a class $\mathcal{D}$ of distinguishers (e.g., probabilistic polynomial-time algorithms) and a class $\mathcal{F}$ of (threshold) functions (e.g., reciprocals of positive polynomials), and require that the generator G satisfies the following: For any $D \in \mathcal{D}$, any $f \in \mathcal{F}$, and for all sufficiently large k's it holds that

$$|\Pr[D(G(U_k)) = 1] - \Pr[D(U_{\ell(k)}) = 1]| < f(k), \tag{8.1}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic. The reader may think of such a distinguisher, D, as trying to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed as $U_{\ell(k)}$). The condition in Eq. (8.1) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases.[4] The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

Complexity of Generation: The archetypical choice is that the generator has to work in polynomial-time (in length of its input – the seed). Other choices will be discussed as well. We note that placing no computational requirements on the generator (or, alternatively, putting very mild requirements such as upper-bounding the running-time by a double-exponential function), yields "generators" that can fool any subexponential-size circuit family (see Exercise 8.1).

Notational conventions. We will consistently use k for denoting the length of the seed of a pseudorandom generator, and $\ell(k)$ for denoting the length of the corresponding output. In some cases, this makes our presentation a little more cumbersome (since a more natural presentation may specify some other parameters and let the seed-length be a function of the latter). However, our choice has the advantage of focusing attention on the fundamental parameter of pseudorandom generation process – the length of the random seed. We note that whenever a pseudorandom generator is used to "derandomize" an algorithm, n will denote the length of the input to this algorithm, and k will be selected as a function of n.

Some instantiations of the general paradigm. Two important instantiations of the notion of pseudorandom generators relate to polynomial-time distinguishers.

1. General-purpose pseudorandom generators correspond to the case that the generator itself runs in polynomial-time and needs to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for

[4]The class of threshold functions $\mathcal{F}$ should be viewed as determining the class of noticeable probabilities (as a function of k). Thus, we require certain functions (i.e., those presented at the l.h.s of Eq. (8.1)) to be smaller than any noticeable function *on all but finitely many integers*. We call the former functions negligible. Note that a function may be neither noticeable nor negligible (e.g., it may be smaller than any noticeable function on infinitely many values and yet larger than some noticeable function on infinitely many other values).

more time than the generator. Thus, the same generator may be used safely in any efficient application. (This notion is treated in Section 8.2.)

2. In contrast, pseudorandom generators intended for derandomization may run more time than the distinguisher, which is viewed as a fixed circuit having size that is upper-bounded by a fixed polynomial. (This notion is treated in Section 8.3.)

In addition, the general paradigm may be instantiated by focusing on the space-complexity of the potential distinguishers (and the generator), rather than on their time-complexity. Furthermore, one may also consider distinguishers that merely reflect probabilistic properties such as pair-wise independence, small-bias, and hitting frequency.

8.2 General-Purpose Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and it is of course central to cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the usage of randomness in real implementations. Thus, general-purpose pseudorandom generators (as defined next) are a key ingredient in an "algorithmic tool-box" – they provide an automatic compiler of programs written with free usage of randomness into programs that make an economical use of randomness.

Organization of this section. Since this is a relatively long section, a short road-map seems in place. In Section 8.2.1 we provide the basic definition of general-purpose pseudorandom generators, and in Section 8.2.2 we describe their archetypical application (which was eluded to in the former paragraph). In Section 8.2.3 we provide a wider perspective on the notion of computational indistinguishability that underlies the basic definition, and in Section 8.2.4 we justify the little concern (shown in Section 8.2.1) regarding the specific stretch function. In Section 8.2.5 we address the existence of general-purpose pseudorandom generators. In Section 8.2.6 we motivate and discuss a non-uniform version of computational indistinguishability. We conclude in Section 8.2.7 by reviewing other variants and reflecting on various conceptual aspects of the notions discussed in this section.

8.2.1 The basic definition

Loosely speaking, general-purpose pseudorandom generators are efficient deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by *any* efficient algorithm. Identifying efficiency with polynomial-time operation, this means that the generator (being a fixed algorithm) works within *some fixed* polynomial-time, whereas the distinguisher may be *any* algorithm that runs in polynomial-time. Thus, the distinguisher is potentially more complex than the generator; for example, the distinguisher *may* run in time that is cubic in the running-time of the generator. Furthermore, to facilitate the development of this theory, we allow the distinguisher to be probabilistic (whereas the generator remains deterministic as stated previously). We require that such distinguishers cannot tell the output of the generator from a truly random string of similar length, or rather that the difference that such distinguishers may detect (or "sense") is negligible. Here a negligible function is a function that vanishes faster than the reciprocal of any positive polynomial.[5]

Definition 8.1 (general-purpose pseudorandom generator): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretch function, $\ell : \mathbb{N} \rightarrow \mathbb{N}$ (satisfying $\ell(k) > k$ for all k), *such that for any probabilistic polynomial-time algorithm D, for any positive polynomial p, and for all sufficiently large k's it holds that*

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] \;-\; \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| \;<\; \frac{1}{p(k)} \tag{8.2}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the internal coin tosses of D.

Thus, Definition 8.1 is derived from the generic framework (presented in Section 8.1) by taking the class of distinguishers to be the set of all probabilistic polynomial-time algorithms, and taking the class of (noticeable) threshold functions to be the set of all functions that are the reciprocals of some positive polynomial.[6] Indeed, the principles underlying Definition 8.1 were discussed in Section 8.1 (and will be further discussed in Section 8.2.3).

We note that Definition 8.1 does not make any requirement regarding the stretch function $\ell : \mathbb{N} \rightarrow \mathbb{N}$, except for the generic requirement that $\ell(k) > k$ for all k. Needless to say, the larger ℓ is the more useful is the pseudorandom generator. Of course, ℓ is upper-bounded by the running-time of the generator (and hence by a polynomial). In Section 8.2.4 we show that any pseudorandom generator (even one having minimal stretch $\ell(k) = k + 1$) can be used for constructing a pseudorandom generator having any desired (polynomial) stretch function. But before doing so, we rigorously discuss the "saving in randomness" offered by pseudorandom generators, and provide a wider perspective on the notion of computational indistinguishability that underlies Definition 8.1.

[5] Definition 8.1 requires that the functions representing the distinguishing gap of certain algorithms should be smaller than the reciprocal of any positive polynomial for all but finitely many k's. The former functions are called *negligible* (cf. Footnote 4, when identifying noticeable functions with the reciprocals of any positive polynomial). The notion of negligible probability is robust in the sense that any event that occurs with negligible probability will occur with negligible probability also when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

[6] The latter choice is naturally coupled with the association of efficient computation with polynomial-time algorithms: An event that occurs with noticeable probability occurs almost always when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

8.2.2 The archetypical application

We note that "pseudo-random number generators" appeared with the first computers, and have been used ever since for generating random choices (or samples) for various applications. However, typical implementations use generators that are not pseudorandom according to Definition 8.1. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test (cf., [145]). We warn that the fact that a "pseudo-random number generator" passes some statistical tests, does not mean that it will pass a new test and that it will be good for a future (untested) application. Needless to say, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses." In contrast, the approach encompassed in Definition 8.1 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 8.1, covers all possible efficient applications and guarantees that for all of them pseudorandom sequences are as good as truly random ones. Indeed, any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. This substitution is spell-out next.

Construction 8.2 (typical application of pseudorandom generators): *Let G be a pseudorandom generator with stretch function $\ell:\mathbb{N}\to\mathbb{N}$. Let A be a probabilistic polynomial-time algorithm, and $\rho:\mathbb{N}\to\mathbb{N}$ denote its randomness complexity. Denote by $A(x,r)$ the output of A on input x and coin tosses sequence $r\in\{0,1\}^{\rho(|x|)}$. Consider the following randomized algorithm, denoted A_G:*

> *On input x, set $k=k(|x|)$ to be the smallest integer such that $\ell(k)\geq\rho(|x|)$, uniformly select $s\in\{0,1\}^k$, and output $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

That is, $A_G(x,s)=A(x,G'(s))$, for $|s|=k(|x|)=\text{argmin}_i\{\ell(i)\geq\rho(|x|)\}$, where $G'(s)$ is the $\rho(|x|)$-bit long prefix of $G(s)$.

Thus, using A_G instead of A, the randomness complexity is reduced from ρ to $\ell^{-1}\circ\rho$, while (as we show next) it is infeasible to find inputs (i.e., x's) on which the *noticeable behavior* of A_G is different from the one of A. For example, if $\ell(k)=k^2$, then the randomness complexity is reduced from ρ to $\sqrt{\rho}$. We stress that the pseudorandom generator G is *universal*; that is, it can be applied to reduce the randomness complexity of *any* probabilistic polynomial-time algorithm A.

Proposition 8.3 *Let A, ρ and G be as in Construction 8.2, and suppose that $\rho:\mathbb{N}\to\mathbb{N}$ is 1-1. Then, for every pair of probabilistic polynomial-time algorithms,* a finder *F and* a tester *T, every positive polynomial p and all sufficiently long n's*

$$\sum_{x\in\{0,1\}^n}\mathsf{Pr}[F(1^n)=x]\cdot|\,\Delta_{A,T}(x)\,| \;<\; \frac{1}{p(n)} \tag{8.3}$$

where $\Delta_{A,T}(x) \stackrel{\text{def}}{=} \Pr[T(x, A(x, U_{\rho(|x|)})) = 1] - \Pr[T(x, A_G(x, U_{k(|x|)})) = 1]$, *and the probabilities are taken over the* U_m*'s as well as over the internal coin tosses of the algorithms* F *and* T.

Algorithm F represents a potential attempt to find an input x on which the output of A_G is distinguishable from the output of A. This "attempt" may be benign as in the case that a user employs algorithm A_G on inputs that are generated by some probabilistic polynomial-time application. However, the attempt may also be adversarial as in the case that a user employs algorithm A_G on inputs that are provided by a potentially malicious party. The potential tester, denoted T, represents the potential use of the output of algorithm A_G, and captures the requirement that this output be as good as a corresponding output produced by A. Thus, T is given x as well as the corresponding output produced either by $A_G(x) \stackrel{\text{def}}{=} A(x, U_{k(|x|)})$ or by $A(x) = A(x, U_{\rho(|x|)})$, and it is required that T cannot tell the difference. In the case that A is a probabilistic polynomial-time *decision procedure*, this means that it is infeasible to find an x on which A_G decides incorrectly (i.e., differently than A). In the case that A is a *search procedure for some NP-relation*, it is infeasible to find an x on which A_G outputs a wrong solution. For details, see Exercise 8.2.

Proof: The proposition is proven by showing that any triple (A, F, T) violating the claim can be converted into an algorithm D that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. The key observation is that for every $x \in \{0,1\}^n$ it holds that

$$\Delta_{A,T}(x) = \Pr[T(x, A(x, U_{\rho(n)}))=1] - \Pr[T(x, A(x, G'(U_{k(n)})))=1], \tag{8.4}$$

where $G'(s)$ is the $\rho(n)$-bit long prefix of $G(s)$. Thus, a method for finding a string x such that $|\Delta_{A,T}(x)|$ is large yields a way of distinguishing $U_{\ell(k(n))}$ from $G(U_{k(n)})$; that is, given a sample $r \in \{0,1\}^{\ell(k(n))}$ and using such a string $x \in \{0,1\}^n$, the distinguisher outputs $T(x, A(x, r'))$, where r' is the $\rho(n)$-bit long prefix of r. Indeed, we shall show that the violation of Eq. (8.3), which refers to $\mathsf{E}_{x \leftarrow F(1^n)}[|\Delta_{A,T}(x)|]$, yields a violation of the hypothesis that G is a pseudorandom generator (by finding an adequate string x and using it). This intuitive argument requires a slightly careful implementation, which is provided next.

As a warm-up, consider the following algorithm D. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), algorithm D first obtains $x \leftarrow F(1^n)$, where n can be obtained easily from $|r|$ (because ρ is 1-1 and $1^n \mapsto \rho(n)$ is computable via A). Next, D obtains $y = A(x, r')$, where r' is the $\rho(|x|)$-bit long prefix of r. Finally D outputs $T(x, y)$. Note that D is implementable in probabilistic polynomial-time, and that

$$\begin{aligned} D(U_{\ell(k(n))}) &\equiv T(X_n, A(X_n, U_{\rho(n)}))\,, \text{ where } X_n \stackrel{\text{def}}{=} F(1^n) \\ D(G(U_{k(n)})) &\equiv T(X_n, A(X_n, G'(U_{k(n)})))\,, \text{ where } X_n \stackrel{\text{def}}{=} F(1^n). \end{aligned}$$

Using Eq. (8.4), it follows that $\Pr[D(U_{\ell(k(n))}) = 1] - \Pr[D(G(U_{k(n)})) = 1]$ equals $\mathsf{E}[\Delta_{A,T}(F(1^n))]$, which implies that $\mathsf{E}[\Delta_{A,T}(F(1^n))]$ must be negligible (because otherwise we derive a contradiction to the hypothesis that G is a pseudorandom generator). This yields a weaker version of the proposition asserting that $\mathsf{E}[\Delta_{A,T}(F(1^n))]$ is negligible (rather than that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|]$ is negligible).

In order to prove that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|]$ (rather than to $\mathsf{E}[\Delta_{A,T}(F(1^n))]$) is negligible, we need to modify D a little. Note that the source of trouble is that $\Delta_{A,T}(\cdot)$ may be positive on some x's and negative on others, and thus it may be the case that $\mathsf{E}[\Delta_{A,T}(F(1^n))]$ is small (due to cancelations) even if $\mathsf{E}[|\Delta_{A,T}(F(1^n))|]$ is large. This difficulty can be overcome by determining the sign of $\Delta_{A,T}(\cdot)$ on $x = F(1^n)$ and changing the outcome of D accordingly; that is, the modified D will output $T(x, A(x, r'))$ if $\Delta_{A,T}(x) > 0$ and $1 - T(x, A(x, r'))$ otherwise. Thus, in each case, the contribution of x to the distinguishing gap of the modified D will be $|\Delta_{A,T}(x)|$. We further note that if $|\Delta_{A,T}(x)|$ is small then it does not matter much whether we act as in the case of $\Delta_{A,T}(x) > 0$ or in the case of $\Delta_{A,T}(x) \leq 0$. Thus, it suffices to correctly determine the sign of $\Delta_{A,T}(x)$ in the case that $|\Delta_{A,T}(x)|$ is large, which is certainly a feasible (approximation) task. Details follow.

We start by assuming, towards the contradiction, that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|] > \varepsilon(n)$ for some non-negligible function ε. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), the modified algorithm D first obtains $x \leftarrow F(1^n)$, just as the basic version. Next, using a sample of size $\mathrm{poly}(n/\varepsilon(n))$, it approximates $p_U(x) \stackrel{\text{def}}{=} \Pr[T(x, A(x, U_{\rho(n)}) = 1]$ and $p_G(x) \stackrel{\text{def}}{=} \Pr[T(x, A(x, G'(U_{k(n)})) = 1]$ such that each probability is approximated to within a deviation of $\varepsilon(n)/8$ with negligible error probability (say, $\exp(-n)$). (Note that, so far, the actions of D only depend on the length of its input r, which determines n.)[7] If these approximations indicate that $p_U(x) \geq p_G(x)$ (equiv., that $\Delta_{A,T}(x) \geq 0$) then D outputs $T(x, A(x, r'))$ else it outputs $1 - T(x, A(x, r'))$, where r' is the $\rho(|x|)$-bit long prefix of r and we assume without loss of generality that the output of T is in $\{0, 1\}$.

The analysis of the modified distinguisher D is based on the fact that *if the approximations yield a correct decision regarding the relation between $p_U(x)$ and $p_G(x)$, then the contribution of x to the distinguishing gap of D is $|p_U(x) - p_G(x)|$.*[8] We also note that if $|p_U(x) - p_G(x)| > \varepsilon(n)/4$, then with overwhelmingly high probability (i.e., $1 - \exp(-n)$) the approximation of $p_U(x) - p_G(x)$ maintains the sign of $p_U(x) - p_G(x)$ (because each of the two quantities is approximated to within an additive error of $\varepsilon(n)/8$). Finally, we note that if $|p_U(x) - p_G(x)| \leq \varepsilon(n)/4$ then we may often err regarding the sign of $p_U(x) - p_G(x)$ but the damage caused (to the distinguishing gap of D) by this error is at most $2|p_U(x) - p_G(x)| \leq \varepsilon(n)/2$. Combining all these observations, we get:

$$\Pr[D(U_{\ell(k(n))}) = 1 | F(1^n) = x] \; - \; \Pr[D(G(U_{k(n)})) = 1 | F(1^n) = x]$$

[7] Specifically, the approximation to $p_U(x)$ (resp., $p_G(x)$) is obtained by generating a sample of $U_{\rho(n)}$ (resp., $G'(U_{k(n)})$) and invoking the algorithms A and T; that is, given a sample $r_1, ..., r_t$ of $U_{\rho(n)}$ (resp., $G'(U_{k(n)})$), where $t = O(n/\varepsilon(n)^2)$, we approximate $p_U(x)$ (resp., $p_G(x)$) by $|\{i \in [t] : T(x, A(x, r_i)) = 1\}|/t$.

[8] Indeed, if $p_U(x) \geq p_G(x)$ then the contribution is $p_U(x) - p_G(x) = |p_U(x) - p_G(x)|$, whereas if $p_U(x) < p_G(x)$ then the contribution is $(1 - p_U(x)) - (1 - p_G(x)) = -(p_U(x) - p_G(x))$, which also equals $|p_U(x) - p_G(x)|$.

$$\geq \quad |p_U(x) - p_G(x)| - \eta(x), \tag{8.5}$$

where $\eta(x) = \varepsilon(n)/2$ if $|p_U(x) - p_G(x)| \leq \varepsilon(n)/4$ and $\eta(x) = \exp(-n)$ otherwise. (Indeed, $\eta(x)$ represents the expected damage due to an error in determining the sign of $p_U(x) - p_G(x)$, where $\varepsilon(n)/2$ upper-bounds the damage caused (by a wrong decision) in the case that $|p_U(x) - p_G(x)| \leq \varepsilon(n)/4$ and $\exp(-n)$ upper-bounds the probability of a wrong decision in the case that $|p_U(x) - p_G(x)| > \varepsilon(n)/4$.) Thus, $\mathsf{Pr}[D(U_{\ell(k(n))})=1] - \mathsf{Pr}[D(G(U_{k(n)}))=1]$ is lower-bounded by the expectation of Eq. (8.5), which equals $\mathsf{E}[|\Delta_{A,T}(F(1^n))|] - \mathsf{E}[\eta(F(1^n))]$. Combining the hypothesis that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|] > \varepsilon(n)$ and the fact that $\max_{x\in\{0,1\}^n}\{\eta(x)\} \leq \varepsilon(n)/2$, we infer that $\mathsf{Pr}[D(U_{\ell(k(n))}) = 1] - \mathsf{Pr}[D(G(U_{k(n)})) = 1] > \varepsilon(n)/2$. Recalling that D runs in time $\mathrm{poly}(n/\varepsilon(n))$, this contradicts the pseudorandomness of G. The proposition follows. ■

Conclusion. Although the foregoing refers to standard probabilistic polynomial-time algorithms, a similar construction and analysis applied to any efficient randomized process (i.e., any efficient multi-party computation). Any such process preserves its behavior when replacing its perfect source of randomness (postulated in its analysis) by a pseudorandom sequence (which may be used in the implementation). Thus, given a pseudorandom generator with a large stretch function, *one can considerably reduce the randomness complexity of any efficient application.*

8.2.3 Computational Indistinguishability

In this section we spell-out (and study) the definition of computational indistinguishability that underlies Definition 8.1.

8.2.3.1 The general formulation

The (general formulation of the) definition of computational indistinguishability refers to *arbitrary* probability ensembles. Here a probability ensemble is an infinite sequence of random variables $\{Z_n\}_{n\in\mathbb{N}}$ such that each Z_n ranges over strings of length that is polynomially related to n (i.e., there exists a polynomial p such that for every n it holds that $|Z_n| \leq p(n)$ and $p(|Z_n|) \geq n$). We say that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \stackrel{\text{def}}{=} |\mathsf{Pr}[A(X_n) = 1] - \mathsf{Pr}[A(Y_n) = 1]|$ is a negligible function in n. That is:

Definition 8.4 (computational indistinguishability): *The probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm D, every positive polynomial p, and all sufficiently large n,*

$$|\mathsf{Pr}[D(X_n)=1] - \mathsf{Pr}[D(Y_n)=1]| \; < \; \frac{1}{p(n)} \tag{8.6}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm D. The l.h.s. of Eq.* (8.6)*, when* *viewed as a function of n, is often called the* distinguishing gap of D, *where* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are understood from the context.*

We can think of D as representing somebody who wishes to distinguish two distributions (based on a given sample drawn from one of the distributions), and think of the output "1" as representing D's verdict that the sample was drawn according to the first distribution. Saying that the two distributions are computationally indistinguishable means that if D is a feasible procedure then its verdict is not really meaningful (because the verdict is almost as often 1 when the sample is drawn from the first distribution as when the sample is drawn from the second distribution). We comment that the absolute value in Eq. (8.6) can be omitted without affecting the definition (see Exercise 8.3), and we will often do so without warning.

In Definition 8.1, we required that the probability ensembles $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$ be computationally indistinguishable. Indeed, an important special case of Definition 8.4 is when one ensemble is uniform, and in such a case we call the other ensemble pseudorandom.

8.2.3.2 Relation to statistical closeness

Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are said to be statistically close (or statistically indistinguishable) if for every positive polynomial p and all sufficient large n the variation distance between X_n and Y_n (i.e., $\frac{1}{2}\sum_z |\Pr[X_n=z]-\Pr[Y_n=z]|$) is bounded above by $1/p(n)$. Clearly, any two probability ensembles that are statistically close are computationally indistinguishable. Needless to say, this is a trivial case of computational indistinguishability, which is due to information theoretic reasons. In contrast, we shall be interested in *non-trivial cases* (of computational indistinguishability), which correspond to probability ensembles that are statistically far apart.

Indeed, as noted in Section 8.1, there exist probability ensembles that are statistically far apart and yet are computationally indistinguishable (see Exercise 8.1). However, at least one of the probability ensembles in Exercise 8.1 is *not* polynomial-time constructible.[9] We shall be much more interested in non-trivial cases of computational indistinguishability in which both ensembles are polynomial-time constructible. An important example is provided by the definition of pseudorandom generators (see Exercise 8.7). As we shall see (in Theorem 8.11), the existence of one-way functions implies the existence of pseudorandom generators, which in turn implies the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable. We mention that this sufficient condition is also necessary (see Exercise 8.9).

8.2.3.3 Indistinguishability by Multiple Samples

The definition of computational indistinguishability (i.e., Definition 8.4) refers to distinguishers that obtain a single sample from one of the two relevant probability

[9] We say that $\{Z_n\}_{n\in\mathbb{N}}$ is polynomial-time constructible if there exists a polynomial-time algorithm S such that $S(1^n)$ and Z_n are identically distributed.

ensembles (i.e., $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$). A very natural generalization of Definition 8.4 refers to distinguishers that obtain several independent samples from such an ensemble.

Definition 8.5 (indistinguishability by multiple samples): *Let $s:\mathbb{N}\to\mathbb{N}$ be polynomially-bounded. Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are* computationally indistinguishable by $s(\cdot)$ samples *if for every probabilistic polynomial-time algorithm, D, every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\left|\Pr\left[D(X_n^{(1)},...,X_n^{(s(n))})=1\right] - \Pr\left[D(Y_n^{(1)},...,Y_n^{(s(n))})=1\right]\right| < \frac{1}{p(n)}$$

where $X_n^{(1)}$ through $X_n^{(s(n))}$ and $Y_n^{(1)}$ through $Y_n^{(s(n))}$ are independent random variables such that each $X_n^{(i)}$ is identical to X_n and each $Y_n^{(i)}$ is identical to Y_n.

It turns out that in the most interesting cases, computational indistinguishability by a single sample implies computational indistinguishability by any polynomial number of samples. One such case is the case of polynomial-time constructible ensembles. We say that the ensemble $\{Z_n\}_{n\in\mathbb{N}}$ is polynomial-time constructible if there exists a polynomial-time algorithm S such that $S(1^n)$ and Z_n are identically distributed.

Proposition 8.6 *Suppose that $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$ are both polynomial-time constructible, and s be a polynomial. Then, X and Y are computationally indistinguishable by a single sample if and only if they are computationally indistinguishable by $s(\cdot)$ samples.*

Clearly, for every polynomial s, computational indistinguishability by $s(\cdot)$ samples implies computational indistinguishability by a single sample (see Exercise 8.5). We now prove that, for efficiently constructible ensembles, indistinguishability by a single sample implies indistinguishability by multiple samples.[10] The proof provides a simple demonstration of a central proof technique, known as the *hybrid technique.*

Proof Sketch:[11] Again, the proof uses the counter-positive, which in such settings is called a reducibility argument (see Section 7.1.2 onwards). Specifically, we show that the existence of an efficient algorithm that distinguishes the ensembles X and Y using several samples, implies the existence of an efficient algorithm that distinguishes the ensembles X and Y using a single sample. The implication is proven using the following argument, which will be latter called a "hybrid argument".

To prove that a sequence of $s(n)$ samples drawn independently from X_n is indistinguishable from a sequence of $s(n)$ samples drawn independently from Y_n, we consider *hybrid* sequences such that the $i^{\rm th}$ hybrid consists of i samples of X_n followed by $s(n)-i$ samples of Y_n. The "homogeneous" sequences (which we wish

[10]The requirement that both ensembles are polynomial-time constructible is essential; see, Exercise 8.10.

[11]For more details see [90, Sec. 3.2.3].

to prove to be computational indistinguishable) are the extreme hybrids (i.e., the first and last hybrids). The key observation is that distinguishing the extreme hybrids (towards the contradiction hypothesis) implies distinguishing neighboring hybrids, which in turn yields a procedure for distinguishing single samples of the two original distributions (contradicting the hypothesis that these two distributions are indistinguishable by a single sample). Details follow.

Suppose, towards the contradiction, that D distinguishes $s(n)$ samples of X_n from $s(n)$ samples of Y_n, with a distinguishing gap of $\delta(n)$. Denoting the i^{th} hybrid by H_n^i (i.e., $H_n^i = (X_n^{(1)}, ..., X_n^{(i)}, Y_n^{(i+1)}, ..., Y_n^{(s(n))})$), this means that D distinguishes the extreme hybrids (i.e., H_n^0 and $H_n^{s(n)}$) with gap $\delta(n)$. It follows that D distinguishes a random pair of neighboring hybrids (i.e., D distinguishes H_n^i from H_n^{i+1}, for a randomly selected i) with gap at least $\delta(n)/s(n)$: the reason being that

$$\begin{aligned}
&\mathsf{E}_{i\in\{0,\dots,s(n)-1\}}\left[\mathsf{Pr}[D(H_n^i)=1]-\mathsf{Pr}[D(H_n^{i+1})=1]\right] \\
&= \frac{1}{s(n)}\cdot\sum_{i=0}^{s(n)-1}\left(\mathsf{Pr}[D(H_n^i)=1]-\mathsf{Pr}[D(H_n^{i+1})=1]\right) \qquad (8.7)\\
&= \frac{1}{s(n)}\cdot\left(\mathsf{Pr}[D(H_n^0)=1]-\mathsf{Pr}[D(H_n^{s(n)})=1]\right) \;=\; \frac{\delta(n)}{s(n)}\,.
\end{aligned}$$

The key step in the argument is transforming the distinguishability of neighboring hybrids into distinguishability of single samples of the original ensembles (thus deriving a contradiction). Indeed, using D, we obtain a distinguisher D' of single samples: Given a single sample, algorithm D' selects $i \in \{0, ..., s(n) - 1\}$ at random, generates i samples from the first distribution and $s(n) - i - 1$ samples from the second distribution, invokes D with the $s(n)$-samples sequence obtained when placing the input sample in location $i + 1$, and answers whatever D does. That is, on input z and when selecting the index i, algorithm D' invokes D on a sample from the distribution $(X_n^{(1)}, ..., X_n^{(i)}, z, Y_n^{(i+2)}, ..., Y_n^{(s(n))})$. Thus, the construction of D' relies on the hypothesis that both probability ensembles are polynomial-time constructible. The analysis of D' is based on the following two facts:

1. When invoked on an input that is distributed according to X_n and selecting the index $i \in \{0, ..., s(n) - 1\}$, algorithm D' behaves like $D(H_n^{i+1})$, because $(X_n^{(1)}, ..., X_n^{(i)}, X_n, Y_n^{(i+2)}, ..., Y_n^{(s(n))}) \equiv H_n^{i+1}$.

2. When invoked on an input that is distributed according to Y_n and selecting the index $i \in \{0, ..., s(n) - 1\}$, algorithm D' behaves like $D(H_n^i)$, because $(X_n^{(1)}, ..., X_n^{(i)}, Y_n, Y_n^{(i+2)}, ..., Y_n^{(s(n))}) \equiv H_n^i$.

Thus, the distinguishing gap of D' (between Y_n and X_n) is captured by Eq. (8.7), and the claim follows (because assuming towards the contradiction that the proposition's conclusion does not hold leads to a contradiction of the proposition's hypothesis). □

The hybrid technique – a digest: The hybrid technique constitutes a special type of a "reducibility argument" in which the computational indistinguishability of *complex* ensembles is proved using the computational indistinguishability of *basic* ensembles. The actual reduction is in the other direction: efficiently distinguishing the basic ensembles is reduced to efficiently distinguishing the complex ensembles, and *hybrid* distributions are used in the reduction in an essential way. The following three properties of the construction of the hybrids play an important role in the argument:

1. *The complex ensembles collide with the extreme hybrids.* This property is essential because our aim is proving something that relates to the complex ensembles (i.e., their indistinguishability), while the argument itself refers to the extreme hybrids.

   In the proof of Proposition 8.6 the extreme hybrids (i.e., $H_n^{s(n)}$ and H_n^0) collide with the complex ensembles that represent $s(n)$-ary sequences of samples of one of the basic ensembles.

2. *The basic ensemble are efficiently mapped to neighboring hybrids.* This property is essential because our starting hypothesis relates to the basic ensembles (i.e., their indistinguishability), while the argument itself refers directly to the neighboring hybrids. Thus, we need to translate our knowledge (i.e., computational indistinguishability) of the basic ensembles to knowledge (i.e., computational indistinguishability) of any pair of neighboring hybrids. Typically, this is done by efficiently transforming strings in the range of a basic distribution into strings in the range of a hybrid such that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighboring hybrid.

   In the proof of Proposition 8.6 the basic ensembles (i.e., X_n and Y_n) were efficiently transformed into neighboring hybrids (i.e., H_n^{i+1} and H_n^i, respectively). Recall that, in this case, the efficiency of this transformation relied on the hypothesis that both the basic ensembles are polynomial-time constructible.

3. *The number of hybrids is small* (i.e., polynomial). This property is essential in order to deduce the computational indistinguishability of extreme hybrids from the computational indistinguishability of each pair of neighboring hybrids. Typically, the "distinguishability gap" established in the argument losses a factor that is proportional to the number of hybrids. This is due to the fact that the gap between the extreme hybrids is upper-bounded by the sum of the gaps between neighboring hybrids.

   In the proof of Proposition 8.6 the number of hybrids equals $s(n)$ and the aforementioned loss is reflected in Eq. (8.7).

We remark that in the course of an hybrid argument, a distinguishing algorithm referring to the complex ensembles is being analyzed and even invoked on arbitrary hybrids. The reader may be annoyed of the fact that the algorithm "was not designed to work on such hybrids" (but rather only on the extreme hybrids). However, *an algorithm is an algorithm*: once it exists we can invoke it on inputs of our choice, and analyze its performance on arbitrary input distributions.

8.2.4 Amplifying the stretch function

Recall that the definition of pseudorandom generators (i.e., Definition 8.1) makes a minimal requirement regarding their stretch; that is, it is only required that the length of the output of such generators is longer than their input. Needless to say, we seek pseudorandom generators with a much more significant stretch, firstly because the stretch determines the saving in randomness obtained via Construction 8.2. It turns out (see Construction 8.7) that pseudorandom generators of any stretch function (and in particular of minimal stretch $\ell_1(k) \stackrel{\text{def}}{=} k+1$) can be easily converted into pseudorandom generators of any desired (polynomially bounded) stretch function, ℓ. (On the other hand, since pseudorandom generators are required (in Definition 8.1) to run in polynomial time, their stretch must be polynomially bounded.)

Construction 8.7 *Let G_1 be a pseudorandom generator with stretch function $\ell_1(k) = k+1$, and ℓ be any polynomially bounded stretch function that is polynomial-time computable. Let*

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)} \tag{8.8}$$

where $x_0 = s$ and $x_i\sigma_i = G_1(x_{i-1})$, for $i = 1, ..., \ell(|s|)$. (That is, σ_i is the last bit of $G_1(x_{i-1})$ and x_i is the $|s|$-bit long prefix of $G_1(x_{i-1})$.)

Needless to say, G is polynomial-time computable and has stretch ℓ. An alternative construction is considered in Exercise 8.11.

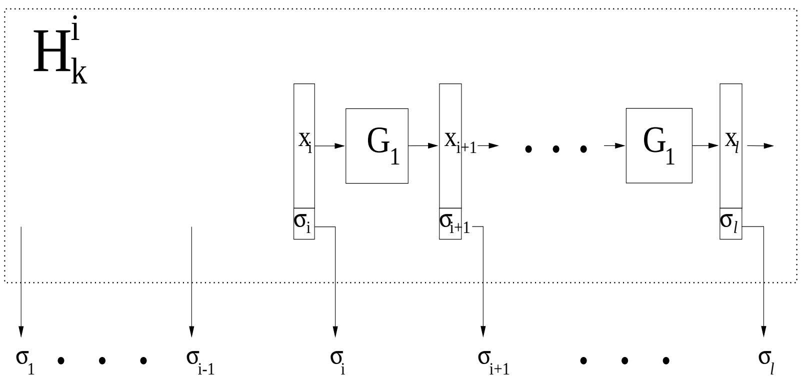


Figure 8.2: Analysis of stretch amplification – the i^{th} hybrid.

Proposition 8.8 *Let G_1 and G be as in Construction 8.7. Then G constitutes a pseudorandom generator.*

Proof Sketch:[12] The proposition is proven using the *hybrid technique*, presented and discussed in Section 8.2.3. Here (for $i = 0, ..., \ell(k)$) we consider the hybrid distributions H_k^i, depicted in Figure 8.2 and defined by

$$H_k^i \stackrel{\text{def}}{=} U_i^{(1)} \cdot g_{\ell(k)-i}(U_k^{(2)}),$$

where $\cdot$ denotes the concatenation of strings, $g_j(x)$ denotes the j-bit long prefix of $G(x)$, and $U_i^{(1)}$ and $U_k^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^k$, respectively). The extreme hybrids (i.e., H_k^0 and H_k^k) correspond to $G(U_k)$ and $U_{\ell(k)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G_1(U_k)$ and U_{k+1}. Details follow.

We shall focus on proving the indistinguishability of neighboring hybrids.[13] Suppose, towards the contradiction, that algorithm D distinguishes H_k^i from H_k^{i+1}. We first take a closer look at these hybrids. Note that, for $j \geq 1$, it holds that $g_j(s) \equiv (\sigma, g_{j-1}(x))$, where $x\sigma = G_1(s)$. Denoting the first $|x|-1$ bits of x by $F(x)$ and the last bit of x by $L(x)$, we may write $g_j(s) \equiv (L(G_1(s)), g_{j-1}(F(G_1(s))))$ and $(U_1^{(1)}, U_k^{(2)}) \equiv (L(U_{k+1}), F(U_{k+1}))$. It follows that

$$\begin{aligned}
H_k^i &= U_i^{(1)} \cdot g_{\ell(k)-i}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(G_1(U_k^{(2)})), g_{(\ell(k)-i)-1}(F(G_1(U_k^{(2)})))) \\
H_k^{i+1} &= U_{i+1}^{(1')} \cdot g_{\ell(k)-i-1}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(U_{k+1}^{(2')}), g_{(\ell(k)-i)-1}(F(U_{k+1}^{(2')}))).
\end{aligned}$$

Now, combining the generation of $U_i^{(1)}$ and the evaluation of $g_{\ell(k)-i-1}$ with the distinguisher D, we distinguish the distribution $(F(G_1(U_k^{(2)})), L(G_1(U_k^{(2)}))) \equiv G_1(U_k)$ from the distribution $(F(U_{k+1}^{(2')}), L(U_{k+1}^{(2')})) \equiv U_{k+1}$, in contradiction to the pseudorandomness of G_1. Specifically, on input $x \in \{0,1\}^{k+1}$, we uniformly select $r \in \{0,1\}^i$ and output $D(r \cdot L(x) \cdot g_{\ell(k)-i-1}(F(x)))$. The analysis of the resulting distinguisher is based on the following two facts:

1. When given an input that is distributed according to $G_1(U_k)$, we invoke algorithm D on input $(U_i', L(G_1(U_k)), g_{\ell(k)-i-1}(F(G_1(U_k)))) \equiv H_k^i$.

2. When given an input that is distributed according to U_{k+1}, we invoke algorithm D on input $(U_i', L(U_{k+1}), g_{\ell(k)-i-1}(F(U_{k+1}))) \equiv H_k^{i+1}$.

Thus, the probability that we output 1 on input $G_1(U_k)$ (resp., U_{k+1}) equals $\Pr[D(H_k^i) = 1]$ (resp., $\Pr[D(H_k^{i+1}) = 1]$). Hence the distinguishability of neighboring hybrids implies the distinguishability of $G_1(U_k)$ and U_{k+1}. □

[12] For more details see [90, Sec. 3.3.3].

[13] As usual (when the hybrid technique is used), the distinguishability of the extreme hybrids (which collide with $G(U_k)$ and $U_{\ell(k)}$, respectively) implies the distinguishability of a random pair of neighboring hybrids. Thus, the following analysis will be applied to a random i (in $\{0, ..., k-1\}$), and the full analysis will refer to an expression analogous to Eq. (8.7).

Conclusion. In view of the foregoing, when talking about the mere existence of pseudorandom generators, in the sense of Definition 8.1, we may ignore the specific stretch function.

8.2.5 Constructions

The constructions surveyed in this section "transform" computational difficulty, in the form of one-way functions, into generators of pseudorandomness. Recall that a *polynomial-time computable function* is called one-way if any efficient algorithm can invert it only with negligible success probability (see Definition 7.1 and Section 7.1 for further discussion). We will actually use hard-core predicates of such functions, and refer the reader to their treatment in Section 7.1.3. Loosely speaking, a *polynomial-time computable* predicate b is called a hard-core of a function f if any efficient algorithm, given $f(x)$, can guess $b(x)$ with success probability that is only negligibly higher than half. Recall that (by Theorem 7.7), for any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x,r) = (f(x), r)$.

8.2.5.1 A simple construction

Intuitively, the definition of a hard-core predicate implies a potentially interesting case of computational indistinguishability. Specifically, as will be shown implicitly in Proposition 8.9 and explicitly in Exercise 8.8, if b is a hard-core of the function f, then the ensemble $\{f(U_n)\cdot b(U_n)\}_{n\in\mathbb{N}}$ is computationally indistinguishable from the ensemble $\{f(U_n)\cdot U_1'\}_{n\in\mathbb{N}}$. Furthermore, if f is 1-1 then the foregoing ensembles are statistically far apart, and thus constitute a non-trivial case of computational indistinguishability. If f is also polynomial-time computable and length-preserving, then this yields a construction of a pseudorandom generator.

Proposition 8.9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 and length-preserving function f. Then, $G(s) \stackrel{\rm def}{=} f(s)\cdot b(s)$ is a pseudorandom generator.*

Proof Sketch:[14] Considering a uniformly distributed $s \in \{0,1\}^n$, we first note that the n-bit long prefix of $G(s)$ is uniformly distributed in $\{0,1\}^n$, because f induces a permutation on the set $\{0,1\}^n$. Hence, the proof boils down to showing that distinguishing $f(s)\cdot b(s)$ from $f(s)\cdot\sigma$, where σ is a random bit, yields contradiction to the hypothesis that b is a hard-core of f (i.e., that $b(s)$ is *unpredictable* from $f(s)$). Intuitively, the reason is that such a hypothetical distinguisher also distinguishes $f(s)\cdot b(s)$ from $f(s)\cdot\overline{b(s)}$, where $\overline{\sigma} = 1-\sigma$, whereas distinguishing $f(s)\cdot b(s)$ from $f(s)\cdot\overline{b(s)}$ yields an algorithm for predicting $b(s)$ based on $f(s)$. Details follow. We start with any potential distinguisher D, and let

$$\delta(k) \stackrel{\rm def}{=} \Pr[D(G(U_k))=1] - \Pr[D(U_{k+1})=1].$$

We may assume, without loss of generality, that $\delta(k)$ is non-negative (for infinitely many k's). Observing that $G(U_k) = f(U_k)\cdot b(U_k)$ and that U_{k+1} is distributed

[14] For more details see [90, Sec. 3.3.4].

identically to a random variable that equals $f(U_k)b(U_k)$ with probability 1/2 and $f(U_k)\overline{b(U_k)}$ otherwise, we have

$$\Pr[D(f(U_k)b(U_k)) = 1] - \Pr[D(f(U_k)\overline{b(U_k)}) = 1] \;=\; 2\delta(k).$$

The key observation is that D effectively distinguishes (with gap $2\delta(k)$) the case that the last bit is $b(U_k)$ from the case that the last bit is $\overline{b(U_k)}$. This distinguishing ability can be transformed to predicting the value of $b(U_k)$, when given the value $f(U_k)$. Indeed, consider an algorithm A that, on input y, uniformly selects $\sigma \in \{0,1\}$, invokes $D(y\sigma)$, and outputs σ if $D(y\sigma) = 1$ and $\overline{\sigma}$ otherwise. Then

$$\begin{aligned}
&\Pr[A(f(U_k)) = b(U_k)] \\
&\quad= \Pr[D(f(U_k)\cdot\sigma) = 1 \wedge \sigma = b(U_k)] \;+\; \Pr[D(f(U_k)\cdot\sigma) = 0 \wedge \sigma = \overline{b(U_k)}] \\
&\quad= \frac{1}{2}\cdot\Big(\Pr[D(f(U_k)\cdot b(U_k)) = 1] \;+\; \Big(1 \;-\; \Pr[D(f(U_k)\cdot\overline{b(U_k)}) = 1]\Big)\Big)
\end{aligned}$$

which equals $(1 + 2\delta(k))/2$. This contradicts the hypothesis that b is a hard-core of f, and the proposition follows. □

Combining Theorem 7.7, Proposition 8.9 and Construction 8.7, we obtain the following corollary.

Theorem 8.10 (A sufficient condition for the existence of pseudorandom generators): *If there exists 1-1 and length-preserving one-way function then, for every polynomially bounded stretch function ℓ, there exists a pseudorandom generator of stretch ℓ.*

Digest. The main part of the proof of Proposition 8.9 is showing that the (next bit) unpredictability of $G(U_k)$ implies the pseudorandomness of $G(U_k)$. The fact that (next bit) unpredictability and pseudorandomness are equivalent, in general, is proven explicitly in the alternative proof of Theorem 8.10 provided next.

8.2.5.2 An alternative presentation

Let us take a closer look at the pseudorandom generators obtained by combining Construction 8.7 and Proposition 8.9. For a stretch function $\ell : \mathbb{N} \to \mathbb{N}$, a 1-1 one-way function f with a hard-core b, we obtain

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)}\,, \tag{8.9}$$

where $x_0 = s$ and $x_i\sigma_i = f(x_{i-1})b(x_{i-1})$ for $i = 1, \ldots, \ell(|s|)$. Denoting by $f^i(x)$ the value of f iterated i times on x (i.e., $f^i(x) = f^{i-1}(f(x))$ and $f^0(x) = x$), we rewrite Eq. (8.9) as follows

$$G(s) \stackrel{\text{def}}{=} b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))\,. \tag{8.10}$$

The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a (random)[15] prefix of Z_k fails to predict the next bit of Z_k with probability non-negligibly higher than $1/2$. Specifically, we establish the following two results.

1. A **general result** asserting that *an ensemble is pseudorandom if and only if it is unpredictable.* Recall that an ensemble is pseudorandom if it is computationally indistinguishable from a uniform distribution (over bit strings of adequate length).

   Clearly, pseudorandomness implies polynomial-time unpredictability, but here we actually need the other direction, which is less obvious. Still, using a hybrid argument, one can show that (next-bit) unpredictability implies indistinguishability from the uniform ensemble. For details see Exercise 8.12.

2. A **specific result** asserting that the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is unpredictable *from right to left.* Equivalently, $G'(U_n)$ is polynomial-time unpredictable (from left to right (as usual)), where $G'(s) = b(f^{\ell(|s|)-1}(s))\cdots b(f(s))\cdot b(s)$ is the reverse of $G(s)$.

   Using the fact that f induces a permutation over $\{0,1\}^n$, observe that the $(j+1)$-bit long prefix of $G'(U_k)$ is distributed identically to $b(f^j(U_k))\cdots b(f(U_k))\cdot b(U_k)$. Thus, an algorithm that predicts the $j+1^{\text{st}}$ bit of $G'(U_n)$ based on the j-bit long prefix of $G'(U_n)$ yields an algorithm that guesses $b(U_n)$ based on $f(U_n)$. For details see Exercise 8.14.

Needless to say, G is a pseudorandom generator if and only if G' is a pseudorandom generator (see Exercise 8.13). We mention that Eq. (8.10) is often referred to as the Blum-Micali Construction.[16]

8.2.5.3 A general condition for the existence of pseudorandom generators

Recall that given any one-way 1-1 length-preserving function, we can easily construct a pseudorandom generator. Actually, the 1-1 (and length-preserving) requirement may be dropped, but the currently known construction – for the general case – is quite complex.

Theorem 8.11 (On the existence of pseudorandom generators): *Pseudorandom generators exist if and only if one-way functions exist.*

To show that the existence of pseudorandom generators imply the existence of one-way functions, consider a pseudorandom generator G with stretch function

[15] For simplicity, we define unpredictability as referring to prefixes of a random length (distributed uniformly in $\{0,...,|Z_k|-1\}$). A more general definition allows the predictor to determine the length of the prefix that it reads on the fly. This seemingly stronger notion of unpredictability is actually equivalent to the one we use, because both notions are equivalent to pseudorandomness.

[16] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. Indeed, this construction originates in [39].

$\ell(k) = 2k$. For $x, y \in \{0,1\}^k$, define $f(x, y) \stackrel{\rm def}{=} G(x)$, and so f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_k)$ from U_{2k} by trying to invert and checking the result: inverting f on the distribution $f(U_{2k})$ corresponds to operating on the distribution $G(U_k)$, whereas the probability that U_{2k} has inverse under f is negligible.

The interesting direction of the proof of Theorem 8.11 is the construction of pseudorandom generators based on any one-way function. Since the known proof is quite complex, we only provide a very rough overview of some of the ideas involved. We mention that these ideas make extensive use of adequate hashing functions (e.g., pairwise independent hashing functions, see Appendix D.2).

We first note that, in general (when f may not be 1-1), the ensemble $f(U_k)$ may not be pseudorandom, and so Construction 8.9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea underlying the known construction is hashing $f(U_k)$ to an almost uniform string of length related to its entropy, using adequate hashing functions.[17] But "hashing $f(U_k)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $k' < k$. This foils the entire point of stretching the k-bit seed. Thus, a second idea underlying the construction is compensating for the loss of $k - k'$ bits by extracting these many bits from the seed U_k itself. This is done by hashing U_k, and the point is that the $(k - k')$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

8.2.6 Non-uniformly strong pseudorandom generators

Recall that we said that truly random sequences can be replaced by pseudorandom sequences without affecting any efficient computation that uses these sequences. The specific formulation of this assertion, presented in Proposition 8.3, refers to randomized algorithms that take a "primary input" and use a secondary "random input" in their computation. Proposition 8.3 asserts that it is infeasible to find a primary input for which the replacement of a truly random secondary input by a pseudorandom one affects the final output of the algorithm in a noticeable way. This, however, does not mean that such primary inputs do not exist (but rather that they are hard to find). Consequently, Proposition 8.3 falls short of yielding a (worst-case)[18] "derandomization" of a complexity class such as $\mathcal{BPP}$.

[17] This is done after guaranteeing that the logarithm of the probability mass of a value of $f(U_k)$ is typically close to the entropy of $f(U_k)$. Specifically, given an arbitrary one-way function f', one first constructs f by taking a "direct product" of sufficiently many copies of f'. For example, for $x_1, ..., x_{k^{2/3}} \in \{0,1\}^{k^{1/3}}$, we let $f(x_1, ..., x_{k^{2/3}}) \stackrel{\rm def}{=} f'(x_1), ..., f'(x_{k^{2/3}})$.

[18] Indeed, Proposition 8.3 yields an *average-case derandomization of* $\mathcal{BPP}$. In particular, for every polynomial-time constructible ensemble $\{X_n\}_{n\in\mathbb{N}}$, every Boolean function $f \in \mathcal{BPP}$, and every $\varepsilon > 0$, there exists a randomized algorithm A' of randomness complexity $r_\varepsilon(n) = n^\varepsilon$ such that the probability that $A'(X_n) \neq f(X_n)$ is negligible. A corresponding deterministic ($\exp(r_\varepsilon)$-time) algorithm A'' can be obtained, as in the proof of Theorem 8.13, and again the probability that $A''(X_n) \neq f(X_n)$ is negligible, where here the probability is taken only over the distribution of the primary input (represented by X_n). In contrast, worst-case derandomization, as captured by the assertion $\mathcal{BPP} \subseteq \mathrm{DTIME}(2^{r_\varepsilon})$, requires that the probability that $A''(X_n) \neq f(X_n)$ is zero.

To obtain such results, we need a stronger notion of pseudorandom generators, presented next. Specifically, we need pseudorandom generators that can fool all polynomial-size circuits (cf. §1.2.4.1), and not merely all probabilistic polynomial-time algorithms.[19]

Definition 8.12 (strong pseudorandom generator – fooling circuits): *A deterministic polynomial-time algorithm G is called a* non-uniformly strong pseudorandom generator *if there exists a* stretch function, $\ell : \mathbb{N} \to \mathbb{N}$, *such that for any family $\{C_k\}_{k\in\mathbb{N}}$ of polynomial-size circuits, for any positive polynomial p, and for all sufficiently large k's*

$$|\,\mathsf{Pr}[C_k(G(U_k)) = 1] \;-\; \mathsf{Pr}[C_k(U_{\ell(k)}) = 1]\,| \;<\; \frac{1}{p(k)}$$

An alternative formulation is obtained by referring to polynomial-time machines that take advice (Section 3.1.2). Using such pseudorandom generators, we can "derandomize" $\mathcal{BPP}$.

Theorem 8.13 (derandomization of $\mathcal{BPP}$): *If there exists non-uniformly strong pseudorandom generators then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\mathrm{def}}{=} 2^{n^\varepsilon}$.*

Proof Sketch: For any $S \in \mathcal{BPP}$ and any $\varepsilon > 0$, we let A denote the decision procedure for S and G denote a non-uniformly strong pseudorandom generator stretching n^ε-bit long seeds into $\mathrm{poly}(n)$-long sequences (to be used by A as secondary input when processing a primary input of length n). Combining A and G, we obtain an algorithm $A' = A_G$ (as in Construction 8.2). We claim that *A and A' may significantly differ in their* (expected probabilistic) *decision on at most finitely many inputs*, because otherwise we can use these inputs (together with A) to derive a (non-uniform) family of polynomial-size circuits that distinguishes $G(U_{n^\varepsilon})$ and $U_{\mathrm{poly}(n)}$, contradicting the the hypothesis regarding G. Specifically, an input x on which A and A' differ significantly yields a circuit C_x that distinguishes $G(U_{|x|^\varepsilon})$ and $U_{\mathrm{poly}(|x|)}$, by letting $C_x(r) = A(x, r)$.[20] Incorporating the finitely many "bad" inputs into A', we derive a probabilistic polynomial-time algorithm that decides S while using randomness complexity n^ε.

Finally, emulating A' on each of the 2^{n^ε} possible random sequences (i.e., seeds to G) and ruling by majority, we obtain a deterministic algorithm A'' as required. That is, let $A'(x, r)$ denote the output of algorithm A' on input x when using coins $r \in \{0,1\}^{n^\varepsilon}$. Then $A''(x)$ invokes $A'(x, r)$ on every $r \in \{0,1\}^{n^\varepsilon}$, and outputs 1 if and only if the majority of these 2^{n^ε} invocations have returned 1. □

[19] Needless to say, strong pseudorandom generators in the sense of Definition 8.12 satisfy the basic definition of a pseudorandom generator (i.e., Definition 8.1); see Exercise 8.15. We comment that the underlying notion of computational indistinguishability (by circuits) is strictly stronger than Definition 8.4, and that it is invariant under multiple samples (regardless of the constructibility of the underlying ensembles); for details, see Exercise 8.16.

[20] Indeed, in terms of the proof of Proposition 8.3, the finder F consists of a non-uniform family of polynomial-size circuits that print the "problematic" primary inputs that are hard-wired in them, and the corresponding distinguisher D is thus also non-uniform.

We comment that stronger results regarding derandomization of $\mathcal{BPP}$ are presented in Section 8.3.

On constructing non-uniformly strong pseudorandom generators. Non-uniformly strong pseudorandom generators (as in Definition 8.12) can be constructed using any one-way function that is hard to invert by any non-uniform family of polynomial-size circuits (as in Definition 7.3), rather than by probabilistic polynomial-time machines. In fact, the construction in this case is simpler than the one employed in the uniform case (i.e., the construction underlying the proof of Theorem 8.11).

8.2.7 Stronger notions and conceptual reflections

We first mention two stronger variants on the definition of pseudorandom generators, and conclude this section by highlighting various conceptual issues.

8.2.7.1 Stronger (uniform-complexity) notions

The following two notions represent strengthening of the standard definition of pseudorandom generators (as presented in Definition 8.1). Non-uniform versions of these notions (strengthening Definition 8.12) are also of interest.

Fooling stronger distinguishers. One strengthening of Definition 8.1 amounts to explicitly quantifying the resources (and success gaps) of distinguishers. We choose to bound these quantities as a function of the length of the seed (i.e., k), rather than as a function of the length of the string that is being examined (i.e., $\ell(k)$). For a class of time bounds $\mathcal{T}$ (e.g., $\mathcal{T} = \{t(k) \stackrel{\text{def}}{=} 2^{c\sqrt{k}}\}_{c\in\mathbb{N}}$) and a class of noticeable functions (e.g., $\mathcal{F} = \{f(k) \stackrel{\text{def}}{=} 1/t(k) : t \in \mathcal{T}\}$), we say that a pseudorandom generator, G, is $(\mathcal{T},\mathcal{F})$-strong if for any probabilistic algorithm D having running-time bounded by a function in $\mathcal{T}$ (applied to k)[21], for any function f in $\mathcal{F}$, and for all sufficiently large k's, it holds that

$$|\Pr[D(G(U_k)) = 1] - \Pr[D(U_{\ell(k)}) = 1]| < f(k).$$

An analogous strengthening may be applied to the definition of one-way functions. Doing so reveals the weakness of the known construction that underlies the proof of Theorem 8.11: It only implies that for some $\varepsilon > 0$ ($\varepsilon = 1/8$ will do), for any $\mathcal{T}$ and $\mathcal{F}$, the existence of "$(\mathcal{T},\mathcal{F})$-strong one-way functions" implies the existence of $(\mathcal{T}',\mathcal{F}')$-strong pseudorandom generators, where $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k^\varepsilon)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(k^\varepsilon) : f \in \mathcal{F}\}$. What we *would like* to have is an analogous result with $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(\Omega(k))/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(\Omega(k)) : f \in \mathcal{F}\}$.

[21] That is, when examining a sequence of length $\ell(k)$ algorithm D makes at most $t(k)$ steps, where $t \in \mathcal{T}$.

Pseudorandom Functions. Recall that pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined in Appendix C.3.3) are even more powerful: They allow *efficient direct access* to a huge pseudorandom sequence, which is not even feasible to scan bit-by-bit. Specifically, based on a (random) k-bit long seed, they allow direct access to a sequence of length 2^k. Put in other words, pseudorandom functions are deterministic polynomial-time algorithms that map a k-bit long seed s and a k-bit long argument x to a value $f_s(x)$ such that, for a uniformly distributed $s \in \{0,1\}^k$, the function f_s looks random to any poly(k)-time observer that may query f_s at arguments of its choice. Thus, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). We mention that pseudorandom functions can be constructed from any pseudorandom generator (see Theorem C.8), and that they found many applications in cryptography (see Appendices C.3.3, C.5.2, and C.6.2). Pseudorandom functions were also used to derive negative results in computational learning theory [230] and in the study of circuit complexity (cf., Natural Proofs [188]).

8.2.7.2 Conceptual reflections

We highlight several conceptual aspects of the foregoing computational approach to randomness. Some of these aspects are common to other instantiation of the general paradigm (esp., the one presented in Section 8.3).

Behavioristic versus Ontological. The behavioristic nature of the computational approach to randomness is best demonstrated by confronting this approach with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string that does not have a substantially simpler (i.e., shorter) explanation than itself. Considering the simplest explanation of a phenomenon may be viewed as an ontological approach. In contrast, considering the effect of phenomena on certain devices (or observations), as underlying the definition of pseudorandomness, is a behavioristic approach. Furthermore, there exist probability distributions that are not uniform (and are not even statistically close to a uniform distribution) and nevertheless are indistinguishable from a uniform distribution (by any efficient device). Thus, *distributions that are ontologically very different, are considered equivalent by the behavioristic point of view taken in the definition of computational indistinguishability.*

A relativistic view of randomness. We have defined pseudorandomness in terms of its observer. Specifically, we have considered the class of efficient (i.e., polynomial-time) observers and defined as pseudorandom objects that look random to any observer in that class. In subsequent sections, we shall consider restricted classes of such observers (e.g., space-bounded polynomial-time observers and even very restricted observers that merely apply specific tests such as linear tests or hitting tests). Each such class of observers gives rise to a different notion of pseudorandomness. Furthermore, the general paradigm (of pseudorandomness) explicitly aims at *distributions that are not uniform and yet are considered as such from the point of view of certain observers.* Thus, our entire approach to pseudorandomness is relativistic and subjective (i.e., depending on the abilities of the observer).

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The general paradigm of pseudorandomness relies on the fact that *placing computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform distributions.* Thus, the pivot of the entire approach is the computational difficulty of distinguishing pseudorandom distributions from truly random ones. Furthermore, many of the constructions of pseudorandom generators rely either on conjectures or on facts regarding computational difficulty (i.e., that certain computations that are hard for certain classes). For example, one-way functions were used to construct general-purpose pseudorandom generators (i.e., those working in polynomial-time and fooling all polynomial-time observers). Analogously, as we shall see in §8.3.3.1, the fact that parity function is hard for polynomial-size constant-depth circuits can be used to generate (highly non-uniform) sequences that fool such circuits.

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions (cf. Sections 8.2.5 and 8.3.2). We wish to highlight the intuitive appeal of this connection.

8.3 Derandomization of time-complexity classes

Let us take a second look at the process of derandomization that underlies the proof of Theorem 8.13. First, a pseudorandom generator was used to shrink the randomness-complexity of a BPP-algorithm, and then derandomization was achieved by scanning all possible seeds to this generator. A key observation regarding this process is that there is no point in insisting that the pseudorandom generator runs in time that is polynomial in its seed length. Instead, it suffices to require that the generator runs in time that is exponential in its seed length, because we are incurring such an overhead anyhow due to the scanning of all possible seeds. Furthermore, in this context, the running-time of the generator may be larger than the running time of the algorithm, which means that the generator need only fool distinguishers that take less steps than the generator. These considerations motivate the following definition of canonical derandomizers.

8.3.1 Defining canonical derandomizers

Recall that in order to "derandomize" a probabilistic polynomial-time algorithm A, we first obtain a functionally equivalent algorithm A_G (as in Construction 8.2) that has (significantly) smaller randomness-complexity. Algorithm A_G has to maintain A's input-output behavior on all (but finitely many) inputs. Thus, the set of the relevant distinguishers (considered in the proof of Theorem 8.13) is the set of all possible circuits obtained from A by hard-wiring any of the possible inputs. Such a circuit, denoted C_x, emulates the execution of algorithm A on input x, when using the circuit's input as the algorithm's internal coin tosses (i.e., $C_x(r) = A(x,r)$). Furthermore, the size of C_x is quadratic in the running-time of A on input x, and the length of the input to C_x equals the running-time of A (on input x).[22] Thus, the size of C_x is quadratic in the length of its own input, and the pseudorandom generator in use (i.e., G) needs to fool each such circuit. Recalling that we may allow the generator to run in exponential-time (i.e., time that is exponential in the length of its own input (i.e., the seed))[23], we arrive at the following definition.

Definition 8.14 (pseudorandom generator for derandomizing $\mathrm{BP}\textsc{time}(\cdot)$)[24]: *Let $\ell : \mathbb{N} \to \mathbb{N}$ be a monotonically increasing function. A* canonical derandomizer of stretch *ℓ is a deterministic algorithm G that satisfies the following two conditions.*

1. *On input a k-bit long seed, G makes at most* $\mathrm{poly}(2^k \cdot \ell(k))$ *steps and outputs a string of length $\ell(k)$.*

2. *For every circuit D_k of size $\ell(k)^2$ it holds that*

$$|\Pr[D_k(G(U_k)) = 1] - \Pr[D_k(U_{\ell(k)}) = 1]| < \frac{1}{6}. \qquad (8.11)$$

The circuit D_k represents a potential distinguisher, which is given an $\ell(k)$-bit long string (sampled either from $G(U_k)$ or from $U_{\ell(k)}$). When seeking to derandomize

[22]Indeed, we assume that algorithm A is represented as a Turing machine and refer to the standard emulation of Turing machines by circuits (as underlying the proof of Theorem 2.21). Thus, the aforementioned circuit C_x has size that is at most quadratic in the running-time of A on input x, which in turn means that C_x has size that is at most quadratic in the length of its own input. (In fact, the circuit size can be made almost-linear in the running-time of A, by using a better emulation [179].) We note that many sources use the fictitious convention by which the circuit size equals the length of its input; this fictitious convention can be justified by considering a (suitably) padded input.

[23]Actually, in Definition 8.14 we allow the generator to run in time $\mathrm{poly}(2^k\ell(k))$, rather than in time $\mathrm{poly}(2^k)$. This is done in order not to trivially rule out generators of super-exponential stretch (i.e., $\ell(k) = 2^{\omega(k)}$). However (see Exercise 8.18), the condition in Eq. (8.11) does not allow for super-exponential stretch (or even for $\ell(k) = \omega(2^k)$). Thus, in retrospect, the two formulations are equivalent (because $\mathrm{poly}(2^k\ell(k)) = \mathrm{poly}(2^k)$ for $\ell(k) = 2^{O(k)}$).

[24]Fixing a model of computation, we denote by $\mathrm{BP}\textsc{time}(t)$ the class of decision problems that are solvable by a randomized algorithm of time complexity t that has two-sided error $1/3$. Using $1/6$ as the "threshold distinguishing gap" (in Eq. (8.11)) guarantees that if $\Pr[D_k(U_{\ell(k)}) = 1] \geq 2/3$ (resp., $\Pr[D_k(U_{\ell(k)}) = 1] \leq 1/3$) then $\Pr[D_k(G(U_k)) = 1] > 1/2$ (resp., $\Pr[D_k(G(U_k)) = 1] < 1/2$). As we shall see, this suffices for a derandomization of $\mathrm{BP}\textsc{time}(t)$ in time T, where $T(n) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$ (and we use a seek of length $k = \ell^{-1}(t(n))$).

an algorithm A of time-complexity t, the aforementioned $\ell(k)$-bit long string represents a possible sequence of coin tosses of A, when invoked on a generic (primary) input of length $n = t^{-1}(\ell(k))$. Thus, for any $x \in \{0,1\}^n$, considering the circuit $D_k(r) = A(x,r)$, where $|r| = t(n) = \ell(k)$, we note that Eq. (8.11) implies that $A_G(x) = A(x, G(U_k))$ *maintains the majority vote of* $A(x) = A(x, U_{\ell(k)})$. On the other hand, the time-complexity of G implies that the straightforward deterministic emulation of $A_G(x)$ takes time $2^k \cdot (\text{poly}(2^k \cdot \ell(k)) + t(n))$, which is upper-bounded by $\text{poly}(2^k \cdot \ell(k)) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. This yields the following (conditional) derandomization result.

Proposition 8.15 *Let $\ell, t : \mathbb{N} \to \mathbb{N}$ be monotonically increasing functions and let $\ell^{-1}(t(n))$ denote the smallest integer k such that $\ell(k) \geq t(n)$. If there exists a canonical derandomizer of stretch ℓ then, for every time-constructible $t : \mathbb{N} \to \mathbb{N}$, it holds that* $\text{BPTIME}(t) \subseteq \text{DTIME}(T)$*, where* $T(n) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$.

Proof Sketch: Just mimic the proof of Theorem 8.13, which in turn uses Construction 8.2. (Recall that given any randomized algorithm A and generator G, Construction 8.2 yields an algorithm A_G of randomness-complexity $\ell^{-1} \circ t$ and time-complexity $\text{poly}(2^{\ell^{-1} \circ t}) + t$.)[25] Observe that the complexity of the resulting deterministic procedure is dominated by the $2^k = 2^{\ell^{-1}(t(|x|))}$ invocations of $A_G(x,s) = A(x, G(s))$, where $s \in \{0,1\}^k$, and each of these invocations takes time $\text{poly}(2^{\ell^{-1}(t(|x|))}) + t(|x|)$. Thus, on input an n-bit long string, the deterministic procedure runs in time $\text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. The correctness of this procedure (which takes a majority vote among the 2^k invocations of A_G) follows by combining Eq. (8.11) with the hypothesis that $\Pr[A(x) = 1]$ is bounded-away from $1/2$. Specifically, using the hypothesis $|\Pr[A(x)=1] - (1/2)| \geq 1/6$, it follows that the majority vote of $(A_G(x,s))_{s \in \{0,1\}^k}$ equals 1 (equiv., $\Pr[A(x, G(U_k)) = 1] > 1/2$) if and only if $\Pr[A(x)=1] > 1/2$ (equiv., $\Pr[A(x, U_{\ell(k)}) = 1] > 1/2$) Indeed, the implication is due to Eq. (8.11), when applied to the circuit $C_x(r) = A(x,r)$ (which has size at most $|r|^2$). $\square$

The goal. In light of Proposition 8.15, we seek canonical derandomizers with stretch that is as large as possible. The stretch cannot be super-exponential (i.e., it must hold that $\ell(k) = O(2^k)$), because there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (8.11) (see Exercise 8.18) whereas for $\ell(k) = \omega(2^k)$ it holds that $O(2^k \cdot \ell(k)) < \ell(k)^2$. Thus, our goal is to construct a canonical derandomizer with stretch $\ell(k) = 2^{\Omega(k)}$. Such a canonical derandomizer will allow for a "full derandomization of $\mathcal{BPP}$":

Theorem 8.16 *If there exists a canonical derandomizer of stretch $\ell(k) = 2^{\Omega(k)}$, then $\mathcal{BPP} = \mathcal{P}$.*

[25] Actually, given any randomized algorithm A and generator G, Construction 8.2 yields an algorithm A_G that is defined such that $A_G(x,s) = A(x, G'(s))$, where $|s| = \ell^{-1}(t(|x|))$ and $G'(s)$ denotes the $t(|x|)$-bit long prefix of $G(s)$. For simplicity, we shall assume here that $\ell(|s|) = t(|x|)$, and thus use G rather than G'. Note that given n we can find $k = \ell^{-1}(t(n))$ by invoking $G(1^i)$ for $i = 1, ..., k$ (using the fact that $\ell : \mathbb{N} \to \mathbb{N}$ is monotonically increasing). Also note that $\ell(k) = O(2^k)$ must hold (see Footnote 23), and thus we may replace $\text{poly}(2^k \cdot \ell(k))$ by $\text{poly}(2^k)$.

Proof: Using Proposition 8.15, we get $\mathrm{BP}\textsc{time}(t) \subseteq \mathrm{D}\textsc{time}(T)$, where $T(n) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n)) = \mathrm{poly}(t(n))$. ■

Reflections: Recall that a canonical derandomizer G was defined in a way that allows it to have time-complexity t_G that is larger than the size of the circuits that it fools (i.e., $t_G(k) > \ell(k)^2$ is allowed). Furthermore, $t_G(k) > 2^k$ was also allowed. Thus, if indeed $t_G(k) = 2^{\Omega(k)}$ (as is the case in Section 8.3.2), then *$G(U_k)$ can be distinguished from $U_{\ell(k)}$ in time* $2^k \cdot t_G(k) = \mathrm{poly}(t_G(k))$ by trying all possible seeds.[26] We stress that the latter distinguisher is a uniform algorithm (and it works by invoking G on all possible seeds). In contrast, for a general-purpose pseudorandom generator G (as discussed in Section 8.2) it holds that $t_G(k) = \mathrm{poly}(k)$, while *for every polynomial p it holds that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ in time $p(t_G(k))$.*

8.3.2 Constructing canonical derandomizers

The fact that canonical derandomizers are allowed to be more complex than the corresponding distinguisher makes *some* of the techniques of Section 8.2 inapplicable in the current context. For example, the stretch function cannot be amplified as in Section 8.2.4 (see Exercise 8.17). On the other hand, the techniques developed in the current section are inapplicable to Section 8.2. For example, the pseudorandomness of some canonical derandomizers (i.e., the generators of Construction 8.17) holds even when the potential distinguisher is given the seed itself. This amazing phenomenon capitalizes on the fact that the distinguisher's time-complexity does not allow for running the generator on the given seed.

8.3.2.1 The construction and its consequences

As in Section 8.2.5, the construction presented next transforms computational difficulty into pseudorandomness, except that here both computational difficulty and pseudorandomness are of a somewhat different form than in Section 8.2.5. Specifically, here we use Boolean predicates that are computable in exponential-time but are T-inapproximable for some exponential function T (see Definition 7.9 recapitulated next). That is, we assume *the existence of a Boolean predicate and constants $c, \varepsilon > 0$ such that for all but finitely many m, the* (residual) *predicate $f : \{0,1\}^m \to \{0,1\}$ is computable in time 2^{cm} but for any circuit C of size $2^{\varepsilon m}$ it holds that* $\mathsf{Pr}[C(U_m) = f(U_m)] < \frac{1}{2} + 2^{-\varepsilon m}$. (Needless to say, $\varepsilon < c$.) Recall that such predicates exist under the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity (see Theorem 7.19). With these preliminaries, we turn to the construction of canonical derandomizers with exponential stretch.

---

[26]We note that this distinguisher does not contradict the hypothesis that G is a canonical derandomizer, because $t_G(k) > \ell(k)$ definitely holds whereas $\ell(k) \leq 2^k$ typically holds (and so $2^k \cdot t_G(k) > \ell(k)^2$).

Construction 8.17 (The Nisan-Wigderson Construction):[27] *Let $f : \{0,1\}^m \to \{0,1\}$ and $S_1, ..., S_\ell$ be a sequence of m-subsets of $\{1, ..., k\}$. Then, for $s \in \{0,1\}^k$, we let*

$$G(s) \stackrel{\text{def}}{=} f(s_{S_1}) \cdots f(s_{S_\ell}) \tag{8.12}$$

where s_S denotes the projection of s on the bit locations in $S \subseteq \{1, ..., |s|\}$; that is, for $s = \sigma_1 \cdots \sigma_k$ and $S = \{i_1, ..., i_m\}$, we have $s_S = \sigma_{i_1} \cdots \sigma_{i_m}$.

Letting k vary and $\ell, m : \mathbb{N} \to \mathbb{N}$ be functions of k, we wish G to be a canonical derandomizer and $\ell(k) = 2^{\Omega(k)}$. One (obvious) necessary condition for this to happen is that the sets must be distinct, and hence $m(k) = \Omega(k)$; consequently, f must be computable in exponential-time. Furthermore, the sequence of sets $S_1, ..., S_{\ell(k)}$ must be constructible in $\mathrm{poly}(2^k)$ time. Intuitively, the function f should be strongly inapproximable (i.e., T-inapproximable for some exponential function T), and furthermore it seems desirable to use a set system with small pairwise intersections (because this restricts the overlap among the various inputs to which f is applied). Interestingly, these conditions are essentially sufficient.

Theorem 8.18 (analysis of Construction 8.17): *Let $\alpha, \beta, \gamma, \varepsilon > 0$ be constants satisfying $\varepsilon > (2\alpha/\beta) + \gamma$, and consider the functions $\ell, m, T : \mathbb{N} \to \mathbb{N}$ such that $\ell(k) = 2^{\alpha k}$, $m(k) = \beta k$, and $T(n) = 2^{\varepsilon n}$. Suppose that the following two conditions hold:*

1. *There exists an exponential-time computable function $f : \{0,1\}^* \to \{0,1\}$ that is T-inapproximable.* (See Definition 7.9.)
2. *There exists an exponential-time computable function $S : \mathbb{N} \times \mathbb{N} \to 2^{\mathbb{N}}$ such that*
   (a) *For every k and $i \in [\ell(k)]$, it holds that $S(k,i) \subseteq [k]$ and $|S(k,i)| = m(k)$.*
   (b) *For every k and $i \neq j$, it holds that $|S(k,i) \cap S(k,j)| \leq \gamma \cdot m(k)$.*

Then, using G as defined in Construction 8.17 with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.

Before proving Theorem 8.18 we note that, for any $\gamma > 0$, a function S as in Condition 2 does exist with some $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$; see Exercise 8.19. Combining such a function S with Theorems 7.19 and 8.18, we obtain a canonical derandomizer with exponential stretch based on the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.[28] Combining this with Theorem 8.16, we get the first part of the following theorem.

---

[27] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. This construction originates in [172, 175].

[28] Specifically, starting with a function having circuit complexity at least $\exp(\varepsilon_0 m)$, we apply Theorem 7.19 and obtain a T-inapproximable predicate for $T(m) = 2^{\varepsilon m}$, where the constant $\varepsilon \in (0, \varepsilon_0)$ depends on the constant ε_0. Next, we set $\gamma = \varepsilon/2$ and invoke Exercise 8.19, which determines $\alpha, \beta > 0$ such that $\ell(k) = 2^{\alpha k}$ and $m(k) = \beta k$. Note that (by possibly decreasing α) we get $(2\alpha/\beta) + \gamma < \varepsilon$.

Theorem 8.19 (derandomization of BPP, revisited):

1. *Suppose that $\mathcal{E}$ contains a decision problem that has almost-everywhere exponential circuit complexity* (i.e., there exists a constant $\varepsilon_0 > 0$ such that, for all but finitely many m's, any circuit that correctly decides this problem on $\{0,1\}^m$ has size at least $2^{\varepsilon_0 m}$). *Then, $\mathcal{BPP} = \mathcal{P}$.*

2. *Suppose that, for every polynomial p, the class $\mathcal{E}$ contains a decision problem that has circuit complexity that is almost-everywhere greater than p. Then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\mathrm{def}}{=} 2^{n^\varepsilon}$.*

Part 2 is proved (in Exercise 8.23) by using a generalization of Theorem 8.18, which in turn is provided in Exercise 8.22. We note that Part 2 of Theorem 8.19 supersedes Theorem 8.13 (see Exercise 7.24). As in the case of general-purpose pseudorandom generators, the hardness hypothesis made in each part of Theorem 8.19 is necessary for the existence of a corresponding canonical derandomizer (see Exercise 8.24).

The two parts of Theorem 8.19 exhibit two extreme cases: Part 1 (often referred to as the "high end") assumes an extremely strong circuit lower-bound and yields "full derandomization" (i.e., $\mathcal{BPP} = \mathcal{P}$), whereas Part 2 (often referred to as the "low end") assumes an extremely weak circuit lower-bound and yields weak but meaningful derandomization. Intermediate results (relying on intermediate lower-bound assumptions) can be obtained analogous to Exercise 8.23, but tight trade-offs are obtained differently (cf., [225]).

8.3.2.2 Analyzing the construction (i.e., proof of Theorem 8.18)

Using the time complexity upper-bounds on f and S, it follows that G can be computed in exponential time. Thus, our focus is on showing that $\{G(U_k)\}$ cannot be distinguished from $\{U_{\ell(k)}\}$ by circuits of size $\ell(k)^2$; specifically, that G satisfies Eq. (8.11). In fact, we will prove that this holds for $G'(s) = s \cdot G(s)$; that is, G fools such circuits even if they are given the seed as auxiliary input. (Indeed, these circuits are smaller than the running time of G, and so they cannot just evaluate G on the given seed.)

We start by presenting the intuition underlying the proof. As a warm-up suppose that the sets (i.e., $S(k,i)$'s) used in the construction are disjoint. In such a case (which is indeed impossible because $k < \ell(k) \cdot m(k)$), the pseudorandomness of $G(U_k)$ would follow easily from the inapproximability of f, because in this case G consists of applying f to non-overlapping parts of the seed (see Exercise 8.21). In the actual construction being analyzed here, the sets (i.e., $S(k,i)$'s) are not disjoint but have relatively small pairwise intersection, which means that G applies f on parts of the seed that have relatively small overlap. Intuitively, such small overlaps guarantee that the values of f on the corresponding inputs are "computationally independent" (i.e., having the value of f at some inputs $x_1, ..., x_i$ does not help in approximating the value of f at another input x_{i+1}). This intuition will be backed by showing that, when fixing all bits that do not appear in the target input (i.e., in x_{i+1}), the former values (i.e., $f(x_1), ..., f(x_i)$) can be computed at a relatively small computational cost. Thus, the values $f(x_1), ..., f(x_i)$ do not (significantly) facilitate the task of approximating $f(x_{i+1})$. With the foregoing intuition in mind, we now turn to the actual proof.

As usual, the actual proof employs a reducibility argument; that is, assuming towards the contradiction that G' does not fool some circuit of size $\ell(k)^2$, we derive a contradiction to the hypothesis that the predicate f is T-inapproximable. The argument utilizes the relation between pseudorandomness and unpredictability (cf. Section 8.2.5). Specifically, as detailed in Exercise 8.20, *any circuit that distinguishes $G'(U_k)$ from $U_{\ell(k)+k}$ with gap $1/6$, yields a next-bit predictor of similar size that succeeds in predicting the next bit with probability at least $\frac{1}{2}+\frac{1}{6\ell'(k)} > \frac{1}{2}+\frac{1}{7\ell(k)}$*, where the factor of $\ell'(k) = \ell(k)+k < (1+o(1))\cdot\ell(k)$ is introduced by the hybrid technique (cf. Eq. (8.7)). Furthermore, given the non-uniform setting of the current proof, we may fix a bit location $i+1$ for prediction, rather than analyzing the prediction at a random bit location. Indeed, $i \geq k$ must hold, because the first k bits of $G'(U_k)$ are uniformly distributed. In the rest of the proof, we transform the foregoing predictor into a circuit that approximates f better than allowed by the hypothesis (regarding the inapproximability of f).

Assuming that a small circuit C' can predict the $i+1^{\text{st}}$ bit of $G'(U_k)$, when given the previous i bits, we construct a small circuit C for approximating $f(U_{m(k)})$ on input $U_{m(k)}$. The point is that the $i+1^{\text{st}}$ bit of $G'(s)$ equals $f(s_{S(k,j+1)})$, where $j = i-k \geq 0$, and so C' approximates $f(s_{S(k,j+1)})$ based on $s, f(s_{S(k,1)}), ..., f(s_{S(k,j)})$, where $s \in \{0,1\}^k$ is uniformly distributed. Note that this is the type of thing that we are after, except that the circuit we seek may only get $s_{S(k,j+1)}$ as input.

The first observation is that C' maintains its advantage when we fix the best choice for the bits of s that are not at bit locations $S_{j+1} = S(k,j+1)$ (i.e., the bits $s_{[k]\setminus S_{j+1}}$). That is, by an averaging argument, it holds that

$$\begin{aligned}
&\max_{s'\in\{0,1\}^{k-m(k)}}\{\mathsf{Pr}_{s\in\{0,1\}^k}[C'(s,f(s_{S_1}),...,f(s_{S_j})) = f(s_{S_{j+1}})\,|\,s_{[k]\setminus S_{j+1}} = s']\} \\
&\geq\ p' \stackrel{\text{def}}{=} \mathsf{Pr}_{s\in\{0,1\}^k}[C'(s,f(s_{S_1}),...,f(s_{S_j})) = f(s_{S_{j+1}})].
\end{aligned}$$

Recall that by the hypothesis $p' > \frac{1}{2}+\frac{1}{7\ell(k)}$. Hard-wiring the fixed string s' into C', and letting $\pi(x)$ denote the (unique) string s satisfying $s_{S_{j+1}} = x$ and $s_{[k]\setminus S_{j+1}} = s'$, we obtain a circuit C'' that satisfies

$$\mathsf{Pr}_{x\in\{0,1\}^{m(k)}}[C''(x,f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})) = f(x)]\ \geq\ p'.$$

The circuit C'' is almost what we seek. The only problem is that C'' gets as input not only x, but also $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$, whereas we seek an approximator of $f(x)$ that only gets x.

The key observation is that each of the "missing" values $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$ depend only on a relatively small number of the bits of x. This fact is due to the hypothesis that $|S_t \cap S_{j+1}| \leq \gamma\cdot m(k)$ for $t = 1,...,j$, which means that $\pi(x)_{S_t}$ is an $m(k)$-bit long string in which $m_t \stackrel{\text{def}}{=} |S_t \cap S_{j+1}|$ bits are projected from x and the rest are projected from the *fixed* string s'. Thus, given x, the value $f(\pi(x)_{S_t})$ can be computed by a (trivial) circuit of size $\widetilde{O}(2^{m_t})$; that is, by a circuit implementing a look-up table on m_t bits. Using all these circuits (together with C''), we will obtain the desired approximator of f. Details follow.

We obtain the desired circuit, denoted C, that T-approximates f as follows. The circuit C depends on the index j and the string s' that are fixed as in the foregoing analysis. Recall that C incorporates ($\widetilde{O}(2^{\gamma\cdot|x|})$-size) circuits for computing $x \mapsto f(\pi(x)_{S_t})$, for $t = 1,...,j$. On input $x \in \{0,1\}^{m(k)}$, the circuit C computes the values $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$, invokes C'' on input x and these values, and outputs the answer as a guess for $f(x)$. That is,

$$C(x) = C''(x, f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})) = C'(\pi(x), f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})).$$

By the foregoing analysis, $\mathsf{Pr}_x[C(x) = f(x)] \geq p' > \frac{1}{2} + \frac{1}{7\ell(k)}$, which is lower-bounded by $\frac{1}{2} + \frac{1}{T(m(k))}$, because $T(m(k)) = 2^{\varepsilon m(k)} = 2^{\varepsilon\beta k} \gg 2^{2\alpha k} \gg 7\ell(k)$, where the first inequality is due to $\varepsilon > 2\alpha/\beta$ and second inequality is due to $\ell(k) = 2^{\alpha k}$. The size of C is upper-bounded by $\ell(k)^2 + \ell(k)\cdot\widetilde{O}(2^{\gamma\cdot m(k)}) \ll \widetilde{O}(\ell(k)^2 \cdot 2^{\gamma\cdot m(k)}) = \widetilde{O}(2^{2\alpha\cdot(m(k)/\beta)+\gamma\cdot m(k)}) \ll T(m(k))$, where the last inequality is due to $T(m(k)) = 2^{\varepsilon m(k)} \gg \widetilde{O}(2^{(2\alpha/\beta)\cdot m(k)+\gamma\cdot m(k)})$ (which in turn uses $\varepsilon > (2\alpha/\beta) + \gamma$). Thus, we derived a contradiction to the hypothesis that f is T-inapproximable. This completes the proof of Theorem 8.18.

8.3.3 Technical variations and conceptual reflections

We start this section by discussing a general framework that emerges from Construction 8.17, and end this section with a conceptual discussion regarding derandomization.

8.3.3.1 Construction 8.17 as a general framework

The Nisan–Wigderson Construction (i.e., Construction 8.17) is actually a general framework, which can be instantiated in various ways. Some of these instantiations, which are based on an abstraction of the construction as well as of its analysis, are briefly reviewed next,

We first note that the generator described in Construction 8.17 consists of a generic algorithmic scheme that can be instantiated with any predicate f. Furthermore, this algorithmic scheme, denoted G, is actually an *oracle machine* that makes (non-adaptive) queries to the function f, and thus the combination may be written as G^f. Likewise, the proof of pseudorandomness of G^f (i.e., the bulk of the proof of Theorem 8.18) is actually a general scheme that, for every f, yields a (non-uniform) oracle-aided circuit C that approximates f by using an oracle call to any distinguisher for G^f (i.e., C uses the distinguisher as a black-box). The circuit C does depends on f (but in a restricted way). Specifically, C contains look-up tables for computing functions obtained from f by fixing some of the input bits (i.e., look-up tables for the functions $f(\pi(\cdot)_{S_t})$'s). The foregoing abstractions facilitate the presentation of the following instantiations of the general framework underlying Construction 8.17

Derandomization of constant-depth circuits. In this case we instantiate Construction 8.17 using the `parity` function in the role of the inapproximable predicate f, noting that `parity` is indeed inapproximable by "small" constant-depth circuits. With an adequate setting of parameters we obtain pseudorandom generators with stretch $\ell(k) = \exp(k^{1/O(1)})$ that fool "small" constant-depth circuits (see [172]). The analysis of this construction proceeds very much like the proof of Theorem 8.18. One important observation is that incorporating the (straightforward) circuits that compute $f(\pi(x)_{S_t})$ into the distinguishing circuit only increases its depth by two levels. Specifically, the circuit C uses depth-two circuits that compute the values $f(\pi(x)_{S_t})$'s, and then obtains a prediction of $f(x)$ by using these values in its (single) invocation of the (given) distinguisher.

The resulting pseudorandom generator, which use a seed of polylogarithmic length (equiv., $\ell(k) = \exp(k^{1/O(1)})$), can be used for derandomizing $\mathcal{RAC}^0$ (i.e., random $\mathcal{AC}^0$), analogously to Theorem 8.16. Thus, we can *deterministically* approximate, in quasi-polynomial-time and up-to an additive error, the fraction of inputs that satisfy a given (constant-depth) circuit. Specifically, for any constant d, given a depth-d circuit C, we can deterministically approximate the fraction of the inputs that satisfy C (i.e., cause C to evaluate to 1) to within any *additive constant error*[29] in time $\exp((\log|C|)^{O(d)})$. Providing a deterministic polynomial-time approximation, even in the case $d = 2$ (i.e., CNF/DNF formulae) is an open problem.

Derandomization of probabilistic proof systems. A different (and more surprising) instantiation of Construction 8.17 utilizes predicates that are inapproximable by small *circuits having oracle access to* $\mathcal{NP}$. The result is a pseudorandom generator robust against two-move public-coin interactive proofs (which are as powerful as constant-round interactive proofs (see §9.1.3.1)). The key observation is that the analysis of Construction 8.17 provides a black-box procedure for approximating the underlying predicate when given oracle access to a distinguisher (and this procedure is valid also in case the distinguisher is a non-deterministic machine). Thus, under suitably strong (and yet plausible) assumptions, constant-round interactive proofs collapse to $\mathcal{NP}$. We note that a stronger result, which deviates from the foregoing framework, has been subsequently obtained (cf. [166]).

Construction of randomness extractors. An even more radical instantiation of Construction 8.17 was used to obtain explicit constructions of randomness extractors (see Appendix D.4). In this case, the predicate f is viewed as (an error correcting encoding of) a somewhat random function, and the construction makes sense because it refers to f in a black-box manner. In the analysis we rely on the fact that f can be approximated by combining relatively little information (regard-

[29] We mention that in the special case of approximating the number of satisfying assignment of a DNF formula, *relative error* approximations can be obtained by employing a deterministic reduction to the case of additive constant error (see §6.2.2.1). Thus, using a pseudorandom generator that fools DNF formulae, we can deterministically obtain a relative (rather than additive) error approximation to the number of satisfying assignment in a given DNF formula.

ing f) with (black-box access to) a distinguisher for G^f. For further details see §D.4.2.2.

8.3.3.2 Reflections regarding derandomization

Part 1 of Theorem 8.19 is often summarized by saying that (under some reasonable assumptions) *randomness is useless*. We believe that this interpretation is wrong even within the restricted context of traditional complexity classes, and is bluntly wrong if taken outside of the latter context. Let us elaborate.

Taking a closer look at the proof of Theorem 8.16 (which underlies Theorem 8.19), we note that a randomized algorithm A of time-complexity t is emulated by a deterministic algorithm A' of time complexity $t' = \text{poly}(t)$. Further noting that $A' = A_G$ invokes A (as well as the canonical derandomizer G) for $\Omega(t)$ times (because $\ell(k) = O(2^k)$ implies $2^k = \Omega(t)$), we infer that $t' = \Omega(t^2)$ must hold. Thus, derandomization via (Part 1 of) Theorem 8.19 is not really for free.

More importantly, we note that derandomization is not possible in various distributed settings, when both parties may protect their conflicting interests by employing randomization. Notable examples include most cryptographic primitives (e.g., encryption) as well as most types of probabilistic proof systems (e.g., PCP). For further discussion see Chapter 9 and Appendix C. Additional settings where randomness makes a difference (either between impossibility and possibility or between formidable and affordable cost) include distributed computing (see [16]), communication complexity (see [147]), parallel architectures (see [150]), sampling (see Appendix D.3), and property testing (see Section 10.1.2).

8.4 Space-Bounded Distinguishers

In the previous two sections we have considered generators that output sequences that look random to any efficient procedures, where the latter were modeled by time-bounded computations. Specifically, in Section 8.2 we considered indistinguishability by polynomial-time procedures. A finer classification of time-bounded procedures is obtained by considering their *space-complexity*; that is, restricting the space-complexity of time-bounded computations. This restriction, which is the focus of Chapter 5, leads to the notion of pseudorandom generators that fool space-bounded distinguishers. Interestingly, in contrast to the notions of pseudorandom generators that were considered in Sections 8.2 and 8.3, the existence of pseudorandom generators that fool space-bounded distinguishers can be established without relying on computational assumptions.

Prerequisites: Technically speaking, the current section is self-contained, but various definitional choices are justified by reference to §6.1.5.1. Thus, we recommend Section 6.1.5 as general background for the current section.

8.4.1 Definitional issues

Unfortunately, natural notions of space-bounded computations are quite subtle, especially when non-determinism or randomization are concerned (see Sections 5.3 and 6.1.5, respectively). Two major definitional issues regarding randomized space-bounded computations are the need for imposing explicit *time bounds* and the type of *access to the random tape*.

1. Time bounds: The question is whether or not the space-bounded machines are restricted to time-complexity that is at most exponential in their space-complexity.[30] Recall that such an upper-bound follows automatically in the deterministic case (Theorem 5.3), and can be assumed without loss of generality in the non-deterministic case (see Section 5.3.2), *but it does not necessarily hold in the randomized case* (see §6.1.5.1). Furthermore, failing to restrict the time-complexity of randomized space-bounded machines makes them unnatural and unintentionally too strong (see §6.1.5.1 again).

   As in Section 6.1.5, seeking a natural model of randomized space-bounded algorithms, we postulate that their time-complexity must be at most exponential in their space-complexity.

2. Access to the random tape: Recall that randomized algorithms may be modeled as machines that are provided with the necessary randomness via a special random-tape. The question is whether the space-bounded machine has uni-directional or bi-directional (i.e., unrestricted) access to its random-tape. (Allowing bi-directional access means that the randomness is recorded "for free"; that is, without being accounted for in the space-bound (see discussions in Sections 5.3 and 6.1.5).)

   Recall that uni-directional access to the random-tape corresponds to the natural model of an on-line randomized machine, which determines its moves based on its internal coin tosses (and thus cannot store its past coin tosses "for free"). Thus, as in Section 6.1.5, we consider uni-directional access.[31]

Hence, we focus on randomized space-bounded computation that have time-complexity that is at most exponential in their space-complexity and access their random-tape in a uni-directional manner. *In accordance with this definition of randomized space-bounded computation, we consider space-bounded distinguishers that have a uni-directional access to the input sequence that they examine.* Let us consider the type of algorithms that arise.

We consider *space-bounded algorithms that have a uni-directional access to their input.* At each step, based on the contents of its temporary storage, such an

[30] Alternatively, one can ask whether these machines must always halt or only halt with probability approaching 1. It can be shown that the only way to ensure "absolute halting" is to have time-complexity that is at most exponential in the space-complexity. (In the current discussion as well as throughout this section, we assume that the space-complexity is at least logarithmic.)

[31] We note that the fact that we restrict our attention to uni-directional access is instrumental in obtaining space-robust generators without making intractability assumptions. Analogous generators for bi-directional space-bounded computations would imply hardness results of a breakthrough nature in the area.

algorithm may either read the next input bit or stay at the current location on the input, where in either case the algorithm may modify its temporary storage. To simplify our analysis of such algorithms, we consider a corresponding *non-uniform model* in which, at each step, the algorithm reads the next input bit and update its temporary storage according to an arbitrary function applied to the previous contents of that storage (and to the new bit). Note that we have strengthened the model by allowing arbitrary (updating) functions, which can be implemented by (non-uniform) circuits having size that is exponential in the space-bound, rather than using (updating) functions that can be (uniformly) computed in time that is exponential in the space-bound. This strengthening is motivated by the fact that the known constructions of pseudorandom generators remain valid also when the space-bounded distinguishers are non-uniform and by the fact that non-uniform distinguishers arise anyhow in derandomization.

The computation of the foregoing non-uniform space-bounded algorithms (or automata)[32] can be represented by directed layered graphs, where the vertices in each layer correspond to possible contents of the temporary storage and transition between neighboring layers corresponds to a step of the computation. Foreseeing the application of this model for the description of potential distinguishers, we parameterize these layered graphs based on the index, denoted k, of the relevant ensembles (e.g., $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$). That is, we present both the input length, denoted $\ell = \ell(k)$, and the space-bound, denoted $s(k)$, as functions of the parameter k. Thus, we define a non-uniform automaton of space $s : \mathbb{N}\rightarrow\mathbb{N}$ as a family, $\{D_k\}_{k\in\mathbb{N}}$, of directed layered graphs with labeled edges such that the following conditions hold:

- The digraph D_k consists of $\ell(k) + 1$ layers, each containing at most $2^{s(k)}$ vertices. The first layer contains a single vertex, which is the digraph's (single) source (i.e., a vertex with no incoming edges), and the last layer contains all the digraph's sinks (i.e., vertices with no outgoing edges).
- The only directed edges in D_k are between adjacent layers, going from layer i to layer $i + 1$, for $i \leq \ell(k)$. These edges are labeled such that each (non-sink) vertex of D_k has two (possibly parallel) outgoing directed edges, one labeled 0 and the other labeled 1.

The result of the computation of such an automaton, on an input of adequate length (i.e., length ℓ where D_k has $\ell + 1$ layers), is defined as the vertex (in last layer) reached when following the sequence of edges that are labeled by the corresponding bits of the input. That is, on input $x = x_1 \cdots x_\ell$, in the i^{th} step (for $i = 1, ..., \ell$) we move from the current vertex (which resides in the i^{th} layer) to one of its neighbors

[32] We use the term automaton (rather than algorithm or machine) in order to remind the reader that this computing device reads its input in a uni-directional manner. Alternative terms that may be used are "real-time" or "on-line" machines. We prefer not using the term "on-line" machine in order to keep a clear distinction from randomized (on-line) algorithms that have free access to their input (and on-line access to a source of randomness). Indeed, the automata consider here arise from the latter algorithms by fixing their primary input and considering the random source as their (only) input. We also note that the automata considered here are a special case of Ordered Binary Decision Diagrams (OBDDs; see [235]).

(which resides in the $i+1^{\rm st}$ layer) by following the outgoing edge labeled x_i. Using a fixed partition of the vertices of the last layer, this defines a natural notion of a decision (by D_k); that is, we write $D_k(x)=1$ if on input x the automaton D_k reached a vertex that belongs to the first part of the aforementioned partition.

Definition 8.20 (Indistinguishability by space-bounded automata):

- *For a non-uniform automaton, $\{D_k\}_{k\in\mathbb{N}}$, and two probability ensembles, $\{X_k\}_{k\in\mathbb{N}}$ and $\{Y_k\}_{k\in\mathbb{N}}$, the function $d:\mathbb{N}\rightarrow[0,1]$ defined as*

$$d(k) \stackrel{\rm def}{=} |\mathsf{Pr}[D_k(X_k)=1] - \mathsf{Pr}[D_k(Y_k)=1]|$$

  is called the distinguishability-gap *of $\{D_k\}$ between the two ensembles.*

- *Let $s:\mathbb{N}\rightarrow\mathbb{N}$ and $\varepsilon:\mathbb{N}\rightarrow[0,1]$. A probability ensemble, $\{X_k\}_{k\in\mathbb{N}}$, is called (s,ε)-*pseudorandom *if for any non-uniform automaton of space $s(\cdot)$, the distinguishability-gap of the automaton between $\{X_k\}_{k\in\mathbb{N}}$ and the corresponding uniform ensemble (i.e., $\{U_{|X_k|}\}_{k\in\mathbb{N}}$) is at most $\varepsilon(\cdot)$.*

- *A deterministic algorithm G of stretch function ℓ is called an (s,ε)-*pseudorandom generator *if the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is (s,ε)-pseudorandom. That is, every non-uniform automaton of space $s(\cdot)$ has a distinguishing-gap of at most $\varepsilon(\cdot)$ between $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$.*

Thus, when using a random seed of length k, an (s,ε)-pseudorandom generator outputs a sequence of length $\ell(k)$ that looks random to observers having space $s(k)$. Note that $s(k)\leq k$ is a necessary condition for the existence of $(s,0.5)$-pseudorandom generators, because a non-uniform automaton of space $s(k)>k$ can recognize the image of a generator (which contains at most 2^k strings of length $\ell(k)>k$). More generally, there is a trade-off between $s(k)-k$ and the stretch of (s,ε)-pseudorandom generators; for details see Exercises 8.25 and 8.26.

Note: Recall that we stated the space-bound of the potential distinguisher (as well as the stretch function) in terms of the seed-length, denoted k, of the generator. In contrast, other sources present a parameterization in terms of the space-bound of the potential distinguisher, denoted m. The translation is obtained by using $m=s(k)$, and we shall provide it following the main statements of Theorems 8.21 and 8.22.

8.4.2 Two Constructions

In contrast to the case of pseudorandom generators that fool time-bounded distinguishers, pseudorandom generators that fool space-bounded distinguishers can be constructed without relying on any computational assumption. The following two theorems exhibit two rather extreme cases of a general trade-off between the space-bound of the potential distinguisher and the stretch function of the generator.[33]

[33] These two results have been "interpolated" in [11]: There exists a parameterized family of "space fooling" pseudorandom generators that includes both results as extreme special cases.

We stress that both theorems fall short of providing parameters as in Exercise 8.26, but they refer to relatively efficient constructions. We start with an attempt to maximize the stretch.

Theorem 8.21 (stretch exponential in the space-bound for $s(k) = \sqrt{k}$): *For every space constructible function $s:\mathbb{N}\to\mathbb{N}$, there exists an $(s, 2^{-s})$-pseudorandom generator of stretch function $\ell(k) = \min(2^{k/O(s(k))}, 2^{s(k)})$. Furthermore, the generator works in space that is linear in the length of the seed, and in time that is linear in the stretch function.*

In other words, for every $t \leq m$, we have a generator that takes a random seed of length $k = O(t \cdot m)$ and produce a sequence of length 2^t that looks random to any (non-uniform) automaton of space m (up to a distinguishing-gap of 2^{-m}). In particular, using a random seed of length $k = O(m^2)$, one can produce a sequence of length 2^m that looks random to any (non-uniform) automaton of space m. Thus, *one may replace random sequences used by any space-bounded computation, by sequences that are efficiently generated from random seeds of length quadratic in the space bound*. The common instantiation of the latter assertion is for log-space algorithms. In §8.4.2.2, we apply Theorem 8.21 (and its underlying ideas) for the derandomization of space-complexity classes such as $\mathcal{BPL}$ (i.e., the log-space analogue of $\mathcal{BPP}$). Theorem 8.21 itself is proved in §8.4.2.1.

We now turn to the case where one wishes to maximize the space-bound of potential distinguishers. We warn that Theorem 8.22 only guarantees a subexponential distinguishing gap (rather than the exponential distinguishing gap guaranteed in Theorem 8.21). This warning is voiced because failing to recall this limitation has led to errors in the past.

Theorem 8.22 (polynomial stretch and linear space-bound): *For any polynomial p and for some $s(k) = k/O(1)$, there exists an $(s, 2^{-\sqrt{s}})$-pseudorandom generator of stretch function p. Furthermore, the generator works in linear-space and polynomial-time* (both stated in terms of the length of the seed).

In other words, we have a generator that takes a random seed of length $k = O(m)$ and produce a sequence of length $\mathrm{poly}(m)$ that looks random to any (non-uniform) automaton of space m. Thus, one may *convert any randomized computation utilizing polynomial-time and linear-space into a functionally equivalent randomized computation of similar time and space complexities that uses only a linear number of coin tosses.*

8.4.2.1 Sketches of the proofs of Theorems 8.21 and 8.22

In both cases, we start the proof by considering a generic space-bounded distinguisher and show that the input distribution that this distinguisher examines can be modified (from the uniform distribution into a pseudorandom one) without having the distinguisher notice the difference. This modification (or rather a sequence of modifications) yields a construction of a pseudorandom generator, which is only spelled-out at the end of the argument.

Sketch of the proof of Theorem 8.21.[34] The main technical tool used in this proof is the "mixing property" of pairwise independent hash functions (see Appendix D.2). A family of functions H_n, which map $\{0,1\}^n$ to itself, is called mixing if for every pair of subsets $A, B \subseteq \{0,1\}^n$ for all but very few (i.e., $\exp(-\Omega(n))$ fraction) of the functions $h \in H_n$, it holds that

$$\mathsf{Pr}[U_n \in A \wedge h(U_n) \in B] \approx \frac{|A|}{2^n} \cdot \frac{|B|}{2^n} \tag{8.13}$$

where the approximation is up to an additive term of $\exp(-\Omega(n))$. (See the generalization of Lemma D.4, which implies that $\exp(-\Omega(n))$ can be set to $2^{-n/3}$.)

We may assume, without loss of generality, that $s(k) = \Omega(\sqrt{k})$, and thus $\ell(k) \leq 2^{s(k)}$ holds. For any $s(k)$-space distinguisher D_k as in Definition 8.20, we consider an auxiliary "distinguisher" D'_k that is obtained by "contracting" every block of $n \stackrel{\text{def}}{=} \Theta(s(k))$ consecutive layers in D_k, yielding a directed layered graph with $\ell' \stackrel{\text{def}}{=} \ell(k)/n < 2^{s(k)}$ layers (and $2^{s(k)}$ vertices in each layer). Specifically,

- each vertex in D'_k has 2^n (possibly parallel) directed edges going to various vertices of the next level; and
- each such edge is labeled by an n-bit long string such that the directed edge (u, v) labeled $\sigma_1\sigma_2\cdots\sigma_n$ in D'_k replaces the n-edge directed path between u and v in D_k that consists of edges labeled $\sigma_1, \sigma_2, ..., \sigma_n$.

The graph D'_k simulates D_k in the obvious manner; that is, the computation of D'_k on an input of length $\ell(k) = \ell' \cdot n$ is defined by breaking the input into consecutive substrings of length n and following the path of edges that are labeled by the corresponding n-bit long substrings.

The key observation is that D'_k cannot distinguish between a random $\ell' \cdot n$-bit long input (i.e., $U_{\ell' \cdot n} \equiv U_n^{(1)} U_n^{(2)} \cdots U_n^{(\ell')}$) and a "pseudorandom" input of the form $U_n^{(1)} h(U_n^{(1)}) U_n^{(2)} h(U_n^{(2)}) \cdots U_n^{(\ell'/2)} h(U_n^{(\ell'/2)})$, where $h \in H_n$ is a (suitably fixed) hash function. To prove this claim, we consider an arbitrary pair of neighboring vertices, u and v (in layers i and $i+1$, respectively), and denote by $L_{u,v} \subseteq \{0,1\}^n$ the set of the labels of the edges going from u to v. Similarly, for a vertex w at layer $i+2$, we let $L'_{v,w}$ denote the set of the labels of the edges going from v to w. By Eq. (8.13), for all but very few of the functions $h \in H_n$, it holds that

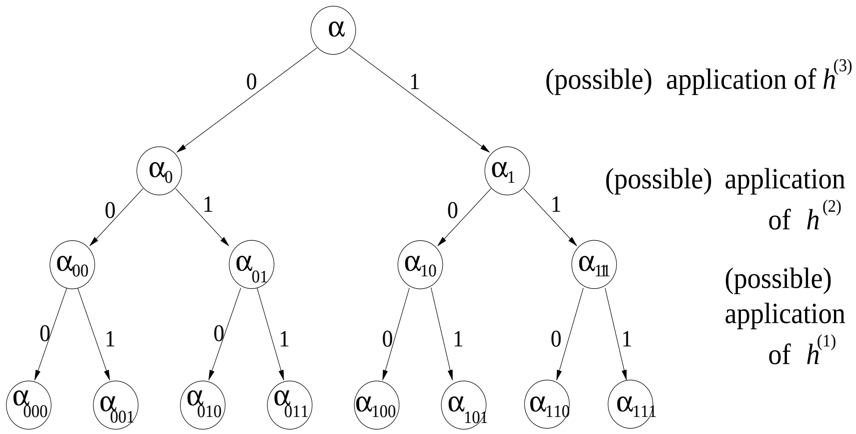
$$\mathsf{Pr}[U_n \in L_{u,v} \wedge h(U_n) \in L'_{v,w}] \approx \mathsf{Pr}[U_n \in L_{u,v}] \cdot \mathsf{Pr}[U_n \in L'_{v,w}], \tag{8.14}$$

where "very few" and $\approx$ are as in Eq. (8.13). Thus, for all but $\exp(-\Omega(n))$ fraction of the choices of $h \in H_n$, *replacing the coins in the second transition* (i.e., the transition from layer $i+1$ to layer $i+2$) *with the value of* h *applied to the outcomes of the coins used in the first transition* (i.e., the transition from layer i to $i+1$), *approximately maintains the probability that* D'_k *moves from* u *to* w *via* v. Using a union bound (on all triples (u, v, w) as in the foregoing), we note that, for all but

[34] A detailed proof appears in [173].

$2^{3s(k)} \cdot \ell' \cdot \exp(-\Omega(n))$ fraction of the choices of $h \in H_n$, the foregoing replacement approximately maintains the probability that D'_k moves through any specific two-edge path of D'_k.

Using $\ell' < 2^{s(k)}$ and a suitable choice of $n = \Theta(s(k))$, it holds that $2^{3s(k)} \cdot \ell' \cdot \exp(-\Omega(n)) < \exp(-\Omega(n))$, and thus all but "few" functions $h \in H_n$ are good for approximating all these transition probabilities. (We stress that the same h can be used in all these approximations.) Thus, *at the cost of extra* $|h|$ *random bits, we can reduce the number of true random coins used in transitions on* D'_k *by a factor of two, without significantly affecting the final decision of* D'_k (where again we use the fact that $\ell' \cdot \exp(-\Omega(n)) < \exp(-\Omega(n))$, which implies that the approximation errors do not accumulate to too much). In other words, at the cost of extra $|h|$ random bits, we can effectively contract the distinguisher to half its length while approximately maintaining the probability that the distinguisher accepts a random input. That is, fixing a good h (i.e., one that provides a good approximation to the transition probability over all $2^{3s(k)} \cdot \ell'$ two-edge paths), we can replace the two-edge paths in D'_k by edges in a new distinguisher D''_k (which depends on h) such that an edge (u,w) labeled $r \in \{0,1\}^n$ appears in D''_k if and only if, for some v, the path (u,v,w) appears in D'_k with the first edge (i.e., (u,v)) labeled r and the second edge (i.e., (v,w)) labeled $h(r)$. Needless to say, the crucial point is that $\Pr[D''_k(U_{(\ell'/2)\cdot n})=1]$ approximates $\Pr[D'_k(U_{\ell' \cdot n})=1]$.



The output of the generator (on seed $\alpha, h^{(1)}, ..., h^{(t)}$*) consists of the concatenation of the strings denoted* $\alpha_{0^t}, ..., \alpha_{1^t}$*, appearing in the leaves of the tree. For every* $x \in \{0,1\}^*$ *it holds that* $\alpha_{x0} = \alpha_x$ *and* $\alpha_{x1} = h^{(t-|x|)}(\alpha_x)$*. In particular, for* $t = 3$*, we have* $\alpha_{011} = h^{(1)}(\alpha_{01})$*, which equals* $h^{(1)}(h^{(2)}(\alpha_0)) = h^{(1)}(h^{(2)}(\alpha))$*, where* $\alpha = \alpha_\lambda$*.*

Figure 8.3: The first generator that "fools" space-bounded automata.

The forgoing process can be applied to D''_k resulting in a distinguisher D'''_k of half the length, and so on. Each time we contract the current distinguisher by a factor of two, and do so by randomly selecting (and fixing) a new hash function. Thus, repeating the process for a logarithmic (in the depth of D'_k) number of times we obtain a distinguisher that only examines n bits, at which point we stop. In total, we have used $t \stackrel{\rm def}{=} \log_2(\ell'/n) < \log_2 \ell(k)$ random hash functions, denoted $h^{(1)}, ..., h^{(t)}$. This means that we can generate a (pseudorandom) sequence that fools the original D_k by using a seed of length $n + t \cdot \log_2 |H_n|$ (see Figure 8.3 and Exercise 8.28). Using $n = \Theta(s(k))$ and an adequate family H_n (e.g., Construction D.3), we obtain the desired $(s, 2^{-s})$-pseudorandom generator, which indeed uses a seed of length $O(s(k) \cdot \log_2 \ell(k)) = k$. □

Rough sketch of the proof of Theorem 8.22.[35] The main technical tool used in this proof is a suitable randomness extractor (as defined in §D.4.1.1), which is indeed a much more powerful tool than hashing functions. The basic idea is that when the distinguisher D_k is at some "distant" layer, say at layer $t = \Omega(s(k))$, it typically "knows" little about the random choices that led it there. That is, D_k has only $s(k)$ bits of memory, which leaves out $t - s(k)$ bits of "uncertainty" (or randomness) regarding the previous moves. Thus, much of the randomness that led D_k to its current state may be "re-used" (or "recycled"). To re-use these bits we need to extract *almost* uniform distribution on strings of sufficient length out of the aforementioned distribution over $\{0,1\}^t$ that has entropy[36] at least $t - s(k)$. Furthermore, such an extraction requires some additional truly random bits, yet relatively few such bits. In particular, using $k' = \Omega(\log t)$ bits towards this end, the extracted bits are $\exp(-\Omega(k'))$ away from uniform.

The gain from the aforementioned recycling is significant if recycling is repeated sufficiently many times. Towards this end, we break the k-bit long seed into two parts, denoted $r' \in \{0,1\}^{k/2}$ and $(r_1, ..., r_{3\sqrt{k}})$, where $|r_i| = \sqrt{k}/6$, and set $n = k/3$. Intuitively, r' will be used for determining the first n steps, and it will be re-used (or recycled) together with r_i for determining the steps $i \cdot n + 1$ through $(i+1) \cdot n$. Looking at layer $i \cdot n$, we consider the information regarding r' that is "known" to D_k (when reaching a specific vertex at layer $i \cdot n$). Typically, the conditional distribution of r', given that we reached a specific vertex at layer $i \cdot n$, has (min-)entropy greater than $0.99 \cdot ((k/2) - s(k))$. Using r_i (as a seed of an extractor applied to r'), we can extract $0.9 \cdot ((k/2) - s(k) - o(k)) > k/3 = n$ bits that are almost-random (i.e., $2^{-\Omega(\sqrt{k})}$-close to U_n) with respect to D_k, and use these bits for determining the next n steps. Hence, using k random bits we are produce a sequence of length $(1 + 3\sqrt{k}) \cdot n > k^{3/2}$ that fools automata of space bound, say, $s(k) = k/10$. Specifically, using an extractor of the form $\text{Ext} : \{0,1\}^{\sqrt{k}/6} \times \{0,1\}^{k/2} \to \{0,1\}^{k/3}$, we map the seed $(r', r_1, ..., r_{3\sqrt{k}})$ to the output sequence $(r', \text{Ext}(r_1, r'), ..., \text{Ext}(r_{3\sqrt{k}}, r'))$. Thus, *we obtained an $(s, 2^{-\Omega(\sqrt{s})})$-pseudorandom*

[35] A detailed proof appears in [176].

[36] Actually, a stronger technical condition needs and can be imposed on the latter distribution. Specifically, with overwhelmingly high probability, at layer t, automaton D_k is at a vertex that can be reached in more than $2^{0.99 \cdot (t - s(k))}$ different ways. In this case, the distribution representing a random walk that reaches this vertex has min-entropy greater than $0.99 \cdot (t - s(k))$. The reader is referred to §D.4.1.1 for definitions of min-entropy and extractors.

generator of stretch function $\ell(k) = k^{3/2}$.

In order to obtain an arbitrary polynomial stretch rather than a specific polynomial stretch (i.e., $\ell(k) = k^{3/2}$), we repeatedly apply an adequate composition, to be outlined next. Suppose that G_1 is an (s_1, ε_1)-pseudorandom generator of stretch function ℓ_1 that works in linear space, and similarly for G_2 with respect to (s_1, ε_1) and ℓ_2. Then, we consider the following construction of a generator G:

1. On input $s \in \{0,1\}^k$, compute $G_1(s)$, and parse it into consecutive blocks, each of length $k' = s_1(k)/O(1)$, denoted $r_1, ..., r_t$, where $t = \ell_1(k)/k'$.

2. Compute and output the $t \cdot \ell_2(k')$-bit long sequence $G_2(r_1) \cdots G_2(r_t)$.

Note that $|G(s)| = \ell_1(k) \cdot \ell_2(k')/k'$, where $k' = s_1(k)/O(1)$ and $k = |s|$. For $s_1(k) = \Theta(k)$, we have $|G(s)| = \ell_1(k) \cdot \ell_2(\Omega(k))/O(k)$, which for polynomials ℓ_1 and ℓ_2 yields $|G(s)| = \ell_1(|s|) \cdot \ell_2(|s|)/O(|s|)$. We claim that G *is an* (s, ε)-*pseudorandom generator, for* $s(k) = \min(s_1(k)/2, s_2(\Omega(s_1(k)))$ *and* $\varepsilon(k) = \varepsilon_1(k) + \ell_1(k) \cdot \varepsilon_2(\Omega(s_1(k))$. The proof uses a hybrid argument, which refers to the natural distributions $G(U_k)$ and $U_{t \cdot \ell_2(k')} \equiv U^{(1)}_{\ell_2(k')} \cdots U^{(t)}_{\ell_2(k')}$ as well as to the intermediate hybrid distribution $I_k \stackrel{\rm def}{=} G_2(U^{(1)}_{k'}) \cdots G_2(U^{(t)}_{k'})$. The reader can verify that I_k is $(s_2(k'), t \cdot \varepsilon_2(k'))$-pseudorandom (see Exercise 8.27), and so we focus on proving that I_k is indistinguishable from $G(U_k)$ by automata of space $s_1(k)/2$ (with respect to distinguishing-gap $\varepsilon_1(k)$). This is proved by converting a potential distinguisher (of I_k and $G(U_k)$) into a distinguisher of $U_{\ell_1(k)} \equiv U_{t \cdot k'}$ and $G_1(U_k)$, where the new distinguisher parses the $\ell_1(k)$-bit long input into t blocks (each of length k'), invokes G_2 on the corresponding k'-bit long blocks, and feeds the resulting sequence of $\ell_1(k')$-bit long blocks to the original distinguisher. For this end, it is crucial that G_2 can be evaluate on k'-bit long strings using space at most $s_1(k)/2$, which is guaranteed by our setting of $k' = s_1(k)/O(1)$ and the hypothesis that G_2 works in linear space. □

8.4.2.2 Derandomization of space-complexity classes

As a direct application of Theorem 8.21, we obtain that $\mathcal{BPL} \subseteq \mathrm{DSPACE}(\log^2)$, where $\mathcal{BPL}$ denotes the log-space analogue of $\mathcal{BPP}$ (see Definition 6.11). (Recall that $\mathcal{NL} \subseteq \mathrm{DSPACE}(\log^2)$, but it is not known whether or not $\mathcal{BPL} \subseteq \mathcal{NL}$.)[37] A stronger derandomization result can be obtained by a finer analysis of the proof of Theorem 8.21.

Theorem 8.23 $\mathcal{BPL} \subseteq \mathcal{SC}$, *where* $\mathcal{SC}$ *denotes the class of decision problems that can be solved by a deterministic algorithm that runs in polynomial-time and polylogarithmic-space.*

Thus, $\mathcal{BPL}$ (and in particular $\mathcal{RL} \subseteq \mathcal{BPL}$) is placed in a class not known to contain $\mathcal{NL}$. Another such result was subsequently obtained in [195]: Randomized

[37]Indeed, the log-space analogue of $\mathcal{RP}$, denoted $\mathcal{RL}$, is contained in $\mathcal{NL} \subseteq \mathrm{DSPACE}(\log^2)$, and thus the fact that Theorem 8.21 implies $\mathcal{RL} \subseteq \mathrm{DSPACE}(\log^2)$ is of no interest.

log-space can be simulated in deterministic space $o(\log^2)$; specifically, in space $\log^{3/2}$. We mention that the archetypical problem of $\mathcal{RL}$ has been recently proved to be in $\mathcal{L}$ (see Section 5.2).

Sketch of the proof of Theorem 8.23.[38] We are going to use the generator construction provided in the proof of Theorem 8.21, but show that the main part of the seed (i.e., the sequence of hash functions) can be fixed (depending on the distinguisher at hand). Furthermore, this fixing can be performed in polylogarithmic space and polynomial-time. Specifically, wishing to derandomize a specific log-space computation (which refers to a specific input), we first obtain the corresponding distinguisher, denoted D'_k, that represents this computation (as a function of the outcomes of the internal coin tosses of the log-space algorithm). The key observation is that the question of whether or not a specific hash function $h \in H_n$ is good for a specific D'_k can be determined in space that is linear in $n = |h|/2$ and logarithmic in the size of D'_k. Indeed, the time-complexity of this decision procedure is exponential in its space-complexity. It follows that we can find a good $h \in H_n$, for a given D'_k, within these complexities (by scanning through all possible $h \in H_n$). Once a good h is found, we can also construct the corresponding graph D''_k (in which edges represent two-edge paths in D'_k), again within the same complexity. Actually, it will be more instructive to note that we can determine a step (i.e., an edge-traversal) in D''_k by making two steps (edge-traversals) in D'_k. This will allow to fix a hash function for D''_k, and so on. Details follow.

The main claim is that the entire process of finding a sequence of $t \stackrel{\rm def}{=} \log_2 \ell'(k)$ good hash functions can be performed in space $t \cdot O(n + \log |D_k|) = O(n + \log |D_k|)^2$ and time $\mathrm{poly}(2^n \cdot |D_k|)$; that is, the time-complexity is sub-exponential in the space-complexity (i.e., the time-complexity is significantly smaller than than the generic bound of $\exp(O(n + \log |D_k|)^2))$. Starting with $D_k^{(1)} = D'_k$, we find a good (for $D_k^{(1)}$) hashing function $h^{(1)} \in H_n$, which defines $D_k^{(2)} = D''_k$. Having found (and stored) $h^{(1)}, ..., h^{(i)} \in H_n$, which determine $D_k^{(i+1)}$, we find a good hashing function $h^{(i+1)} \in H_n$ for $D_k^{(i+1)}$ by emulating pairs of edge-traversals on $D_k^{(i+1)}$. Indeed, a key point is that we do *not* construct the sequence of graphs $D_k^{(2)}, ..., D_k^{(i+1)}$, but rather emulate an edge-traversal in $D_k^{(i+1)}$ by making 2^i edge-traversals in D'_k, using $h^{(1)}, ..., h^{(i)}$: The (edge-traversal) move $\alpha \in \{0,1\}^n$ starting at vertex v of $D_k^{(i+1)}$ translates to a sequence of 2^i moves starting at vertex v of D'_k, where the moves are determined by the 2^i-long sequence (of n-bit strings)

$$\overline{h}^{(0^i)}(\alpha), \overline{h}^{(0^{i-2}01)}(\alpha), \overline{h}^{(0^{i-2}10)}(\alpha), \overline{h}^{(0^{i-2}11)}(\alpha), ..., \overline{h}^{(1^i)}(\alpha),$$

where $\overline{h}^{(\sigma_i \cdots \sigma_1)}$ is the function obtained by the composition of a subsequence of the functions $h^{(i)}, ..., h^{(1)}$ determined by $\sigma_i \cdots \sigma_1$. Specifically, $\overline{h}^{(\sigma_i \cdots \sigma_1)}$ equals $h^{(i_{t'})} \circ \cdots \circ h^{(i_2)} \circ h^{(i_1)}$, where $i_1 < i_2 < \cdots < i_{t'}$ and $\{i_j : j = 1, ..., t'\} = \{j : \sigma_j = 1\}$.

Recall that the ability to perform edge-traversals on $D_k^{(i+1)}$ allows to determine whether a specific function $h \in H_n$ is good for $D_k^{(i+1)}$. This is done by considering

[38] A detailed proof appears in [174].

all the relevant triples (u,v,w) in $D_k^{(i+1)}$, computing for each such (u,v,w) the three quantities (i.e., probabilities) appearing in Eq. (8.14), and deciding accordingly. Trying all possible $h \in H_n$, we find a function (to be denoted $h^{(i+1)}$) that is good for $D_k^{(i+1)}$. This is done while using an additional storage of $s' = O(n + \log|D'_k|)$ (on top of the storage used to record $h^{(1)}, ..., h^{(i)}$), and in time that is exponential in s'. Thus, given D'_k, *we find a good sequence of hash functions,* $h^{(1)}, ..., h^{(t)}$, *in time exponential in* s' *and while using space* $s' + t \cdot \log_2|H_n| = O(t \cdot s')$. Such a sequence of functions allows us to emulate edge-traversals on $D_k^{(t+1)}$, which in turn allows to (deterministically) approximate the probability that D'_k accepts a random input (i.e., the probability that, starting at the single source vertex of the first layer, automaton D'_k reaches some accepting vertex at the last layer). This approximation is obtained by computing the corresponding probability in $D_k^{(t+1)}$ by traversing all 2^n edges.

To summarize, given D'_k, we can (deterministically) approximate the probability that D'_k accepts a random input in $O(t \cdot s')$-space and $\exp(O(s'+n))$-time, where $s' = O(n + \log|D'_k|)$ and $t < \log_2|D'_k|$. For $n = \Theta(\log|D'_k|)$, this means $O(\log|D'_k|)^2$-space and $\mathrm{poly}(|D'_k|)$-time. We comment that the approximation can be made accurate up to an additive term of $1/\mathrm{poly}(|D'_k|)$, but an additive term of $1/6$ suffices here.

We conclude the proof by recalling the connection between such an approximation and the derandomization of $\mathcal{BPL}$ (indeed, note the analogy to the proof of Theorem 8.13). The computation of a log-space probabilistic machine M on input x, can be represented by a directed layer graph $G_{M,x}$ of size $\mathrm{poly}(|x|)$. Specifically, the vertices of each layer represent possible configurations of the computation of $M(x)$, and the edges between the i^{th} layer and the $i+1^{\text{st}}$ layer represent the i^{th} move of such a computation, which depends on the i^{th} bit of the random-tape of M (or, equivalently, on the i^{th} internal coin toss of M).[39] Thus, the probability that M accepts x equals the probability that a random walk starting at the single vertex of the first layer of $G_{M,x}$ reaches some vertex in the last layer that represents an accepting configuration. Setting $k = \Theta(\log|x|)$ and $n = \Theta(k)$, the graph $G_{M,x}$ coincides with the graph D_k referred to at the beginning of the proof of Theorem 8.21, and D'_k is obtained from D_k by an "n-layer contraction" (see ibid.). Furthermore, D_k and D'_k can be constructed (from x) in logarithmic-space (and by using the emulative composition of Lemma 5.2 we may just proceed as if D'_k is given as input). Combining this with the foregoing analysis, we conclude that the probability that M accepts x can be deterministically approximated in $O(\log|x|)^2$-space and $\mathrm{poly}(|x|)$-time. The theorem follows. □

[39] Note that $G_{M,x}$ is a "layered version" of the graph that was considered (and denoted G_x) in the proof of Theorem 5.11. Furthermore, while in the proof of Theorem 5.11 we cared about the existence of certain paths, here we care about their quantity (or rather the probability of traversing one of them).

8.5 Special Purpose Generators

In this section we consider even weaker types of pseudorandom generators, producing sequences that can fool only very restricted types of distinguishers. Still, such generators have many applications in complexity theory and in the design of algorithms. (These applications will only be mentioned briefly.)

We start with the simplest of these generators: the pairwise-independence generator, and its generalization to t-wise independence for any $t \geq 2$. Such generators *perfectly* fool any distinguisher that only observe t locations in the output sequence. This leads naturally to almost pairwise (or t-wise) independence generators, which also fool such distinguishers (albeit non-perfectly). The latter generators are implied by a stronger class of generators, which is of independent interest: the small-bias generators. Small-bias generators fool any linear test (i.e., any distinguisher that merely considers the XOR of some fixed locations in the input sequence). We then turn to the Expander Random Walk Generator: this generator produces a sequence of strings that hit any dense subset of strings with probability that is close to the hitting probability of a truly random sequence. Related notions such as samplers, dispersers, and extractors are treated in Appendix D.

Teaching note: Unlike the constructions presented in previous sections, the constructions presented in this section do not utilize any insight into the nature of (time- or space-bounded) computation. Instead, they are based on various purely mathematical facts, and their analysis is deferred to exercises.

Comment regarding our parameterization: To maintain consistency with prior sections, we continue to present the generators in terms of the seed length, denoted k. Since this is not the common presentation for most results presented in the sequel, we provide (in footnotes) the common presentation in which the seed length is determined as a function of other parameters.

8.5.1 Pairwise-Independence Generators

Pairwise (resp., t-wise) independence generators fool tests that inspect only two (resp., t) elements in the output sequence of the generator. Such local tests are indeed very restricted, yet they arise naturally in many settings. For example, such a test corresponds to a probabilistic analysis (of a procedure) that only relies on the pairwise independence of certain choices made by the procedure. We also mention that, in some natural range of parameters, pairwise independent sampling is as good as sampling by totally independent sample points; see Appendices D.1.2 and D.3.

A t-wise independence generator of block-length $b:\mathbb{N}\to\mathbb{N}$ (and stretch function ℓ) is a relatively efficient deterministic algorithm (e.g., one that works in time polynomial in the output length) that expands a k-bit long random seed into a sequence of $\ell(k)/b(k)$ blocks, each of length $b(k)$, such that any t blocks are uniformly and independently distributed in $\{0,1\}^{t \cdot b(k)}$. That is, denoting the i^{th} block of the generator's output (on seed s) by $G(s)_i$, we requite that for every $i_1 < i_2 < \cdots < i_t$

(in $[\ell(k)/b(k)]$) it holds that

$$G(U_k)_{i_1}, G(U_k)_{i_2}, ..., G(U_k)_{i_t} \equiv U_{t \cdot b(k)}. \tag{8.15}$$

We note that this condition holds even if the inspected t blocks are selected adaptively (see Exercise 8.29). In case $t = 2$, we call the generator pairwise independent.

8.5.1.1 Constructions

In the first construction, we refer to $\mathrm{GF}(2^{b(k)})$, the finite field of $2^{b(k)}$ elements, and associate its elements with $\{0,1\}^{b(k)}$.

Proposition 8.24 (t-wise independence generator):[40] *Let t be a fixed integer and $b, \ell, \ell' : \mathbb{N} \to \mathbb{N}$ such that $b(k) = k/t$, $\ell'(k) = \ell(k)/b(k) > t$ and $\ell'(k) \leq 2^{b(k)}$. Let $\alpha_1, ..., \alpha_{\ell'(k)}$ be fixed distinct elements of the field $\mathrm{GF}(2^{b(k)})$. For $s_0, s_1, ..., s_{t-1} \in \{0,1\}^{b(k)}$, let*

$$G(s_0, s_1, ..., s_{t-1}) \stackrel{\text{def}}{=} \left(\sum_{j=0}^{t-1} s_j \alpha_1^j \, , \, \sum_{j=0}^{t-1} s_j \alpha_2^j \, , ..., \, \sum_{j=0}^{t-1} s_j \alpha_{\ell'(k)}^j \right) \tag{8.16}$$

where the arithmetic is that of $\mathrm{GF}(2^{b(k)})$. Then, G is a t-wise independence generator of block-length b and stretch ℓ.

That is, given a seed that consists of t elements of $\mathrm{GF}(2^{b(k)})$, the generator outputs a sequence of $\ell'(k)$ such elements. To make the foregoing generator totally explicit, we need an explicit representation of $\mathrm{GF}(2^{b(k)})$, which requires an irreducible polynomial of degree $b(k)$ over $\mathrm{GF}(2)$. For specific values of $b(k)$, a good representation does exist: For example, for $d \stackrel{\text{def}}{=} b(k) = 2 \cdot 3^e$ (with e being an integer), the polynomial $x^d + x^{d/2} + 1$ is irreducible over $\mathrm{GF}(2)$. The proof of Proposition 8.24 is left as an exercise (see Exercise 8.30). It is based on the observation that, for any fixed $v_0, v_1, ..., v_{t-1}$, the condition $\{G(s_0, s_1, ..., s_{t-1})_{i_j} = v_j\}_{j=1}^t$ constitutes a system of t linear equations over $\mathrm{GF}(2^{b(k)})$ (in the variables $s_0, s_1, ..., s_{t-1}$) such that the equations are linearly-independent. (Thus, linear independence of certain expressions yields statistical independence of the corresponding random variables.)

We note that a construction analogous to Eq. (8.16) works for every finite field (e.g., a finite field of any prime cardinality), but the problem of providing an explicit representation of such a field remains non-trivial also in other cases (e.g., consider the problem of finding a prime of size approximately $2^{b(k)}$). The latter fact is the main motivation for considering the following alternative construction for the case of $t = 2$.

The following construction uses (random) affine transformations (as possible seeds). In fact, better performance (i.e., shorter seed length) is obtained by using affine transformations affected by Toeplitz matrices. A Toeplitz matrix is a

[40] In the common presentation of this t-wise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and $\ell' \leq 2^b$, the seed length is set to $t \cdot b$.

matrix with all diagonals being homogeneous (see Figure 8.4); that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$ for all i, j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s).

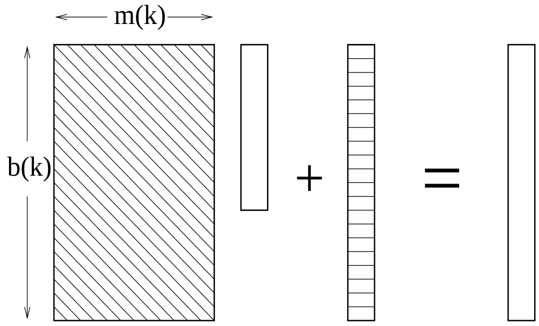


Figure 8.4: An affine transformation affected by a Toeplitz matrix.

Proposition 8.25 (alternative pairwise independence generator, see Figure 8.4):[41] *Let $b, \ell, \ell', m : \mathbb{N} \rightarrow \mathbb{N}$ such that $\ell'(k) = \ell(k)/b(k)$ and $m(k) = \lceil \log_2 \ell'(k) \rceil = k - 2b(k) + 1$. Associate $\{0,1\}^n$ with the n-dimensional vector space over* GF(2), *and let $v_1, ..., v_{\ell'(k)}$ be fixed distinct vectors in the $m(k)$-dimensional vector space over* GF(2). *For $s \in \{0,1\}^{b(k)+m(k)-1}$ and $r \in \{0,1\}^{b(k)}$, let*

$$G(s,r) \stackrel{\text{def}}{=} (T_s v_1 + r\ ,\ T_s v_2 + r\ , ...,\ T_s v_{\ell'(k)} + r) \tag{8.17}$$

where T_s is an $b(k)$-by-$m(k)$ Toeplitz matrix specified by the string s. Then G is a pairwise independence generator of block-length b and stretch ℓ.

That is, given a seed that represents an affine transformation defined by an $b(k)$-by-$m(k)$ Toeplitz matrix and a $b(k)$-dimensional vector, the generator outputs a sequence of $\ell'(k) \leq 2^{m(k)}$ strings, each of length $b(k)$. Note that $k = 2b(k) + m(k) - 1$, and that the stretching property requires $\ell'(k) > k/b(k)$. The proof of Proposition 8.25 is left as an exercise (see Exercise 8.31). This proof is also based on the observation that linear independence of certain expressions yields statistical independence of the corresponding random variables: here $\{G(s,r)_{i_j} = v_j\}_{j=1}^{2}$ is a system of $2b(k)$ linear equations over GF(2) (in Boolean variables representing the bits of s and r) such that the equations are linearly-independent. We mention that a construction analogous to Eq. (8.17) works for every finite field.

A stronger notion of efficient generation. Ignoring the issue of finding a representation for a large finite field, both the foregoing constructions are efficient in the sense that the generator's output can be produced in time that is polynomial

[41] In the common presentation of this pairwise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $2b + \lceil \log_2 \ell' \rceil - 1$.

in its length. Actually, the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed, $s \in \{0,1\}^k$, and a block location $i \in [\ell'(k)]$ (in binary), outputs the i^{th} block of the corresponding output (i.e., the i^{th} block of $G(s)$). Note that, in the case of the first construction (captured by Eq. (8.16)), this stronger notion depends on the ability to find a representation of $\mathrm{GF}(2^{b(k)})$ in $\mathrm{poly}(k)$-time.[42] Recall that this is possible in the case that $b(k)$ is of the form $2 \cdot 3^e$.

8.5.1.2 Applications (a brief review)

Pairwise independence generators do suffice for a variety of applications (cf., [236, 160]). In particular, we mention the application to sampling discussed in Appendix D.3, and the derandomization of the fast parallel algorithm for the Maximal Independent Set problem. This derandomization relies on the fact that the analysis of the randomized algorithm only relies on the hypothesis that some objects are distributed in pairwise independent manner. Thus, this analysis holds also when these objects are selected using a pairwise independence generator. In general, pairwise independence generators do suffice to fool distinguishers that are derived from some natural and interesting randomized algorithms.

Referring to Eq. (8.16), we remark that, for any constant $t \geq 2$, the cost of derandomization (i.e., going over all 2^k possible seeds) is exponential in the block-length (because $b(k) = k/t$). On the other hand, the number of blocks is at most exponential in the block-length (because $\ell'(k) \leq 2^{b(k)}$), and so if a larger number of blocks is needed, then we can artificially increase the block-length in order to accommodate this (i.e., set $(k) = \log_2 \ell'(k)$). Thus, the cost of derandomization is polynomial in $\max(\ell'(k), 2^{b'(k)})$, where $\ell'(k)$ denotes the desired number of blocks and $b'(k)$ the desired block-length. It follows that *whenever the analysis of a randomized algorithm can be based on a constant amount of independence* between feasibly-many random choices, each taken within a domain of feasible size, *then a feasible derandomization is possible.*

8.5.2 Small-Bias Generators

As stated in §8.5.1.2, $O(1)$-wise independence generators allow for the efficient derandomization of any efficient randomized algorithm the analysis of which is only based on a *constant amount of independence* between the bits of its random-tape. This restriction is due to the fact that t-wise independence generators of stretch ℓ require a seed of length $\Omega(t \cdot \log \ell)$. Trying to go beyond constant-independence in such derandomizations (while using seeds of length that is logarithmic in the length of the pseudorandom sequence) was the original motivation of the notion of small-bias generators. Specifically, as we shall see in §8.5.2.2, small-bias generators yield meaningful approximations of t-wise independence sequences (based on logarithmic-length seeds).

[42]For the basic notion of efficiency, it suffices to find a representation of $\mathrm{GF}(2^{b(k)})$ in $\mathrm{poly}(\ell(k))$-time, which can be done by an exhaustive search in the case that $b(k) = O(\log \ell(k))$.

While the aforementioned type of derandomizations remains an important application of small-bias generators, the latter are of independent interest and have found numerous other applications. In particular, small-bias generators fool "global tests" that examine the entire output sequence and not merely a fixed number of positions in it (as in the case of limited independence generators). Specifically, a small-bias generator produces a sequence of bits that fools any linear test (i.e., a test that computes a fixed linear combination of the bits).

For $\varepsilon : \mathbb{N} \to [0,1]$, an ε-bias generator with stretch function ℓ is a relatively efficient deterministic algorithm (e.g., working in poly$(\ell(k))$ time) that expands a k-bit long random seed into a sequence of $\ell(k)$ bits such that for any fixed non-empty set $S \subseteq \{1,...,\ell(k)\}$ the bias of the output sequence over S is at most $\varepsilon(k)$. The bias of a sequence of n (possibly dependent) Boolean random variables $\zeta_1,...,\zeta_n \in \{0,1\}$ over a set $S \subseteq \{1,..,n\}$ is defined as

$$2 \cdot \left| \mathsf{Pr}[\oplus_{i\in S}\zeta_i = 1] - \frac{1}{2} \right| = \left| \mathsf{Pr}[\oplus_{i\in S}\zeta_i = 1] - \mathsf{Pr}[\oplus_{i\in S}\zeta_i = 0] \right|. \qquad (8.18)$$

The factor of 2 was introduced so to make these biases correspond to the Fourier coefficients of the distribution (viewed as a function from $\{0,1\}^n$ to the reals). To see the correspondence replace $\{0,1\}$ by $\{\pm 1\}$, and substitute XOR by multiplication. The bias with respect to a set S is thus written as

$$\left| \mathsf{Pr}\left[\prod_{i\in S}\zeta_i = +1\right] - \mathsf{Pr}\left[\prod_{i\in S}\zeta_i = -1\right] \right| = \left| \mathsf{E}\left[\prod_{i\in S}\zeta_i\right] \right|, \qquad (8.19)$$

which is merely the (absolute value of the) Fourier coefficient corresponding to S.

8.5.2.1 Constructions

Relatively efficient small-bias generators with exponential stretch and exponentially vanishing bias are known.

Theorem 8.26 (small-bias generators):[43] *For some universal constant $c > 0$, let $\ell : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$ such that $\ell(k) \leq \varepsilon(k) \cdot \exp(k/c)$. Then, there exists an ε-bias generator with stretch function ℓ operating in time that is polynomial in the length of its output.*

In particular, we may have $\ell(k) = \exp(k/2c)$ and $\varepsilon(k) = \exp(-k/2c)$. Three simple constructions of small-bias generators that satisfy Theorem 8.26 are known (see [9]). One of these constructions is based on Linear Feedback Shift Registers (LFSRs), where the seed of the generator is used to determine both the "feedback rule" and the "start sequence" of the LFSR. Specifically, a feedback rule of a t-long LFSR is an irreducible polynomial of degree t over GF(2), denoted $f(x) = x^t + \sum_{j=0}^{t-1} f_j x^j$

[43] In the common presentation of this generator, the length of the seed is determined as a function of the desired bias and stretch. That is, given the parameters ε and ℓ, the seed length is set to $c \cdot \log(\ell/\varepsilon)$. We comment that using [9] the constant c is merely 2 (i.e., $k \approx 2\log_2(\ell/\varepsilon)$), whereas using [169] $k \approx \log_2 \ell + 4\log_2(1/\varepsilon)$.

where $f_0 = 1$, and the (ℓ-bit long) sequence produced by the corresponding LFSR based on the start sequence $s_0 s_1 \cdots s_{t-1} \in \{0,1\}^t$ is defined as $r_0 r_1 \cdots r_{\ell-1}$, where

$$r_i = \begin{cases} s_i & \text{if } i \in \{0,1,...,t-1\} \\ \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j} & \text{if } i \in \{t, t+1, ..., \ell-1\} \end{cases} \tag{8.20}$$

(see Figure 8.5). As stated previously, in the corresponding small-bias generator the k-bit long seed is used for selecting an *almost* uniformly distributed feedback rule f (i.e., a random irreducible polynomial of degree $t = k/2$) and a uniformly distributed start sequence s (i.e., a random t-bit string).[44] The corresponding $\ell(k)$-bit long output $r = r_0 r_1 \cdots r_{\ell(k)-1}$ is computed as in Eq. (8.20).

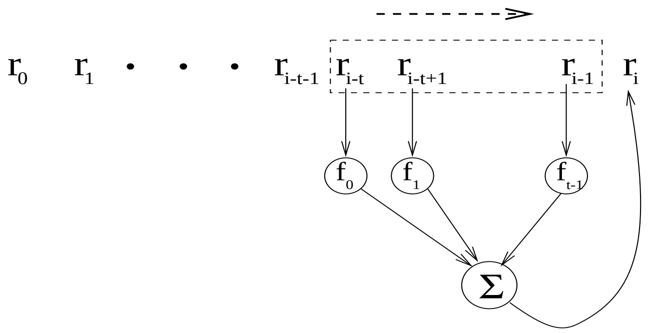


Figure 8.5: The LFSR small-bias generator (for $t = k/2$).

A stronger notion of efficient generation. As in Section 8.5.1.1, we note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. That is, there exists a polynomial-time algorithm that given a k-bit long seed and a bit location $i \in [\ell(k)]$ (in binary), outputs the i^{th} bit of the corresponding output. Specifically, in case of the LFSR construction, given a seed $f_0, ..., f_{(k/2)-1}, s_0, ..., s_{(k/2)-1}$ and a bit location $i \in [\ell(k)]$ (in binary), the algorithm outputs the i^{th} bit of the corresponding output (i.e., r_i).[45]

[44] Note that an implementation of this generator requires an algorithm for selecting an almost random irreducible polynomial of degree $t = \Omega(k)$. A simple algorithm proceeds by enumerating all irreducible polynomials of degree t, and selecting one of them at random. This algorithm can be implemented (using t random bits) in $\exp(t)$-time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A $\text{poly}(t)$-time algorithm that uses $O(t)$ random bits is described in [9, Sec. 8].

[45] The assertion is based on the fact that

$$\begin{pmatrix} r_{i-t+1} \\ r_{i-t+2} \\ \vdots \\ r_{i-1} \\ r_i \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ f_0 & f_1 & f_2 & \cdots & f_{t-1} \end{pmatrix} \begin{pmatrix} r_{i-t} \\ r_{i-t+1} \\ \vdots \\ r_{i-2} \\ r_{i-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ f_0 & f_1 & f_2 & \cdots & f_{t-1} \end{pmatrix}^{i-t+1} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{t-2} \\ s_{t-1} \end{pmatrix}$$

8.5.2.2 Applications (a brief review)

An archetypical application of small-bias generators is for producing short and random "fingerprints" (or "digests") of strings such that equality/inequality among strings is (probabilistically) reflected in equality/inequality between their corresponding fingerprints. The key observation is that checking whether or not $x = y$ is probabilistically reducible to checking whether the inner product modulo 2 of x and r equals the inner product modulo 2 of y and r, *where r is produced by a small-bias generator G*. Thus, the pair (s, v), where s is a random seed to G and v equals the inner product modulo 2 of z and $G(s)$, serves as the randomized fingerprint of the string z. One advantage of this reduction is that only few bits (i.e., the seed of the generator and the result of the inner product) needs to be "communicated between x and y" in order to enable the checking (see Exercise 8.33). A related advantage is the low randomness complexity of this reduction, which uses $|s|$ rather than $|G(s)|$ random bits, where $|s|$ may be $O(\log |G(s)|)$. This low (i.e., logarithmic) randomness-complexity underlies the application of small-bias generators to the construction of PCP systems (see, e.g., §9.3.2.2) and amplifying reductions of gap problems regarding the satisfiability of systems of equations (see Section 9.3.3 and Exercise 10.6).

Small-bias generators have been used in a variety of areas (e.g., inapproximation, structural complexity, and applied cryptography; see references in [89, Sec 3.6.2]). In addition, as shown next, small-bias generators seem an important tool in the design of various types of "pseudorandom" objects.

Approximate independence generators. As hinted at the beginning of this section, small-bias is related to approximate versions of limited independence.[46] Actually, even a restricted type of ε-bias (in which only subsets of size $t(k)$ are required to have bias upper-bounded by ε) implies that any $t(k)$ bits in the said sequence are $2^{t(k)/2} \cdot \varepsilon(k)$-close to $U_{t(k)}$, where here we refer to the variation distance (i.e., Norm-1 distance) between the two distributions. (The max-norm of the difference is bounded by $\varepsilon(k)$.)[47] Combining Theorem 8.26 and the foregoing upper-bound, and relying on the linearity of the construction presented in Proposition 8.24, we obtain generators with double-exponential stretch (i.e., $\ell(k) = \exp(2^{\Omega(k)})$ rather than $\ell(k) = \exp(\Omega(k))$) that are approximately $t(k)$-independent, for some non-constant $t(k)$; see Exercise 8.40. Specifically, we may obtain generators with stretch $\ell(k) = 2^{2^{\Omega(k)}}$ producing bit sequences in which any $t(k) = \Omega(k)$ positions have variation distance at most $\varepsilon(k) = 2^{-\Omega(k)}$ from uniform; that is, such generators may have seed-length $k = O(t(k) + \log(1/\varepsilon(k)) + \log\log \ell(k))$. In the corresponding result for the max-norm distance, it suffices to have $k = O(\log(t(k)/\varepsilon(k)) + \log\log \ell(k))$. Thus, whenever the analysis of a randomized algorithm can be based on a logarithmic amount of (almost) independence between

[46] We warn that, unlike in the case of perfect independence, here we refer only to the distribution on fixed bit locations. See Exercise 8.32 for further discussion.

[47] Both bounds are derived from the Norm2 bound on the difference vector (i.e., the difference between the two probability vectors). For details, see Exercise 8.34.

feasibly-many binary random choices, a feasible derandomization is possible (by using an adequate generator of logarithmic seed length).

Extensions to non-binary choices were considered in various works (see references in [89, Sec 3.6.2]). Some of these works also consider the related problem of constructing small "discrepancy sets" for geometric and combinatorial rectangles.

t-universal set generators. Using the aforementioned upper-bound on the max-norm (of the deviation from uniform of any t locations), any ε-bias generator yields a *t-universal set generator*, provided that $\varepsilon < 2^{-t}$. The latter generator outputs sequences such that in every subsequence of length t all possible 2^t patterns occur (i.e., each for at least one possible seed). Such generators have many applications.

8.5.2.3 Generalization

In this subsection, we outline a generalization of the treatment of small-bias generators to the generation of sequences over an arbitrary finite field. Focusing on the case of a field of prime characteristic, denoted $\mathrm{GF}(p)$, we first define an adequate notion of bias. Generalizing Eq. (8.19), we define the bias of a sequence of n (possibly dependent) random variables $\zeta_1, ..., \zeta_n \in \mathrm{GF}(p)$ with respect to the linear combination $(c_1, ..., c_n) \in \mathrm{GF}(p)^n$ as $\left\| \mathsf{E}\left[\omega^{\sum_{i=1}^n c_i\zeta_i}\right]\right\|$, where ω denotes the p^{th} (complex) root of unity (i.e., $\omega = -1$ if $p = 2$). Referring to Exercise 8.42, we note that upper-bounds on the biases of $\zeta_1, ..., \zeta_n$ (with respect to any non-zero linear combinations) yield upper-bounds on the distance of $\sum_{i=1}^n c_i\zeta_i$ from the uniform distribution over $\mathrm{GF}(p)$.

We say that $S \subseteq \mathrm{GF}(p)^n$ is an ε-bias probability space if a uniformly selected sequence in S has bias at most ε with respect to any non-zero linear combination over $\mathrm{GF}(p)$. (Whenever such a space is efficiently constructible, it yields a corresponding ε-biased generator.) We mention that the LFSR construction, outlined in §8.5.2.1 and analyzed in Exercise 8.36, generalizes to $\mathrm{GF}(p)$ and yields an ε-bias probability space of size (at most) p^{2e}, where $e = \lceil \log_p(n/\varepsilon) \rceil$. Such constructions can be used in applications that generalize those in §8.5.2.2.

8.5.3 Random Walks on Expanders

In this section we review generators that produce a sequence of values by taking a random walk on a large graph that has a small degree but an adequate "mixing" property. Such a graph is called an expander, and by taking a random walk on it we may generate a sequence of ℓ' values over its vertex set, while using a random seed of length $b + (\ell' - 1) \cdot \log_2 d$, where 2^b denotes the number of vertices in the graph and d denotes its degree. This seed length should be compared against the $\ell' \cdot b$ random bits required for generating a sequence of ℓ' independent samples from $\{0,1\}^b$ (or taking a random walk on a clique of size 2^b). Interestingly, as we shall see, the pseudorandom sequence (generated by the said random walk on an expander) *behaves similarly to a truly random sequence with respect to hitting any* *fixed subset of* $\{0,1\}^b$. Let us start by defining this property (or rather by defining the corresponding hitting problem).

Definition 8.27 (the hitting problem): *A sequence of* (possibly dependent) *random variables, denoted* $(X_1,...,X_{\ell'})$*, over* $\{0,1\}^b$ *is* (ε,δ)-hitting *if for any* (target) *set* $T \subseteq \{0,1\}^b$ *of cardinality at least* $\varepsilon \cdot 2^b$*, with probability at least* $1-\delta$*, at least one of these variables hits* T*; that is,* $\Pr[\exists i \text{ s.t. } X_i \in T] \geq 1-\delta$.

Clearly, a truly random sequence of length ℓ' over $\{0,1\}^b$ is (ε,δ)-hitting for $\delta = (1-\varepsilon)^{\ell'}$. The aforementioned "expander random walk generator" (to be described next) achieves similar behavior. Specifically, for arbitrary small $c > 0$ (which depends on the degree and the mixing property of the expander), the generator's output is (ε,δ)-hitting for $\delta = (1-(1-c)\cdot\varepsilon)^{\ell'}$. To describe this generator, we need to discuss expanders.

Expanders. By expander graphs (or expanders) of degree d and eigenvalue bound $\lambda < d$, we actually mean an infinite family of d-regular graphs, $\{G_N\}_{N\in\mathbb{S}}$ ($\mathbb{S} \subseteq \mathbb{N}$), such that G_N is a d-regular graph over N vertices and the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of G_N is upper-bounded by λ. For simplicity, we shall assume that the vertex set of G_N is $[N]$ (although in some cases a somewhat more redundant representation is more convenient). We will refer to such a family as to a (d,λ)-expander (for $\mathbb{S}$). This technical definition is related to the aforementioned notion of "mixing" (which refers to the rate at which a random walk starting at a fixed vertex reaches uniform distribution over the graph's vertices). For further detail, see Appendix E.2.

We are interested in explicit constructions of such graphs, by which we mean that there exists a polynomial-time algorithm that on input N (in binary), a vertex v in G_N and an index $i \in \{1,...,d\}$, returns the i^{th} neighbor of v. (We also require that the set $\mathbb{S}$ for which G_N's exist is sufficiently "tractable" – say that given any $n \in \mathbb{N}$ one may efficiently find an $s \in \mathbb{S}$ such that $n \leq s < 2n$.) Several explicit constructions of expanders are known (see Appendix E.2.2). Below, we rely on the fact that for every $\overline{\lambda} > 0$, there exist d and an explicit construction of a $(d, \overline{\lambda}\cdot d)$-expander over $\{2^b : b \in \mathbb{N}\}$.[48] The relevant (to us) fact about expanders is stated next.

Theorem 8.28 (Expander Random Walk Theorem): *Let* $G = (V,E)$ *be an expander graph of degree* d *and eigenvalue bound* λ*. Let* W *be a subset of* V *and* $\rho \stackrel{\text{def}}{=} |W|/|V|$*, and consider walks on* G *that start from a uniformly chosen vertex and take* $\ell'-1$ *additional random steps, where in each such step one uniformly selects one out of the* d *edges incident at the current vertex and traverses it. Then the probability that such a random walk stays in* W *is at most*

$$\rho \cdot \left(\rho + (1-\rho)\cdot\frac{\lambda}{d}\right)^{\ell'-1} \tag{8.21}$$

[48]This can be obtained with $d = \text{poly}(1/\overline{\lambda})$. In fact $d = O(1/\overline{\lambda}^2)$, which is optimal, can be obtained too, albeit with graphs of sizes that are only approximately close to powers of two.

Thus, a random walk on an expander is "pseudorandom" with respect to the hitting property (i.e., when we consider hitting the set $V \setminus W$ and use $\varepsilon = 1 - \rho$); that is, a set of density ε is hit with probability $1-\delta$, where $\delta = (1-\varepsilon)\cdot(1-\varepsilon+(\lambda/d)\cdot\varepsilon)^{\ell'-1} < (1-(1-(\lambda/d))\cdot\varepsilon)^{\ell'}$. A proof of Theorem 8.28 is given in [134], while a proof of an upper-bound that is weaker than Eq. (8.21) is outlined in Exercise 8.43. Using Theorem 8.28 and an explicit $(2^t, \overline{\lambda}\cdot 2^t)$-expander, we obtain a generator that produces sequences that are (ε,δ)-hitting for δ that is almost optimal.

Proposition 8.29 (The Expander Random Walk Generator):[49]

- *For every constant $\overline{\lambda} > 0$, consider an explicit construction of $(2^t, \overline{\lambda}\cdot 2^t)$-expanders for $\{2^n : n \in \mathbb{N}\}$, where $t \in \mathbb{N}$ is a sufficiently large constant. For $v \in [2^n] \equiv \{0,1\}^n$ and $i \in [2^t] \equiv \{0,1\}^t$, denote by $\Gamma_i(v)$ the vertex of the corresponding 2^n-vertex graph that is reached from vertex v when following its i^{th} edge.*
- *For $b, \ell' : \mathbb{N} \to \mathbb{N}$ such that $k = b(k) + (\ell'(k) - 1)\cdot t < \ell'(k)\cdot b(k)$, and for $v_0 \in \{0,1\}^{b(k)}$ and $i_1, ..., i_{\ell'(k)-1} \in [2^t]$, let*

$$G(v_0, i_1, ..., i_{\ell'(k)-1}) \stackrel{\text{def}}{=} (v_0, v_1, ..., v_{\ell'(k)-1}), \tag{8.22}$$

  where $v_j = \Gamma_{i_j}(v_{j-1})$.

Then G has stretch $\ell(k) = \ell'(k)\cdot b(k)$, and $G(U_k)$ is (ε,δ)-hitting for any $\varepsilon > 0$ and $\delta = (1-(1-\overline{\lambda})\cdot\varepsilon)^{\ell'(k)}$.

The stretch of G is maximized at $b(k) \approx k/2$ (and $\ell'(k) = k/2t$), but maximizing the stretch is not necessarily the goal in all applications. In many applications, the parameters n, ε and δ are given, and the goal is to derive a generator that produces (ε,δ)-hitting sequences over $\{0,1\}^n$ while minimizing both the length of the sequence and the amount of randomness used by the generator (i.e., the seed length). Indeed, Proposition 8.29 suggests using sequences of length $\ell' \approx \varepsilon^{-1}\log_2(1/\delta)$ that are generated based on a random seed of length $n + O(\ell')$.

Expander random-walk generators have been used in a variety of areas (e.g., PCP and inapproximability (see [28, Sec. 11.1]), cryptography (see [90, Sec. 2.6]), and the design of various types of "pseudorandom" objects (see, in particular, Appendix D.3)).

Chapter Notes

Figure 8.6 depicts some of the notions of pseudorandom generators discussed in this chapter. We highlight a key distinction between the case of general-purpose pseudorandom generators (treated in Section 8.2) and the other cases (cf. Sections 8.3 and 8.4): in the former case the distinguisher is more complex than the

[49] In the common presentation of this generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $b + O(\ell' - 1)$.

TYPE	distinguisher's resources	generator's resources	stretch (i.e., $\ell(k)$)	comments
gen.-purpose	$p(k)$-time, $\forall$ poly. p	poly(k)-time	poly(k)	Assumes OW[50]
canon. derandom.	$2^{k/O(1)}$-time	$2^{O(k)}$-time	$2^{k/O(1)}$	Assumes EvEC[50]
space-bounded	$s(k)$-space, $s(k) < k$	$O(k)$-space	$2^{k/O(s(k))}$	runs in time
robustness	$k/O(1)$-space	$O(k)$-space	poly(k)	poly$(k) \cdot \ell(k)$
t-wise independ.	inspect t positions	poly$(k) \cdot \ell(k)$-time	$2^{k/O(t)}$	(e.g., pairwise)
small bias	linear tests	poly$(k) \cdot \ell(k)$-time	$2^{k/O(1)} \cdot \varepsilon(k)$	
expander	"hitting"	poly$(k) \cdot \ell(k)$-time	$\ell'(k) \cdot b(k)$	
random walk	$(0.5, 2^{-\ell'(k)/O(1)})$-hitting for $\{0,1\}^{b(k)}$, with $\ell'(k) = ((k - b(k))/O(1)) + 1$.			

Figure 8.6: Pseudorandom generators at a glance

generator, whereas in the latter cases the generator is more complex than the distinguisher. Specifically, in the general-purpose case the generator runs in (some *fixed*) polynomial-time and needs to withstand *any* probabilistic polynomial-time distinguisher. In fact, some of the proofs presented in Section 8.2 utilize the fact that the distinguisher can invoke the generator on seeds of its choice. In contrast, the Nisan-Wigderson Generator, analyzed in Theorem 8.18 (of Section 8.3), runs more time than the distinguishers that it tries to fool, and the proof relies on this fact in an essential manner. Similarly, the space-complexity of the space-resilient generators presented in Section 8.4 is higher than the space-bound of the distinguishers that they fool.

The general paradigm of pseudorandom generators. Our presentation, which views vastly different notions of pseudorandom generators as incarnations of a general paradigm, has emerged mostly in retrospect. We note that, while the historical study of the various notions was mostly unrelated at a technical level, the case of general-purpose pseudorandom generators served as a source of inspiration to most of the other cases. In particular, the concept of computational indistinguishability, the connection between hardness and pseudorandomness, and the equivalence between pseudorandomness and unpredictability, appeared first in the context of general-purpose pseudorandom generators (and inspired the development of "generators for derandomization" and "generators for space bounded machines"). Indeed, the study of the special-purpose generators (see Section 8.5) was unrelated to all of these.

General-purpose pseudorandom generators. The concept of *computational indistinguishability*, which underlies the entire computational approach to randomness, was suggested by Goldwasser and Micali [107] in the context of defining secure encryption schemes. Indeed, computational indistinguishability plays a key role in cryptography (see Appendix C). The general formulation of computational indistinguishability is due to Yao [237]. Using the hybrid technique of [107], Yao also

[50]By the OW we denote the assumption that one-way functions exists. By EvEC we denote the assumption that the class $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.

observed that defining pseudorandom generators as producing sequences that are computationally indistinguishable from the corresponding uniform distribution is equivalent to defining such generators as producing unpredictable sequences. The latter definition originates in the earlier work of Blum and Micali [39].

Blum and Micali [39] pioneered the rigorous study of pseudorandom generators and, in particular, the construction of pseudorandom generators based on some simple intractability assumption. In particular, they constructed pseudorandom generators assuming the intractability of Discrete Logarithm problem over prime fields. Their work also introduces basic paradigms that were used in all subsequent improvements (cf., e.g., [237, 117]). We refer to the transformation of computational difficulty into pseudorandomness, the use of hard-core predicates (also defined in [39]), and the iteration paradigm (cf. Eq. (8.10)).

Theorem 8.11 (by which pseudorandom generators exist if and only if one-way functions exist) is due to Håstad, Impagliazzo, Levin and Luby [117], building on the hard-core predicate of [98] (see Theorem 7.7). Unfortunately, the current proof of Theorem 8.11 is very complicated and unfit for presentation in a book of the current nature. Presenting a simpler and tighter (cf. §8.2.7.1) proof is indeed an important research project.

Pseudorandom functions (further discussed in Appendix C.3.3) were defined and first constructed by Goldreich, Goldwasser and Micali [94]. We also mention (and advocate) the study of a general theory of pseudorandom objects initiated in [95]. Finally, we mention that a more detailed treatment of general-purpose pseudorandom generators is provided in [90, Chap. 3].

Derandomization of time-complexity classes. As observed by Yao [237], a non-uniformly strong notion of pseudorandom generators yields improved derandomization of time-complexity classes. A key observation of Nisan [172, 175] is that whenever a pseudorandom generator is used in this way, it suffices to require that the generator runs in time that is exponential in its seed length, and so the generator may have running-time greater than the distinguisher (representing the algorithm to be derandomized). This observation motivates the definition of canonical derandomizers as well as the construction of Nisan and Wigderson [172, 175], which is the basis for further improvements culminating in [127]. Part 1 of Theorem 8.19 (i.e., the so-called "high end" derandomization of $\mathcal{BPP}$) is due to Impagliazzo and Wigderson [127], whereas Part 2 (the "low end") is from [175].

The Nisan–Wigderson Generator [175] was subsequently used in several ways transcending its original presentation. We mention its application towards fooling non-deterministic machines (and thus derandomizing constant-round interactive proof systems) and to the construction of randomness extractors [221] (see overview in §D.4.2.2).

In contrast to the aforementioned derandomization results, which place $\mathcal{BPP}$ in some worst-case deterministic complexity class based on some non-uniform (worst-case) assumption, we now mention a result that places $\mathcal{BPP}$ in an average-case deterministic complexity class (cf. Section 10.2) based on a uniform-complexity (worst-case) assumption. We refer specifically to a theorem, which is due to Impagliazzo and Wigderson [128] (but is not presented in the main text), that asserts the following: *if $\mathcal{BPP}$ is not contained in $\mathcal{EXP}$* (almost everywhere) *then $\mathcal{BPP}$ has deterministic sub-exponential time algorithms that are correct on all typical cases* (i.e., with respect to any polynomial-time sampleable distribution).

Pseudorandom with respect to space-bounded distinguishers. As stated in the first paper on the subject of "space-resilient pseudorandom generators" [4][51], this research direction was inspired by the derandomization result obtained via the use of general-purpose pseudorandom generators. The latter result (necessarily) depends on intractability assumptions, and so the objective was identifying natural classes of algorithms for which derandomization is possible without relying on intractability assumptions (but rather by relying on intractability results that are known for the corresponding classes of distinguishers). This objective was achieved before for the case of constant-depth (randomized) circuits, but space-bounded (randomized) algorithms offer a more appealing class that refers to natural algorithms. Fundamentally different constructions of space-resilient pseudorandom generators were given in several works, but are superseded by the two incomparable results mentioned in Section 8.4.2: Theorem 8.21 (a.k.a Nisan's Generator [173]) and Theorem 8.22 (a.k.a the Nisan–Zuckerman Generator [176]). These two results have been "interpolated" in [11]. Theorem 8.23 ($\mathcal{BPL} \subseteq \mathcal{SC}$) was proved by Nisan [174].

Special Purpose Generators. The various generators presented in Section 8.5 were not inspired by any of the other types of pseudorandom generator (nor even by the generic notion of pseudorandomness). Pairwise-independence generator were explicitly suggested in [53] (and are implicit in [49]). The generalization to t-wise independence (for $t \geq 2$) is due to [6]. Small-bias generators were first defined and constructed by Naor and Naor [169], and three simple constructions were subsequently given in [9]. The Expander Random Walk Generator was suggested by Ajtai, Komlos, and Szemerédi [4], who discovered that random walks on expander graphs provide a good approximation to repeated independent attempts with respect to hitting any fixed subset of sufficient density (within the vertex set). The analysis of the hitting property of such walks was subsequently improved, culminating in the bound cited in Theorem 8.28, which is taken from [134, Cor. 6.1].

(The foregoing historical notes do not mention several technical contributions that played an important role in the development of the area. For further details, the reader is referred to [89, Chap. 3]. In fact, the current chapter is a revision of [89, Chap. 3], providing significantly more details for the main topics, and omitting relatively secondary material (a revision of which appears in Appendices D.3 and D.4.)

We mention that an alternative treatment of pseudorandomness, which puts more emphasis on the relation between various techniques, is provided in [228]. In particular, the latter text highlights the connections between information theoretic

[51] This paper is more frequently cited for the Expander Random Walk technique, which it has introduced.

and computational phenomena (e.g., randomness extractors and canonical derandomizers), while the current text tends to decouple the two (see, e.g., Section 8.3 and Appendix D.4).

Exercises

Exercise 8.1 Show that placing no computational requirements on the generator enables unconditional results regarding "generators" that fool any family of subexponential-size circuits. That is, making no computational assumptions, prove that there exist functions $G : \{0,1\}^* \to \{0,1\}^*$ such that $\{G(U_k)\}_{k\in\mathbb{N}}$ is (strongly) pseudorandom, while $|G(s)| = 2|s|$ for every $s \in \{0,1\}^*$. Furthermore, show that G can be computed in double-exponential time.

Guideline: Use the Probabilistic Method (cf. [10]). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probability that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\mathsf{Pr}[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/8}$. Next, using a union bound, prove the existence of a set $S \subset \{0,1\}^n$ of size $2^{n/2}$ such that no circuit of size $2^{n/5}$ can distinguish a uniformly distributed element of S from a uniformly distributed element of $\{0,1\}^n$, where distinguishing means with a probability gap of at least $2^{-n/8}$.

Exercise 8.2 Prove the following corollaries to Proposition 8.3.

1. Let A be a probabilistic polynomial-time algorithm solving a decision problem $\chi : \{0,1\}^* \to \{0,1\}$ (in $\mathcal{BPP}$), and let A_G be as in Construction 8.2. Prove that it is infeasible to find an x on which A_G errs with probability that is significantly higher than the error probability of A; that is, prove that on input 1^n it is infeasible to find an $x \in \{0,1\}^n$ such that $\mathsf{Pr}[A_G(x) \neq \chi(x)] < \mathsf{Pr}[A(x) = \chi(x)] + 0.01$.

2. Let A be a probabilistic polynomial-time algorithm solving the search associated with the NP-relation R, and let A_G be as in Construction 8.2. Prove that it is infeasible to find an x on which A_G outputs a wrong solution; that is, assuming for simplicity that A has error probability $1/3$, prove that on input 1^n it is infeasible to find an $x \in \{0,1\}^n \cap S_R$ such that $\mathsf{Pr}[(x, A_G(x)) \notin R] > 0.4$, where $S_R \stackrel{\text{def}}{=} \{x : \exists y\,(x,y) \in R\}$. Likewise, it is infeasible to find an $x \in \{0,1\}^n \setminus S_R$ such that $\mathsf{Pr}[A_G(x) \neq \perp] > 0.4$.

Exercise 8.3 Prove that omitting the absolute value in Eq. (8.6) keeps Definition 8.4 intact.
(Hint: consider $D'(z) \stackrel{\text{def}}{=} 1 - D(z)$.)

Exercise 8.4 Prove that computational indistinguishability is an equivalence relation (defined over pair of probability ensembles). Specifically, prove that this relation is transitive (i.e., $X \equiv Y$ and $Y \equiv Z$ implies $X \equiv Z$).

Exercise 8.5 Prove that *if* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are computationally indistinguishable and* A *is a probabilistic polynomial-time algorithm then* $\{A(X_n)\}_{n\in\mathbb{N}}$ *and* $\{A(Y_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*

Guideline: If D distinguishes the latter ensembles, then D' such that $D'(z) \stackrel{\text{def}}{=} D(A(z))$ distinguishes the former.

Exercise 8.6 In contrast to Exercise 8.5, show that the conclusion may not hold in case A is not computationally bounded. That is, show that there exists computationally indistinguishable ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, and an exponential-time algorithm A such that $\{A(X_n)\}_{n\in\mathbb{N}}$ and $\{A(Y_n)\}_{n\in\mathbb{N}}$ are *not* computationally indistinguishable.

Guideline: For any pair of ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, consider the Boolean function f such that $f(z) = 1$ if and only if $\mathsf{Pr}[X_n = z] > \mathsf{Pr}[Y_n = z]$. Show that $|\mathsf{Pr}[f(X_n) = 1] - \mathsf{Pr}[f(Y_n) = 1]|$ equals the statistical difference between X_n and Y_n. Consider an adequate (approximate) implementation of f (e.g., approximate $\mathsf{Pr}[X_n = z]$ and $\mathsf{Pr}[Y_n = z]$ up to $\pm 2^{-2|z|}$).

Exercise 8.7 Show that the existence of pseudorandom generators implies the existence of polynomial-time constructible probability ensembles that are statistically far apart and yet are computationally indistinguishable.

Guideline: Lower-bound the statistical distance between $G(U_k)$ and $U_{\ell(k)}$, where G is a pseudorandom generator with stretch ℓ.

Exercise 8.8 Relying on Theorem 7.7, provide a self-contained proof of the fact that the existence of one-way 1-1 functions implies the existence of polynomial-time constructible probability ensembles that are statistically far apart and yet are computationally indistinguishable.

Guideline: Assuming that b is a hard-core of the function f, consider the ensembles $\{f(U_n)\cdot b(U_n)\}_{n\in\mathbb{N}}$ and $\{f(U_n)\cdot U_1'\}_{n\in\mathbb{N}}$. Prove that these ensembles are computationally indistinguishable by using the main ideas of the proof of Proposition 8.9. Show that if f is 1-1 then these ensembles are statistically far apart.

Exercise 8.9 (following [87]) Prove that the sufficient condition in Exercise 8.7 is in fact necessary. Recall that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are said to be statistically far apart if, for some positive polynomial p and all sufficiently large n, the variation distance between X_n and Y_n is greater than $1/p(n)$. Using the following three steps, prove that the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable implies the existence of pseudorandom generators.

1. Show that, without loss of generality, we may assume that the variation distance between X_n and Y_n is greater than $1 - \exp(-n)$.

   Guideline: For X_n and Y_n as in the forgoing, consider $\overline{X}_n = (X_n^{(1)}, ..., X_n^{(t(n))})$ and $\overline{Y}_n = (Y_n^{(1)}, ..., Y_n^{(t(n))})$, where the $X_n^{(i)}$'s (resp., $Y_n^{(i)}$'s) are independent copies of X_n (resp., Y_n), and $t(n) = O(n \cdot p(n)^2)$. To lower-bound the statistical difference between $\overline{X}_n$ and $\overline{Y}_n$, consider the set $S_n \stackrel{\rm def}{=} \{z : \Pr[X_n = z] > \Pr[Y_n = z]\}$ and the random variable representing the number of copies in $\overline{X}_n$ (resp., $\overline{Y}_n$) that reside in S_n.

2. Using $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ as in Step 1, prove the existence of a *false entropy generator*, where a false entropy generator is a deterministic polynomial-time algorithm G such that $G(U_k)$ has entropy $e(k)$ but $\{G(U_k)\}_{k\in\mathbb{N}}$ is computationally indistinguishable from a polynomial-time constructible ensemble that has entropy greater than $e(\cdot) + (1/2)$.

   Guideline: Let S_0 and S_1 be sampling algorithms such that $X_n \equiv S_0(U_{{\rm poly}(n)})$ and $Y_n \equiv S_1(U_{{\rm poly}(n)})$. Consider the generator $G(\sigma, r) = (\sigma, S_\sigma(r))$, and the distribution Z_n that equals (U_1, X_n) with probability 1/2 and (U_1, Y_n) otherwise. Note that in $G(U_1, U_{{\rm poly}(n)})$ the first bit is almost determined by the rest, whereas in Z_n the first bit is statistically independent of the rest.

3. Using a false entropy generator, obtain one in which the excess entropy is $\sqrt{k}$, and using the latter construct a pseudorandom generator.

   Guideline: Use the ideas presented in §8.2.5.3 (i.e., the discussion of the interesting direction of the proof of Theorem 8.11).

Exercise 8.10 (multiple samples vs single sample, a separation) In contrast to Proposition 8.6, prove that there exist two probability ensembles that are computational indistinguishable by a single sample, but are efficiently distinguishable by two samples. Furthermore, one of these ensembles is the uniform ensemble and the other has a sparse support (i.e., only poly(n) many strings are assigned a non-zero probability weight by the second distribution). Indeed, the second ensemble is not polynomial-time constructible.

Guideline: Prove that, for every function $d : \{0,1\}^n \to [0,1]$, there exists two strings, x_n and y_n (in $\{0,1\}^n$), and a number $p \in [0,1]$ such that $\Pr[d(U_n)=1] = p \cdot \Pr[d(x_n)=1] + (1-p) \cdot \Pr[d(y_n)=1]$. Generalize this claim to m functions, using $m+1$ strings and a convex combination of the corresponding probabilities.[52] Conclude that there exists a distribution Z_n with a support of size at most $m+1$ such that for each of the first (in lexicographic order) m (randomized) algorithms A it holds that $\Pr[A(U_n)=1] = \Pr[A(Z_n)=1]$. Note that with probability at least $1/(m+1)$, two independent samples of Z_n are assigned the same value, yielding a simple two-sample distinguisher of U_n from Z_n.

Exercise 8.11 (amplifying the stretch function, an alternative construction) For G_1 and ℓ as in Construction 8.7, consider $G(s) \stackrel{\rm def}{=} G_1^{\ell(|s|)-|s|}(s)$, where $G_1^i(x)$ denotes G_1 iterated i times on x (i.e., $G_1^i(x) = G_1^{i-1}(G_1(x))$ and $G_1^0(x) = x$). Prove that G is a pseudorandom generator of stretch ℓ. Reflect on the advantages of Construction 8.7 over the current construction (e.g., consider generation time).

[52] That is, prove that for every m functions $d_1, ..., d_m : \{0,1\}^n \to [0,1]$ there exist $m+1$ strings $z_n^{(1)}, ..., z_n^{(m+1)}$ and $m+1$ non-negative numbers $p_1, ..., p_{m+1}$ that sum-up to 1 such that for every $i \in [m]$ it holds that $\Pr[d_i(U_n)=1] = \sum_j p_j \cdot \Pr[d_i(z_n^{(j)})=1]$.

Guideline: Use a hybrid argument, with the i^{th} hybrid being $G_1^i(U_{\ell(k)-i})$, for $i = 0,...,\ell(k)-k$. Note that $G_1^{i+1}(U_{\ell(k)-(i+1)}) = G_1^i(G_1(U_{\ell(k)-i-1}))$ and $G_1^i(U_{\ell(k)-i}) = G_1^i(U_{|G_1(U_{\ell(k)-i-1})|})$, and use Exercise 8.5.

Exercise 8.12 (pseudorandom versus unpredictability) Prove that a probability ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is pseudorandom if and only if it is unpredictable. For simplicity, we say that $\{Z_k\}_{k\in\mathbb{N}}$ is (next-bit) unpredictable if for every probabilistic polynomial-time algorithm A it holds that $\mathsf{Pr}_i[A(F_i(Z_k)) = B_{i+1}(Z_k)] - (1/2)$ is negligible, where $i \in \{0,...,|Z_k|-1\}$ is uniformly distributed, and $F_i(z)$ (resp., $B_{i+1}(z)$) denotes the i-bit prefix (resp., $i+1^{\text{st}}$ bit) of z.

Guideline: Show that pseudorandomness implies polynomial-time unpredictability; that is, polynomial-time predictability violates pseudorandomness (because the uniform ensemble is unpredictable regardless of computing power). Use a hybrid argument to prove that unpredictability implies pseudorandomness. Specifically, the i^{th} hybrid consists of the i-bit long prefix of Z_k followed by $|Z_k|-i$ uniformly distributed bits. Thus, distinguishing the extreme hybrids (which correspond to Z_k and $U_{|Z_k|}$) implies distinguishing a random pair of neighboring hybrids, which in turn implies next-bit predictability. For the last step, use an argument as in the proof of Proposition 8.9.

Exercise 8.13 Prove that a probability ensemble is unpredictable (from left to right) if and only if it is unpredictable from right to left (or in any other canonical order).

Guideline: Use Exercise 8.12, and note that an ensemble is pseudorandom if and only if its reverse is pseudorandom.

Exercise 8.14 Let f be 1-1 and length preserving, and b be a hard-core predicate of f. For any polynomial ℓ, letting $G'(s) \stackrel{\text{def}}{=} b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$, prove that $\{G'(U_k)\}$ is unpredictable (in the sense of Exercise 8.12).

Guideline: Suppose towards the contradiction that, for a uniformly distributed $j \in \{0,...,\ell(k)-1\}$, given the j-bit long prefix of $G'(U_k)$ an algorithm A' can predict the $j+1^{\text{st}}$ bit of $G'(U_k)$. That is, given $b(f^{\ell(k)-1}(s)) \cdots b(f^{\ell(k)-j}(s))$, algorithm A' predicts $b(f^{\ell(k)-(j+1)}(s))$, where s is uniformly distributed in $\{0,1\}^k$. Consider an algorithm A that given $y = f(x)$ approximates $b(x)$ by invoking A' on input $b(f^{j-1}(y)) \cdots b(y)$, where j is uniformly selected in $\{0,...,\ell(k)-1\}$. Analyze the success probability of A using the fact that f induces a permutation over $\{0,1\}^n$, and thus $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$ is distributed identically to $b(f^{\ell(k)-1}(U_k)) \cdots b(f^{\ell(k)-j}(U_k)) \cdot b(f^{\ell(k)-(j+1)}(U_k))$.

Exercise 8.15 Prove that if G is a strong pseudorandom generator in the sense of Definition 8.12 then it a pseudorandom generator in the sense of Definition 8.1.

Guideline: Consider a sequence of internal coin tosses that maximizes the probability in Eq. (8.2).

Exercise 8.16 (strong computational indistinguishability) Provide a definition of the notion of computational indistinguishability that underlies Definition 8.12 (i.e., indistinguishability with respect to (non-uniform) polynomial-size circuits). Prove the following two claims:

1. Computational indistinguishability with respect to (non-uniform) polynomial-size circuits is strictly stronger than Definition 8.4.

2. Computational indistinguishability with respect to (non-uniform) polynomial-size circuits is invariant under (polynomially-many) multiple samples, even if the underlying ensembles are not polynomial-time constructible.

Guideline: For Part 1, see the solution to Exercise 8.10. For Part 2 note that samples as generated in the proof of Proposition 8.6 can be hard-wired into the distinguishing circuit.

Exercise 8.17 Show that Construction 8.7 may fail in the context of canonical derandomizers. Specifically, prove that it fails for the canonical derandomizer G' that is presented in the proof of Theorem 8.18.

Exercise 8.18 In relation to Definition 8.14 (and assuming $\ell(k) > k$), show that there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (8.11).

Guideline: The circuit may incorporate all values in the range of G and decide by comparing its input to these values.

Exercise 8.19 (constructing a set system for Theorem 8.18) For every $\gamma > 0$, show a construction of a set system S as in Condition 2 of Theorem 8.18, with $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$.

Guideline: We assume, without loss of generality, that $\gamma < 1$, and set $m(k) = (\gamma/2) \cdot k$ and $\ell(k) = 2^{\gamma m(k)/6}$. We construct the set system $S_1, ..., S_{\ell(k)}$ in iterations, selecting S_i as the first $m(k)$-subset of $[k]$ that has sufficiently small intersections with each of the previous sets $S_1, ..., S_{i-1}$. The existence of such a set S_i can be proved using the Probabilistic Method (cf. [10]). Specifically, for a fixed $m(k)$-subset S', the probability that a random $m(k)$-subset has intersection greater than $\gamma m(k)$ with S' is smaller than $2^{-\gamma m(k)/6}$, because the expected intersection size is $(\gamma/2) \cdot m(k)$. Thus, with positive probability a random $m(k)$-subset has intersection at most $\gamma m(k)$ with each of the previous $i-1 < \ell(k) = 2^{\gamma m(k)/6}$ subsets. Note that we construct S_i in time $\binom{k}{m(k)} \cdot (i-1) \cdot m(k) < 2^k \cdot \ell(k) \cdot k$, and thus S is computable in time $k2^k \cdot \ell(k)^2 < 2^{2k}$.

Exercise 8.20 (pseudorandom versus unpredictability, by circuits) In continuation to Exercise 8.12, show that if there exists a circuit of size s that distinguishes Z_n from U_ℓ with gap δ, then there exists an $i < \ell = |Z_n|$ and a circuit of size $s + O(1)$ that given an i-bit long prefix of Z_n guesses the $i + 1^{\text{st}}$ bit with success probability at least $\frac{1}{2} + \frac{\delta}{\ell}$.

Guideline: Defining hybrids as in Exercise 8.12, note that, for some i, the given circuit distinguishes the i^{th} hybrid from the $i + 1^{\text{st}}$ hybrid with gap at least δ/ℓ.

Exercise 8.21 Suppose that the sets S_i's in Construction 8.17 are disjoint and that $f : \{0,1\}^m \to \{0,1\}$ is T-inapproximable. Prove that for every circuit C of size $T - O(1)$ it holds that $|\mathsf{Pr}[C(G(U_k)) = 1] - \mathsf{Pr}[C(U_\ell) = 1]| < \ell/T$.

Guideline: Prove the contrapositive using Exercise 8.20. Note that the value of the $i+1^{\rm st}$ bit of $G(U_k)$ is statistically independent of the values of the first i bits of $G(U_k)$, and thus predicting it yields an approximator for f. Indeed, such an approximator can be obtained by fixing the the first i bits of $G(U_k)$ via an averaging argument.

Exercise 8.22 (Theorem 8.18, generalized) Let $\ell, m, m', T : \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k)^2 + \widetilde{O}(\ell(k)2^{m'(k)}) < T(m(k))$. Suppose that the following two conditions hold:

1. There exists an exponential-time computable function $f:\{0,1\}^* \to \{0,1\}$ that is T-inapproximable.
2. There exists an exponential-time computable function $S:\mathbb{N}{\times}\mathbb{N} \to 2^{\mathbb{N}}$ such that for every k and $i = 1, ..., \ell(k)$ it holds that $S(k,i) \subseteq [k]$ and $|S(k,i)| = m(k)$, and $|S(k,i) \cap S(k,j)| \leq m'(k)$ for every k and $i \neq j$.

Prove that using G as defined in Construction 8.17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.

Guideline: Following the proof of Theorem 8.18, just note that the circuit constructed for approximating $f(U_{m(k)})$ has size $\ell(k)^2 + \ell(k) \cdot \widetilde{O}(2^{m'(k)})$ and success probability at least $(1/2) + (1/7\ell(k))$.

Exercise 8.23 (Part 2 of Theorem 8.19) Prove that if for every polynomial T there exists a T-inapproximable predicate in $\mathcal{E}$ then $\mathcal{BPP} \subseteq \cap_{\varepsilon>0}\mathrm{D}\textsc{time}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\rm def}{=} 2^{n^\varepsilon}$.

Guideline: Using Proposition 8.15, it suffices to present, for every polynomial p and every constant $\varepsilon > 0$, a canonical derandomizer of stretch $\ell(k) = p(k^{1/\varepsilon})$. Such a derandomizer can be presented by applying Exercise 8.22 using $m(k) = \sqrt{k}$, $m'(k) = O(\log k)$, and $T(m(k)) = \ell(k)^2 + \widetilde{O}(\ell(k)2^{m'(k)})$. Note that T is a polynomial, revisit Exercise 8.19 in order to obtain a set system as required in Exercise 8.22 (for these parameters), and use Theorem 7.10.

Exercise 8.24 (canonical derandomizers imply hard problems) Prove that the hardness hypothesis made in each part of Theorem 8.19 is essential for the existence of a corresponding canonical derandomizer. More generally, prove that the existence of a canonical derandomizer with stretch ℓ implies the existence of a predicate in $\mathcal{E}$ that is T-inapproximable for $T(n) = \ell(n)^{1/O(1)}$.

Guideline: We focus on obtaining a predicate in $\mathcal{E}$ that cannot be computed by circuits of size ℓ, and note that the claim follows by applying the techniques in §7.2.1.3. Given a canonical derandomizer $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$, we consider the predicate $f : \{0,1\}^{k+1} \to \{0,1\}$ that satisfies $f(x) = 1$ if and only if there exists $s \in \{0,1\}^{|x|-1}$ such that x is a prefix of $G(s)$. Note that f is in $\mathcal{E}$ and that an algorithm computing f yields a distinguisher of $G(U_k)$ and $U_{\ell(k)}$.

Exercise 8.25 (limitations on the stretch of (s,ε)-pseudorandom generators) Referring to Definition 8.20, establish the following upper-bounds on the stretch ℓ of (s,ε)-pseudorandom generators.

1. If $s(k) \geq 2$ and $\varepsilon(k) \leq 1/2$ then $\ell(k) < \varepsilon(k) \cdot (k+2) \cdot 2^{k+2-s(k)}$.
2. For every $s(k) \geq 1$ and $\varepsilon(k) < 1$ it holds that $\ell(k) < 2^k$.

Guideline: Part 2 follows by combining Exercises 8.37 and 8.38. For Part 1, consider towards the contradiction a generator of stretch $\ell(k) = \varepsilon(k) \cdot (k+2) \cdot 2^{k+2-s(k)}$ and an enumeration, $\alpha^{(1)}, ..., \alpha^{(2^k)} \in \{0,1\}^{\ell(k)}$, of all 2^k outputs of the generator (on k-bit long seeds). Construct a non-uniform automaton of space s that accepts $x_1 \cdots x_{\ell(k)} \in \{0,1\}^{\ell(k)}$ if for some $i \in [\ell(k)/(k+2)]$ it holds that $x_{(i-1)\cdot(k+2)+1} \cdots x_{i\cdot(k+2)}$ equals some string in S_i, where S_i contains the projection of the strings $\alpha^{((i-1)\cdot 2^{s(k)-1}+1)}, ..., \alpha^{(i \cdot 2^{s(k)-1})}$ on the coordinates $(i-1)\cdot(k+2)+1, ..., i\cdot(k+2)$. Note that such an automaton accepts at least $(\ell(k)/(k+2)) \cdot 2^{s(k)-1} = 2\varepsilon(k) \cdot 2^k$ of the possible outputs of the generator, whereas a random ($\ell(k)$-bit long) string is accepted with probability at most $(\ell(k)/(k+2)) \cdot 2^{(s(k)-1)-(k+2)} = \varepsilon(k)/2$.

Exercise 8.26 (on the existence of (s,ε)-pseudorandom generators) In contrast to Exercise 8.25, for any s and ε such that $s(k) < k - 2\log_2(k/\varepsilon(k)) - O(1)$, prove the existence of (non-efficient) (s,ε)-pseudorandom generators of stretch $\ell(k) = \Omega(\varepsilon(k)^2 \cdot 2^{k-s(k)}/s(k))$.

Guideline: Use the Probabilistic Method as in Exercise 8.1. Note that non-uniform automata of space s and time ℓ can be described by strings of length $\ell \cdot 2s2^s$.

Exercise 8.27 (multiple samples and space-bounded distinguishers) Suppose that two probability ensembles, $\{X_k\}_{k\in\mathbb{N}}$ and $\{Y_k\}_{k\in\mathbb{N}}$, are (s,ε)-indistinguishable by non-uniform automata (i.e., the distinguishability-gap of any non-uniform automaton of space s is bounded by the function ε). For any function $t:\mathbb{N}\rightarrow\mathbb{N}$, prove that the ensembles $\{(X_k^{(1)}, ..., X_k^{(t(k))})\}_{k\in\mathbb{N}}$ and $\{(Y_k^{(1)}, ..., X_k^{(t(k))})\}_{k\in\mathbb{N}}$ are $(s, t\varepsilon)$-indistinguishable, where $X_k^{(1)}$ through $X_k^{(t(k))}$ and $Y_k^{(1)}$ through $Y_k^{(t(k))}$ are independent random variables, with each $X_k^{(i)}$ identical to X_k and each $Y_k^{(i)}$ identical to Y_k.

Guideline: Use the hybrid technique. When distinguishing the i^{th} and $(i+1)^{\text{st}}$ hybrids, note that the first i blocks (i.e., copies of X_k) as well as the last $t(k)-(i+1)$ blocks (i.e., copies of Y_k) can be fixed and hard-wired into the non-uniform distinguisher.

Exercise 8.28 Provide a more explicit description of the generator outlined in the proof of Theorem 8.21.

Guideline: for $r \in \{0,1\}^n$ and $h^{(1)}, ..., h^{(t)} \in H_n$, the generator outputs a 2^t-long sequence of n-bit strings such that the i^{th} string in this sequence equals $h'(r)$, where h' is a composition of some of the $h^{(j)}$'s.

Exercise 8.29 (adaptive t-wise independence tests) Recall that a generator $G:\{0,1\}^k \rightarrow \{0,1\}^{\ell'(k)\cdot b(k)}$ is called t-wise independent if *for any t fixed block positions*, the distribution $G(U_k)$ restricted to these t blocks is uniform over $\{0,1\}^{t\cdot b(k)}$. Prove that the output of a t-wise independence generator is (perfectly) indistinguishable from the uniform distribution *by any test that examines t of the blocks*, *even if the examined blocks are selected adaptively* (i.e., the location of the i^{th} block to be examined is determined based on the contents of the previously inspected blocks).

Guideline: First show that, without loss of generality, it suffices to consider deterministic (adaptive) testers. Next, show that the probability that such a tester sees any fixed sequence of t values at the locations selected *adaptively* (in the generator's output) equals $2^{-t \cdot b(k)}$, where $b(k)$ is the block length.

Exercise 8.30 (a t-wise independence generator) Prove that G as defined in Proposition 8.24 produces a t-wise independent sequence over $\mathrm{GF}(2^{b(k)})$.

Guideline: For every t fixed indices $i_1, ..., i_t \in [\ell'(k)]$, consider the distribution of $G(U_k)_{i_1,...,i_t}$ (i.e., the projection of $G(U_k)$ on locations $i_1, ..., i_t$). Show that for every sequence of t possible values $v_1, ..., v_t \in \mathrm{GF}(2^{b(k)})$, there exists a unique seed $s \in \{0,1\}^k$ such that $G(s)_{i_1,...,i_t} = (v_1, ..., v_t)$.

Exercise 8.31 (pairwise independence generators) As a warm-up, consider a construction analogous to the one in Proposition 8.25, except that here the seed specifies an arbitrary affine $b(k)$-by-$m(k)$ transformation. That is, for $s \in \{0,1\}^{b(k) \cdot m(k)}$ and $r \in \{0,1\}^{b(k)}$, where $k = b(k) \cdot m(k) + b(k)$, let

$$G(s, r) \stackrel{\text{def}}{=} (A_s v_1 + r\,,\, A_s v_2 + r\,, ...,\, A_s v_{\ell'(k)} + r) \tag{8.23}$$

where A_s is an $b(k)$-by-$m(k)$ matrix specified by the string s. Show that G as in Eq. (8.23) is a pairwise independence generator of block-length b and stretch ℓ. (Note that a related construction appears in the proof of Theorem 7.7; see also Exercise 7.5.) Next, show that G as in Eq. (8.17) is a pairwise independence generator of block-length b and stretch ℓ.

Guideline: The following description applies to both constructions. First note that for every fixed $i \in [\ell'(k)]$, the i^{th} element in the sequence $G(U_k)$, denoted $G(U_k)_i$, is uniformly distributed in $\{0,1\}^{b(k)}$. Actually, show that for every fixed $s \in \{0,1\}^{k-b(k)}$, it holds that $G(s, U_{b(k)})_i$ is uniformly distributed in $\{0,1\}^{b(k)}$. Next note that it suffices to show that, for every $j \neq i$, conditioned on the value of $G(U_k)_i$, the value of $G(U_k)_j$ is uniformly distributed in $\{0,1\}^{b(k)}$. The key technical detail is showing that, for any non-zero vector $v \in \{0,1\}^{m(k)}$ and a uniformly selected $s \in \{0,1\}^{k-b(k)}$, it holds that $A_s v$ (resp., $T_s v$) is uniformly distributed in $\{0,1\}^{b(k)}$. This is easy in case of a random $b(k)$-by-$m(k)$ matrix, and can be proven also for a random Toeplitz matrix.

Exercise 8.32 (adaptive t-wise independence tests, revisited) Note that in contrast to Exercise 8.29, with respect to *non-perfect* indistinguishability, there is a discrepancy between adaptive and non-adaptive tests that inspects t locations.

1. Present a distribution over 2^{t-1}-bit long strings in which every t fixed bit positions are $t \cdot 2^{-t}$-close to uniform, but there exists a test that adaptively inspects t positions and distinguish this distribution from the uniform one with gap $1/2$.

Guideline: Modify the uniform distribution over $((t-1)+2^{t-1})$-bit long strings such that the first $t-1$ locations indicate a bit position (among the rest) that is set to zero.

2. On the other hand, prove that if every t fixed bit positions in a distribution X are ε-close to uniform, then every test that adaptively inspects t positions can distinguish X the uniform distribution with gap at most $2^t \cdot \varepsilon$.

   Guideline: See Exercise 8.29.

Exercise 8.33 Suppose that G is an ε-bias generator with stretch ℓ. Show that equality between the $\ell(k)$-bit strings x and y can be probabilistically checked (with error probability $(1+\varepsilon)/2$) by comparing the inner product modulo 2 of x and $G(s)$ to the inner product modulo 2 of y and $G(s)$, where $s \in \{0,1\}^k$ is selected uniformly.
(Hint: reduce the problem to the special case in which $y = 0^{\ell(k)}$.)

Exercise 8.34 (bias versus statistical difference from uniform) Let X be a random variable assuming values in $\{0,1\}^t$. Prove that if X has bias at most ε over any non-empty set then the statistical difference between X and U_t is at most $2^{t/2} \cdot \varepsilon$, and that for every $x \in \{0,1\}^t$ it holds that $\mathsf{Pr}[X = x] = 2^{-t} \pm \varepsilon$.

Guideline: Consider the probability function $p : \{0,1\}^t \to [0,1]$ defined by $p(x) \stackrel{\text{def}}{=} \mathsf{Pr}[X = x]$, and let $\delta(x) \stackrel{\text{def}}{=} p(x) - 2^{-t}$ denote the deviation of p from the uniform probability function. Viewing the set of real functions over $\{0,1\}^t$ as a 2^t-dimensional vector space, consider two orthonormal bases for this space. The first basis consists of the (Kroniker) functions $\{k_\alpha\}_{\alpha\in\{0,1\}^t}$ such that $k_\alpha(x) = 1$ if $x = \alpha$ and $k_\alpha(x) = 0$ otherwise. The second basis consists of the (normalize Fourier) functions $\{f_S\}_{S\subseteq[t]}$ defined by $f_S(x_1\cdots x_t) \stackrel{\text{def}}{=} 2^{-t/2}\prod_{i\in S}(-1)^{x_i}$ (where $f_\emptyset \equiv 2^{-t/2}$).[53] Note that the bias of X over any $S \neq \emptyset$ equals $|\sum_x p(x)\cdot 2^{t/2} f_S(x)|$, which in turn equals $2^{t/2}|\sum_x \delta(x) f_S(x)|$. Thus, for every S (including the empty set), we have $|\sum_x \delta(x) f_S(x)| \leq 2^{-t/2}\varepsilon$, which means that the representation of δ in the normalize Fourier basis is by coefficients that have each an absolute value of at most $2^{-t/2}\varepsilon$. It follows that the Norm-2 of this vector of coefficients is upper-bounded by $\sqrt{2^t\cdot(2^{-t/2}\varepsilon)^2} = \varepsilon$, and the two claims follow by noting that they refer to norms of δ according to the Kroniker basis. In particular, Norm-2 is preserved under orthonormal bases, the max-norm is upper-bounded by Norm-2, and Norm-1 is upper-bounded by $\sqrt{2^t}$ times the value of the Norm-2.

Exercise 8.35 (on the existence of (non-explicit) small-bias generators) Prove that, for $k = \log_2(\ell(k)/\varepsilon(k)^2) + O(1)$, there exists a function $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ such that $G(U_k)$ has bias at most $\varepsilon(k)$ over any non-empty subset of $[\ell(k)]$.

Guideline: Use the Probabilistic Method as in Exercise 8.1.

[53] Verify that both bases are indeed orthogonal (i.e., $\sum_x k_\alpha(x)k_\beta(x) = 0$ for every $\alpha \neq \beta$ and $\sum_x f_S(x) f_T(x) = 0$ for every $S \neq T$) and normal (i.e., $\sum_x k_\alpha(x)^2 = 1$ and $\sum_x f_S(x)^2 = 1$).

Exercise 8.36 (The LFSR small-bias generator (following [9])) Using the following guidelines (and letting $t = k/2$), analyze the construction outlined following Theorem 8.26 (and depicted in Figure 8.5):

1. Prove that r_i equals $\sum_{j=0}^{t-1} c_j^{(f,i)} \cdot s_j$, where $c_j^{(f,i)}$ is the coefficient of z^j in the (degree $t-1$) polynomial obtained by reducing z^i modulo the polynomial $f(z)$ (i.e., $z^i \equiv \sum_{j=0}^{t-1} c_j^{(f,i)} z^j \pmod{f(z)}$).

   Guideline: Recall that $z^t \equiv \sum_{j=0}^{t-1} f_j z^j \pmod{f(z)}$, and thus for every $i \geq t$ it holds that $z^i \equiv \sum_{j=0}^{t-1} f_j z^{i-t+j} \pmod{f(z)}$. Note the correspondence to $r_i = \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j}$.

2. For any non-empty $S \subseteq \{0, ..., \ell(k)-1\}$, evaluate the bias of the sequence $r_0, ..., r_{\ell(k)-1}$ over S, where f is a random irreducible polynomial of degree t and $s = (s_0, ..., s_{t-1}) \in \{0,1\}^t$ is uniformly distributed. Specifically:

   (a) For a fixed f and random $s \in \{0,1\}^t$, prove that $\sum_{i \in S} r_i$ has non-zero bias if and only if $f(z)$ divides $\sum_{i \in S} z^i$.

   (Hint: Note that $\sum_{i \in S} r_i = \sum_{j=0}^{t-1} \sum_{i \in S} c_j^{(f,i)} s_j$, and use Item 1.)

   (b) Prove that the probability that a random irreducible polynomial of degree t divides $\sum_{i \in S} z^i$ is $\Theta(\ell(k)/2^t)$.

   (Hint: A polynomial of degree n can be divided by at most n/d different irreducible polynomials of degree d. On the other hand, the number of irreducible polynomials of degree d over GF(2) is $\Theta(2^d/d)$.)

   Conclude that for random f and s, the sequence $r_0, ..., r_{\ell(k)-1}$ has bias $O(\ell(k)/2^t)$.

Note that an implementation of the LFSR generator requires a mapping of random $k/2$-bit long string to *almost* random irreducible polynomials of degree $k/2$. Such a mapping can be constructed in $\exp(k)$ time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A more efficient mapping that uses a $O(k)$-bit long seek is described in [9, Sec. 8].

Exercise 8.37 (limitations on small-bias generators) Let G be an ε-bias generator with stretch ℓ, and view G as a mapping from $\text{GF}(2)^k$ to $\text{GF}(2)^{\ell(k)}$. As such, each bit in the output of G can be viewed as a polynomial[54] in the k input variables (each ranging in GF(2)). Prove that if $\varepsilon(k) < 1$ and each of these polynomials has *total degree* at most d, then $\ell(k) \leq \sum_{i=1}^{d} \binom{k}{i}$. Derive the following corollaries:

1. If $\varepsilon(k) < 1$ then $\ell(k) < 2^k$ (regardless of d).[55]
2. If $\varepsilon(k) < 1$ and $\ell(k) > k$ then G cannot be a linear transformation.[56]

[54]Recall that every Boolean function over GF(p) can be expressed as a polynomial of *individual degree* at most $p-1$.

[55]This upper-bound is optimal, because (efficient) ε-bias generators of stretch $\ell(k) = \text{poly}(\varepsilon(k)) \cdot 2^k$ do exists (see [169]).

[56]In contrast, bilinear ε-bias generators do exist; for example, $G(s) = (s, b(s))$, where $b(s_1, ..., s_k) = \sum_{i=1}^{k/2} s_i s_{(k/2)+i} \bmod 2$, is an ε-bias generator with $\varepsilon(k) = \exp(-\Omega(k))$. (Hint: Focusing on bias over sets that include the last output bit, prove that without loss of generality it suffices to analyze the bias of $b(U_k)$.)

Guideline (for the main claim): Note that, without loss of generality, all the aforementioned polynomials have a free term equal to zero (and have individual degree at most 1 in each variable). Next, consider the vector space spanned by all d-monomials over k variables (i.e., monomial having at most d variables). Since $\varepsilon(k) < 1$, the polynomials representing the output bits of G must correspond to a sequence of independent vectors in this space.

Exercise 8.38 (a sanity check for space-bounded pseudorandomness) The following fact is suggested as a sanity check for candidate pseudorandom generators with respect to space-bounded automata. The fact (to be proven as an exercise) is that, for every $\varepsilon(\cdot)$ and $s(\cdot)$ such that $s(k) \geq 1$ for every k, if G is (s, ε)-pseudorandom (as per Definition 8.20), then G is an ε-bias generator.

Exercise 8.39 In contrast to Exercise 8.38, prove that there exist $\exp(-\Omega(n))$-bias distributions over $\{0,1\}^n$ that are not $(2, 0.666)$-pseudorandom.

Guideline: Show that the uniform distribution over the set

$$\left\{\sigma_1 \cdots \sigma_n : \sum_{i=1}^{n} \sigma_i \equiv 0 \pmod 3\right\}$$

has bias $\exp(-\Omega(n))$.

Exercise 8.40 (approximate t-wise independence generators (following [169])) Combining a small-bias generator as in Theorem 8.26 with the t-wise independence generator of Eq. (8.16), and relying on the linearity of the latter, construct a generator producing ℓ-bit long sequences in which any t positions are at most ε-away from uniform (in variation distance), while using a seed of length $O(t + \log(1/\varepsilon) + \log\log \ell)$. (For max-norm a seed of length $O(\log(t/\varepsilon) + \log\log \ell)$ suffices.)

Guideline: First note that, for any t, ℓ' and $b \geq \log_2 \ell'$, the transformation of Eq. (8.16) can be implemented by a fixed linear (over GF(2)) transformation of a $t \cdot b$-bit seed into an ℓ-bit long sequence, where $\ell = \ell' \cdot b$. It follows that, for $b = \log_2 \ell'$, there exists a fixed GF(2)-linear transformation T of a random seed of length $t \cdot b$ into a t-wise independent bit sequence of the length ℓ (i.e., $T\,U_{t \cdot b}$ is t-wise independent over $\{0,1\}^\ell$). Thus, every t rows of T are linearly independent. The key observation is that when we replace the aforementioned random seed by an ε'-bias sequence, every $i \leq t$ positions in the output sequence have bias at most ε' (because they define a non-zero linear test on the bits of the ε'-bias sequence). Note that the length of the new seed (used to produce ε'-bias sequence of length $t \cdot b$) is $O(\log tb/\varepsilon')$. Applying Exercise 8.34, we conclude that any t positions are at most $2^{t/2} \cdot \varepsilon'$-away from uniform (in variation distance). Recall that this was obtained using a seed of length $O(\log(t/\varepsilon') + \log\log \ell)$, and the claim follows by using $\varepsilon' = 2^{-t/2} \cdot \varepsilon$.

Exercise 8.41 (small-bias generator and error-correcting codes) Show a correspondence between ε-bias generators of stretch ℓ and binary linear error-correcting codes (cf. Appendix E.1) mapping $\ell(k)$-bit long strings to 2^k-bit long strings such that every two codewords are at distance $(1 \pm \varepsilon(k)) \cdot 2^{k-1}$ apart.

Guideline: Associate $\{0,1\}^k$ with $[2^k]$. Then, a generator $G : [2^k] \to \{0,1\}^{\ell(k)}$ corresponds to the code $C : \{0,1\}^{\ell(k)} \to \{0,1\}^{2^k}$ such that, for every $i \in [\ell(k)]$ and $j \in [2^k]$, the i^{th} bit of $G(j)$ equals the j^{th} bit of $C(0^{i-1}10^{\ell(k)-i})$.

Exercise 8.42 (on the bias of sequences over a finite field) For a prime p, let ζ be a random variable assigned values in $\mathrm{GF}(p)$ and $\delta(v) \stackrel{\text{def}}{=} \Pr[\zeta = v] - (1/p)$. Prove that $\max_{v\in\mathrm{GF}(p)}\{|\delta(v)|\}$ is upper-bounded by $b \stackrel{\text{def}}{=} \max_{c\in\{1,\ldots,p-1\}}\{\|\mathsf{E}[\omega^{c\zeta}]\|\}$, where ω denotes the p^{th} (complex) root of unity, and that $\sum_{v\in\mathrm{GF}(p)}|\delta(v)|$ is upper-bounded by $\sqrt{p}\cdot b$.

Guideline: Analogously to Exercise 8.34, view probability distributions over $\mathrm{GF}(p)$ as p-dimensional vectors, and consider two bases for the set of complex functions over $\mathrm{GF}(p)$: the Kroniker basis (i.e., $k_i(x) = 1$ if $x = i$ and $k_i(x) = 0$) and the (normalize) Fourier basis (i.e., $f_i(x) = p^{-1/2}\cdot\omega^{ix}$). Note that the biases of ζ corresponds to the inner products of δ with the non-constant Fourier functions, whereas the distances of ζ from the uniform distribution correspond to the inner products of δ with the Kroniker functions.

Exercise 8.43 (a version of the Expander Random Walk Theorem) Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ' stays in W is at most $(\rho + (\lambda/d)^2)^{\ell'/2}$. In fact, prove a more general claim that refers to the probability that a random walk of length ℓ' intersects $W_0 \times W_1 \times \cdots \times W_{\ell'-1}$. The claimed upper-bound is

$$\sqrt{\rho_0}\cdot\prod_{i=1}^{\ell'-1}\sqrt{\rho_i + (\lambda/d)^2}, \tag{8.24}$$

where $\rho_i \stackrel{\text{def}}{=} |W_i|/|V|$.

Guideline: View the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in the graph and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first transformation shrinks the component that is orthogonal to the uniform distribution (which is the first eigenvalue of the adjacency matrix of the expander), whereas the second transformation shrinks the component that is in the direction of the uniform distribution. For further details, see §E.2.1.3.

Exercise 8.44 Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ' visits W more than $\alpha\ell'$ times is smaller than $\binom{\ell'}{\alpha\ell'}\cdot(\rho + (\lambda/d)^2)^{\alpha\ell'/2}$. For example, for $\alpha = 1/2$ and $\lambda/d < \sqrt{\rho}$, we get an upper-bound of $(32\rho)^{\ell'/4}$. We comment that much better bounds can be obtained (cf., e.g., [119]).

Guideline: Use a union bound on all possible sequences of $m = \alpha\ell'$ visits, and upper-bound the probability of visiting W in steps $j_1, ..., j_m$ by applying Eq. (8.24) with $W_i = W$ if $i \in \{j_1, ..., j_m\}$ and $W = V$ otherwise.