Computational Complexity:

A Conceptual Perspective

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.

February 5, 2007

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 $R \stackrel{\text{def}}{=} \{(f(x), x) : x \in \{0,1\}^*\}$. 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*,*

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

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*,*

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

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 $\Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] \;<\; 1/p(n)$, 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 $n^2 p(n) \leq |x|$ 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 $m = n^2 p(n)$, 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 $t'(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2 p(n))$).

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

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

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

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

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

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

Using $t(n) = n \cdot p(n)$, we get $\Pr[U_n \in S_n] > (1/2q(n^2p(n)))^{1/(n \cdot p(n))}$, which implies $\Pr[U_n \in S_n] > 1 - (1/2p(n))$ 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).

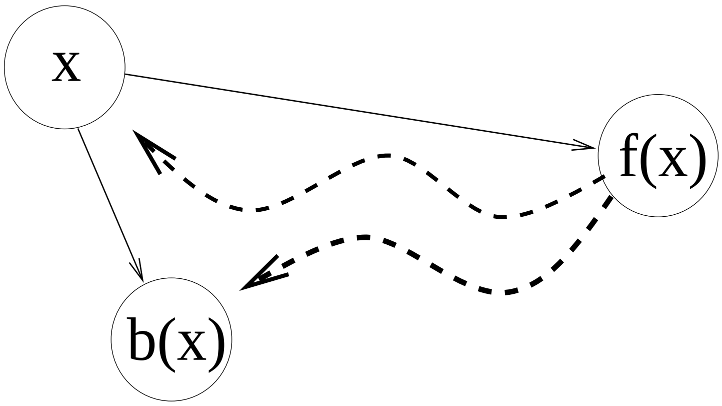
We use the term a *reducibility argument*, rather than just saying a reduction so as to emphasize that we do *not* refer here to standard (worst-case complexity) reductions. Let us clarify the distinction: In both cases we refer to *reducing* the task of solving one problem to the task of solving another 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 $f'(x,r) \stackrel{\text{def}}{=} (f(x),r)$, 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 $f(\sigma x') \stackrel{\rm def}{=} 0x'$. 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 $\varepsilon \stackrel{\rm def}{=} 1/2p(n)$

[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 $p>\frac{3}{4}+1/\mathrm{poly}(|x|)$ rather than $p>\frac{1}{2}+1/\mathrm{poly}(|x|)$. 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 $b(x,y)=\sum_{i=1}^n x_iy_i \bmod 2$ 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 $1-2\cdot(1-p)>\frac{1}{2}+\frac{1}{\mathrm{poly}(|x|)}$. 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 $p>\frac{3}{4}+\frac{1}{\mathrm{poly}(|x|)}$, whereas we only know that $p>\frac{1}{2}+\varepsilon$ for $\varepsilon=1/\mathrm{poly}(|x|)$.

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.

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

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 $B_x(r) \stackrel{\text{def}}{=} B(f(x), r)$. 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*

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

outputs x *with probability at least* $\mathrm{poly}(\varepsilon/n)$. Specifically, x is output with probability at least $p \stackrel{\text{def}}{=} \Omega(\varepsilon^2/n)$. 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*

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

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 $\mathsf{Pr}_{r \in \{0,1\}^n}[B(r) = b(x, r)] < \frac{1}{2} + \frac{\varepsilon}{2}$.

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 $\mathcal{P}$/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 $\mathcal{P}$/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 $\mathcal{E} \setminus \mathcal{P}$/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 $\mathcal{P}$/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 [125].

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).

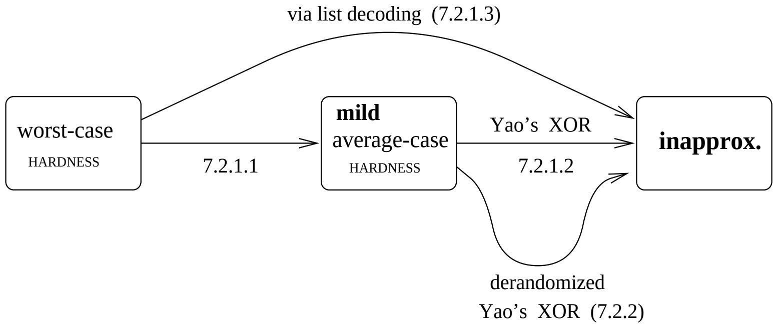


Figure 7.2: Proofs of hardness amplification: organization

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 $|F^m| = \text{poly}(2^n)$, which leads to setting $m = n/\log_2 n$ (yielding $|H| = n$ and $|F| = \text{poly}(n)$). 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* $S'(n') = S(n'/O(1))/\text{poly}(n')$ *it holds that* $\Pr[C'_{n'}(U_{n'}) \neq \hat{f}(U_{n'})] > (1/n')^2$*. 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 $(\text{poly}(n), n^{-3})$-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 $n^2 < |F| = \text{poly}(n)$. 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

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

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 $|C'_{n'}| > S(n'/O(1))/\mathrm{poly}(n')$. 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., [40].)

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 $\mathsf{Pr}[C_n(U_n) \neq f(U_n)] > 1/p_2(n)$. 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 $\mathsf{Pr}[C'_m(U_m) = P(U_m)] < \varepsilon'(m)$. 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 $\varepsilon'(m) \leq 1/p'(m)$.

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 $p'(t(n)\cdot n) = p_1(n)^{\Omega(1)}/\text{poly}(t(n))$. The less obvious part of the argument is reducing the approximation of P' to the approximation of F. The key observation is that

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

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 $\{C_m\}$*

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

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

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

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

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

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 $s(n) = p_1(n)/\mathrm{poly}(t(n)/\varepsilon(n))$ 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 $s(n) = p_1(n)/\mathrm{poly}(t(n)/\varepsilon(n))$).

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 $s_2^{(i)}(n) = \mathrm{poly}(n/\varepsilon(n))\cdot s(n) = p_1(n)$. 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 $\min(s_1^{(i)}(i\cdot n), s_2^{(i)}(n)/\mathrm{poly}(n/\varepsilon(n))) = s(n)$ 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.* □

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, $\mathsf{Pr}[C_n(X_n) = F(X_n)] > \rho(n)$. 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^{th} output of $C_n(x)$ by $C_n(x)_i$ (i.e., $C_n(x) = (C_n(x)_1, C_n(x)_2)$):

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

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 $\mathsf{Pr}[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ holds with noticeable probability. The last caveat motivates a separate treatment of z's having a noticeable value of $\mathsf{Pr}[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ 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 $\mathsf{Pr}[C(Y,z)_1 = F_1(Y)] \geq \varepsilon/2$ 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

$$\mathsf{Pr}[C(Y,z) = F(Y,z)] \leq \mathsf{Pr}[C(Y,z)_1 = F_1(Y)] < \varepsilon/2$$

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

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

We proceed according to the forgoing outline, first showing that if $\mathsf{Pr}[C(Y,Z)_1 = F_1(Y)] > \rho_1(\ell)$ 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 $\mathsf{Pr}[C(Y,z)_1 = F_1(Y)] \leq \rho_1(\ell)$.

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 $C'(y) \stackrel{\text{def}}{=} C(y,z)_1$ that contradicts the lemma's hypothesis concerning F_1. □

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 $\text{poly}(n/\varepsilon)$-large sample, denoted S, from the distribution $(Y, F_1(Y))$ and let $C''(z) \stackrel{\text{def}}{=} C(y,z)_2$, 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:

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

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 [99]. 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., [88, Sec. 2.3.2] and [99, 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 $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < \varepsilon'(n') \stackrel{\rm def}{=} \varepsilon(n'/3)$.*

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 [214]).

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 [214]).

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 $\Pr[C'_{n'}(U_{n'}) = g(U_{n'})] < \varepsilon'(n') \stackrel{\text{def}}{=} \varepsilon(m^{-1}(n'))$.*

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 $\Pr[C'_{n'}(U_{n'}) = g(U_{n'})] < \varepsilon'(n')$, 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*

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

- *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 $\Pr[X_n\in S_n] \geq \rho(n)$.*

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 $\mathsf{Pr}[C_n(X_n)=f(X_n)]\le 1-\rho(n)$. 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 $\mathsf{Pr}[X_n=x]\ge\rho(n)\cdot\mathsf{Pr}[Y_n=x]$. 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

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

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 [99, 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

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

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 $\mathsf{Pr}[C(Y_n^{(i)}) = f(Y_n^{(i)})]$. 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 [229]) 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

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

where the probability refers both to the choice of the circuit D_n and to the random variable Y_n. Let $B_n = \{x : \mathsf{Pr}[D_n(x) = f(x)] \leq 0.5 + \varepsilon'(n)\}$. Then, $\mathsf{Pr}[X_n \in B_n] < \rho(n)$, because otherwise we reach a contradiction to Eq. (7.14) by defining Y_n such that $\mathsf{Pr}[Y_n = x] = \mathsf{Pr}[X_n = x]/\mathsf{Pr}[X_n \in B_n]$ if $x \in B_n$ and $\mathsf{Pr}[Y_n = x] = 0$ 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 $\mathsf{Pr}[D'_n(x) = f(x)] > 1 - 2^{-n}$. 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 $\mathsf{Pr}[C_n(X_n) = f(X_n)] \geq \mathsf{Pr}[X_n \in \{0,1\}^n \setminus B_n] > 1 - \rho(n)$, 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 $\mathsf{Pr}[C_n(Y_n) =$

[26]Note that Y_n is ρ-dominated by X_n, whereas by the hypothesis $\mathsf{Pr}[D_n(Y_n) = f(Y_n)] \leq 0.5 + \varepsilon'(n)$. Using the fact that any ρ-dominated distribution is a convex combination of critically ρ-dominated distributions, it follows that $\mathsf{Pr}[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] \leq 0.5 + \varepsilon'(n)$ 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 [99, 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 $p'(t(n) \cdot n) \leq s'(n) - 1$).

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 $\rho(n) > 1/\text{poly}(n)$, and assume for simplicity that $\rho(n) \leq 1/n$. 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 $T'(n') = T(n'/3)/\text{poly}(n')$ 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 $(T, 1/\text{poly})$-inapproximable predicate into an $(T'', \Omega(1))$-inapproximable one, where $T''(n'') = T(n''/O(1))/\text{poly}(n'')$.

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 $t'(n) = T(n)/\text{poly}(n)$, 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 $\rho(n) \leq 1/n$ 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 $\delta(n') = \Omega(n' \cdot \alpha(n'/3))$). $\square$

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 [125, 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 [125]. □

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 [125].

[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 [63]. 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 [233]. A proof of Theorem 7.5 has first appeared in [84].

The concept of hard-core predicates was suggested by Blum and Micali [37]. 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 [96]. The proof presented here was suggested by Rackoff. We comment that the original proof has its own merits (cf., e.g., [102]).

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 [20], whereas Theorem 7.19 is due to [125]. 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 [233], Yao's XOR Lemma (i.e., Theorem 7.13) is not even stated in [233] 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 [99, Sec. 3]). The proof presented in §7.2.1.2 is due to Goldreich, Nisan and Wigderson [99, 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 [219].

The notion of a hard region and its applications to proving the original version of Yao's XOR Lemma are due to Impagliazzo [123] (see also [99, Sec. 4]). The first derandomization of Yao's XOR Lemma (i.e., Lemma 7.22) also originates in [123], while the second derandomization (i.e., Lemma 7.23) as well as Theorem 7.19 are due to Impagliazzo and Wigderson [125].

The worst-case to average-case reduction (i.e., §7.2.1.1, yielding Theorem 7.12) is due to [20]. This reduction follows the self-correction paradigm of Blum, Luby and Rubinfeld [38], which was first employed in the context of a (strict)[31] worst-case to average-case reduction by Lipton [153].

[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., [105]); 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).

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 [214].

Hardness amplification for $\mathcal{NP}$ has been the subject of recent attention: An amplification of mild inapproximability to strong inapproximability is provided in [118], and an indication to the impossibility of a worst-case to average-case reductions (at least non-adaptive ones) is provided in [40].

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 $r^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$.

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 $\mathrm{poly}(n/\varepsilon)$ 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 $\mathrm{poly}(n/\varepsilon)$-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 $\mathcal{P}/\mathrm{poly}$ 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 $t \stackrel{\mathrm{def}}{=} dm + 1$. 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 $\Pr[C_n(X_n) = h(X_n)] \le \varepsilon(n)$. Suppose that $s' : \mathbb{N} \to \mathbb{N}$ and $\varepsilon' : \mathbb{N} \to [0,1]$ satisfy $s'(n + \ell(n)) \le s(n)/\text{poly}(n/\varepsilon'(n + \ell(n)))$ and $\varepsilon'(n + \ell(n)) \ge 2\varepsilon(n)$. 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

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

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 $C_z(x) \stackrel{\rm def}{=} C(x,z)$ 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 $F'(x_1, ..., x_{t(n)}, r) \stackrel{\rm def}{=} b(f(x_1) \cdots f(x_{t(n)}), r)$, 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 $F(x) \stackrel{\rm def}{=} F'(x, 1^{t'(|x|)})$, 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).[32]

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 the guessing of the value of the direct product function P. Present a reduction in

[32] 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 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 $\Pr[C_n(U_n) \notin f(U_n)] > 1/p_2(n)$ then for every family of p'-size circuits, $\{C'_m\}_{m\in\mathbb{N}}$, it holds that $\Pr[C'_m(U_m) \in P(U_m)] < \varepsilon'(m)$, 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).[33]

   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 $q(N) \le \text{poly}(N)$.

[33] 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.)[34] 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.

[34] 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').