Texts in Computational Complexity: Amplification of Hardness

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

February 1, 2006

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.

One key issue that arises whenever one tries to utilize hard problem is bridging the gap between "occasional" hardness (e.g., worst-case hardness or mild average-case hardness) and "typical" hardness (i.e., inapproximability). Much of the current chapter is devoted to this issue, which is known by the term *hardness amplification*.

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 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 of the type that can be used towards derandomized $\mathcal{BPP}$ in a non-trivial way (see [9, Text 17]).

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 hard to solve for other people. 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 almost everywhere, and that the latter yield predicates that are very hard to approximate (called *hard-core predicates*). The latter are useful for the construction of general-purpose pseudorandom generators (see [9, Text 17]) as well as for a host of cryptographic applications (see [7, 8]).

The order of presentation of the two aforementioned conjectures and their consequences is actually reversed: We start (in Section 1) with the study of one-way function, and only later (in Section 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 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 in comparison to the amplification of hardness in the context of $\mathcal{E}$. (In fact, Section 2 seems the most technical text in this course.) 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 chosen 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 and randomized algorithms. In particular, standard conventions regarding random variables (presented in [9, Text 20]) will be extensively used.

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 exists is a complexity theoretic analogue of 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 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 [7, 8]). Similarly, one-way functions are closely related to (general-purpose) pseudorandom generators, but this connection will be explored in [9, Text 17]. 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.

1.1 The concept of one-way functions

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, because the assumption refers to the worst-case complexity of problems, and it may be that hard instances are hard to find. But then, it seems that if we cannot generate hard instances then we cannot benefit from their existence.

In Section 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 wish the hard on the average problem to be in $\mathcal{NP}$. In this case, we need to assume that, for some problem in $\mathcal{NP}$, hard instances not only exist but are easy to generate. That is, $\mathcal{NP}$ is "hard on the average" with respect to a distribution that is efficiently sampleable. This assumption will be further discussed in [9, Text 18].

However, for the aforementioned applications (e.g., in cryptography) this assumption does not seem to suffice either and 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 (of course, for a person that does not get the solution (resp., witness)).

Let us formulate the latter notion. 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, $\Pr[G(1^n) \in R \cap (\{0,1\}^n \times \{0,1\}^*)] = 1$.

2. It is 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 $\Pr[(G_1(1^n), S(G_1(1^n)) \in R] = \mu(n)$, 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 exists, 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 1 (one-way functions): *A function $f:\{0,1\}^* \to \{0,1\}^*$ is called* one-way *if the following two conditions hold:*

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

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

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

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

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'.[2]

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 24), 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 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 s.

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

Comments. Several candidates one-way functions and variation on the basic definition are presented in [7, Chap. 2]. 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. Here we use the form discussed in Footnote 2.

Definition 3 (one-way functions, non-uniformly hard): *A one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be* non-uniformly hard to invert *if for every family of polynomial-size circuits $\{C_n\}$, every polynomial p, and all sufficiently large n,*

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

[2]An alternative formulation of Eq. (1) relies on the conventions in [9, Text 2]. Specifically, letting U_n denote a random variable uniformly distributed in $\{0,1\}^n$, we may write Eq. (1) as $\mathsf{Pr}[A'(f(U_n),1^n)\in f^{-1}(f(U_n))] < 1/p(n)$, recalling that both occurrences of U_n refer to the same sample.

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 25.

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

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

$$\mathsf{Pr}_{x\in\{0,1\}^n}[A'(f(x),1^n) \not\in f^{-1}(f(x))] \;>\; \frac{1}{p(n)} \tag{2}$$

   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.

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 1). For clarity, we will occasionally refer to one-way functions as in Definition 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 26). 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 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

$$\begin{aligned} F(x_1,...,x_t) &= (f(x_1),...,f(x_t)) \\ &\quad \text{where } t \stackrel{\text{def}}{=} n\cdot p(n) \text{ and } x_1,...,x_t \in \{0,1\}^n. \end{aligned} \tag{3}$$

(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 the relative volume of S^t in $(\{0,1\}^n)^t$, for $S \subset \{0,1\}^n$, where S represents the set of "easy to invert" f-images). Specifically, one may *not* assume that the potential inverting algorithm works independently on each block. Indeed this assumption seems reasonable, but we should not make assumptions regarding arbitrary algorithms (as appearing in 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 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 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

$$\mathsf{Pr}[B'(F(U_m)) \in F^{-1}(F(U_m))] > \frac{1}{q(m)} \tag{4}$$

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. (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,

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

[3] One simple extension is to define $F(x)$ 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., $x = x_1 \cdot x_{n \cdot p(n)} x'$, where $x_1, ..., x_{n \cdot p(n)} \in \{0,1\}^n$).

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 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 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 5.1 and 5.2, the theorem follows.

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

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

where $U_n^{(1)}, ..., U_n^{(n\cdot p(n))}$ denote the n-bit long blocks in the random variable $U_{n^2p(n)}$. On the other hand, by Eq. (4) we have

$$\begin{aligned}&\sum_{i=1}^{t(n)} \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)}))\ \wedge\ U_n^{(i)} \in \{0,1\}^n \setminus S_n\right]\\ &\quad> \frac{1}{q(n^2p(n))} - \mathsf{Pr}\left[(\forall i)\, U_n^{(i)} \in S_n\right]\end{aligned}$$

Thus, we have $\mathsf{Pr}[U_n \in S_n]^{t(n)} > \frac{1}{q(n^2p(n))} - t(n)\cdot\frac{n}{t'(n)}$. Using $t'(n) = 2n^2\cdot p(n)\cdot q(n^2p(n))$ and $t(n) = n\cdot p(n)$, we get $\mathsf{Pr}[U_n \in S_n] > (1/2q(n^2p(n)))^{1/(n\cdot p(n))}$, which implies $\mathsf{Pr}[U_n \in S_n] > 1-(1/2p(n))$ for sufficiently large n. Claim 5.2 follows, and so does the theorem. $\square$

Digest. Let us recall the structure of the proof of Theorem 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, as 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).

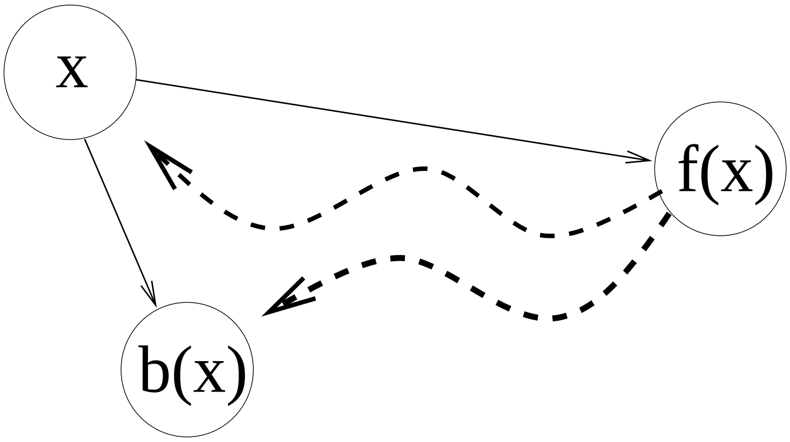
We use the term a *reducibility argument*, rather than just saying a reduction so as to emphasize that we do *not* refer here to standard (worst-case complexity) reductions. Let us clarify the distinction: In both cases we refer to *reducing* the task of solving one problem to the task of solving another 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 other cases may be handled too (e.g., these distributions may be "sufficiently close" for the specific purpose). In any case, a careful analysis 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 [9, Text 18].)

An information theoretic analogue. Theorem 5 has a natural information theoretic (or "probabilistic") analogue that asserts that repeating an experiment that has a noticeable failure probability, sufficiently many times yields some failure with very high probability. The reader is probably convinced at this stage that the proof of Theorem 5 is much more complex than the proof of the information theoretic analogue. In the information theoretic context the repeated events are independent by definition, whereas in the computational context no such independence (which corresponds to the naive argument discussed at the beginning of the proof of Theorem 5) can be guaranteed. Another indication to the difference between the two settings follows. In the information theoretic setting the probability that none of the failure events occurs decreases exponentially in 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, it may be the case that F constructed in the proof of Theorem 5 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}).

1.3 Hard-Core Predicates

One-way functions *per se* suffice for one central application: the construction of secure signature schemes (see [8, Chap. 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 out 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 1: The hard-core of a one-way function – an illustration.

Definition 6 (hard-core predicates): *A polynomial-time computable predicate $b : \{0,1\}^* \rightarrow \{0,1\}$ 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*

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

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 $b : \{0,1\}^* \rightarrow \{0,1\}$ and $f : \{0,1\}^* \rightarrow \{0,1\}^*$, 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). Finally, if b is a hard-core of a 1-1 function f that is polynomial-time computable then f must be a one-way function. In general, the interesting case is when being a hard-core is a computational phenumenon rather an information theoretic one (which is due to "information loss" of f).

Theorem 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, 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 probbaility $1/2$, 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 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 the inner-product mod 2 of X and r, conditioned on a uniformly selected $r \in \{0,1\}^n$, where X is a random variable with super-logarithmic min-entropy, which represents the "effective" knowledge of x, when given $f(x)$).[4]

Our starting point is a probabilistic polynomial-time algorithm B that satisfies, for some polynomial p and infinitely many n's, $\mathsf{Pr}[B(f(X_n),U_n) = b(X_n,U_n)] > (1/2) + (1/p(n))$, where X_n and U_n are uniformly and independently distributed over $\{0,1\}^n$. Using a simple averaging argument, we focus on a $\varepsilon \stackrel{\text{def}}{=} 1/2p(n)$ fraction of the x's for which $\mathsf{Pr}[B(f(x),U_n) = b(x,U_n)] > (1/2) + \varepsilon$ holds. We will show how to use B in order to invert f, on input $f(x)$, provided that x is in the 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 $p > \frac{3}{4} + 1/\text{poly}(|x|)$ rather than at least $\frac{1}{2} + 1/\text{poly}(|x|)$. In this case, retrieving x from $f(x)$ is quite easy: To retrieve the i^{th} bit of x, denoted x_i, we randomly select $r \in \{0,1\}^{|x|}$, and obtain $B(f(x),r)$ and $B(f(x),r \oplus e^i)$, where $e^i = 0^{i-1}10^{|x|-i}$ and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. A key observation underlying the foregoing scheme as well as the rest of the proof is that $b(x,r \oplus s) = b(x,r) \oplus b(x,s)$, which can be readily verified by writing $b(x,y) = \sum_{i=1}^{n} x_i y_i \bmod 2$ and noting that addition modulo 2 of bits corresponds to their XOR. Indeed, note that if both $B(f(x),r) = b(x,r)$ and $B(f(x),r \oplus e^i) = b(x,r \oplus e^i)$ hold, then $B(f(x),r) \oplus B(f(x),r \oplus e^i)$ equals $b(x,r) \oplus b(x,r \oplus e^i) = b(x,e^i) = x_i$. The probability that both $B(f(x),r) = b(x,r)$ and $B(f(x),r \oplus e^i) = b(x,r \oplus e^i)$ hold, for a random r, is at least $1 - 2 \cdot (1-p) > \frac{1}{2} + \frac{1}{\text{poly}(|x|)}$. Hence, repeating the above procedure sufficiently many times (using independent random choices of such r's) and ruling by majority, we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $p > \frac{3}{4} + \frac{1}{\text{poly}(|x|)}$, whereas we only know that $p > \frac{1}{2} + \varepsilon$ for $\varepsilon = 1/\text{poly}(|x|)$.

The problem with the foregoing procedure is that it doubles the original error probability of algorithm B on inputs of the form $(f(x),\cdot)$. Under the unrealistic assumption (made above), 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 above 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 to apply algorithm B only once per each r (and i), instead of twice. Specifically, we will use algorithm B to obtain a "guess" for $b(x,r \oplus e^i)$, and obtain $b(x,r)$ in a different way (which does not require 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

[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 [9, Text 20])) 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 $\{0,1\}^n$, and $b(u,v)$ denotes the inner-product mod 2 of u and v.

(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 (seemingly contradictory) 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 28. 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.

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 any fixed 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

$$\begin{aligned} &\Pr_{s^1,...,s^\ell}[\rho^J \oplus B(f(x), r^J \oplus e^i) = b(x,e^i)] \\ &\quad = \Pr_{s^1,...,s^\ell}[B(f(x), r^J \oplus e^i) = b(x, r^J \oplus e^i)] \;>\; \frac{1}{2} + \varepsilon \end{aligned} \tag{5}$$

[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.

[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 to set $t = \text{poly}(n)$ and work with error $1/2n$, the difference in the error probability between the two approximation schemes is not important here. For a wider perspective see [9, Text 20].

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. (5) refers to the correctness of a single vote for $b(x,e^i)$. Using $m = O(n/\varepsilon^2)$ 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 retreived 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 retreived correctly with probability $1/\mathrm{poly}(n)$, and the theorem follows. $\square$

Digest. Looking at the proof of Theorem 7, we note that it actually refers to a black-box $B_x(\cdot)$ that approximates $b(x,\cdot)$; specifically, in the case of Theorem 7 we used $B_x(r) \stackrel{\rm def}{=} B(f(x),r)$. In particular, the proof does not use the fact that we can verify the correctness of the preimage recovered by the described process. Indeed, using the alternative procedure outlined in Footnote 5, the proof extends to establish *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*

$$\mathsf{Pr}_{r\in\{0,1\}^n}[B_x(r) = b(x,r)] \geq \frac{1}{2}+\varepsilon \tag{6}$$

outputs, with probability at least $1/2$*, a list of* n*-bit strings that includes* x. Noting that x is merely a string for which Eq. (6) holds, and that the procedure may get n and ε as inputs, we derive

Theorem 8 (Theorem 7, revisited): *There exists a probabilistic oracle machine that, given parameters* n,ε *and oracle access to any function* $B:\{0,1\}^n\to\{0,1\}$*, for every* $x\in\{0,1\}^n$*, given oracle access to any* B_x *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 8 can be viewed as a list decoding[7] 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 $r\in\{0,1\}^n$.

Applications. Hard-core predicates play a central role in the construction of general-purpose pseudorandom generators (see [9, Text 17]). commitment schemes and zero-knowledge proofs (see [7, Chap. 4]), and encryption schemes (see [8, Chap. 5]).

[7]In contrast to standard decoding in which one recovers the unique information that is encoded in the codeword that is closest to the given string, in list decoding one recovers all strings having encoding that is at a specified distance from the given string. We mention that list decoding is applicable and valuable in the case that the specified distance does not allow for unique decoding and/or that the specified distance is greater than half the distance of the code. See further discussion in [9, Text 12].

2 Hard Predicates in E

We start again with the assumption $\mathcal{P} \neq \mathcal{NP}$. In fact, we consider the seemingly stronger assumption by which $\mathcal{NP}$ 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 weaker assumption (see Exercise 31). 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.*

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

Definition 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*

$$\Pr[C(U_n) \neq f(U_n)] \geq \frac{\rho(n)}{2} \tag{7}$$

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

We chose the specific form of Eq. (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 $\Pr[C(U_n) \neq f(U_n)] \geq 2^{-n}$ for every circuit C_n of size $S(n)$), whereas no predicate can be (S,ρ)-inapproximability for $\rho(n) = 1 - O(2^{-n})$ even with $S(n) = O(n)$ (i.e., in this case $\Pr[C(U_n) = f(U_n)] \geq 0.5 + O(2^{-n})$ for some linear-size circuit; see Exercise 32). Indeed, Eq. (7) can be interpreted as an upper-bound on the *correlation* of each adequate circuit with f (i.e., $\mathsf{E}[\chi(C(U_n), f(U_n))] \leq 1 - \rho(n)$, where $\chi(\sigma,\tau) = 1$ if $\sigma = \tau$ and $\chi(\sigma,\tau) = -1$ otherwise). Thus, T-inapproximability means that no family of size T circuits can correlate f better than $1/T$.

Comments. Recall that $\mathcal{E}$ denote the class of exponential-time solvable decision problems (equivalently, exponential-time computable Boolean predicates); that is, $\mathcal{E} = \cup_\varepsilon \mathrm{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{\varepsilon n}$. We highlight the aforementioned term *almost everywhere*: 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 computes 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.

2.1 Amplification wrt polynomial-size circuits

As hinted above, our goal is to prove the following result.

Theorem 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 $\mathcal{E}$.*

Theorem 10 is used towards deriving a meaningful derandomization of $\mathcal{BPP}$ under the aforementioned assumption (see [9, Text 17]). We present two proofs of Theorem 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 $\varepsilon(n) = 1/n^2$.

2. For any polynomial p, we prove that *if for every polynomial q the function f is $(q, 1/p)$-inapproximable, then the function $F(x_1, ..., x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$, is T-inapproximable for any polynomial T.* This claim is known as Yao's XOR Lemma, and its proof is far more complex than the proof of its information theoretic analogue.

The second proof consists of showing that the construction employed in the first step, when composed with Theorem 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 2).

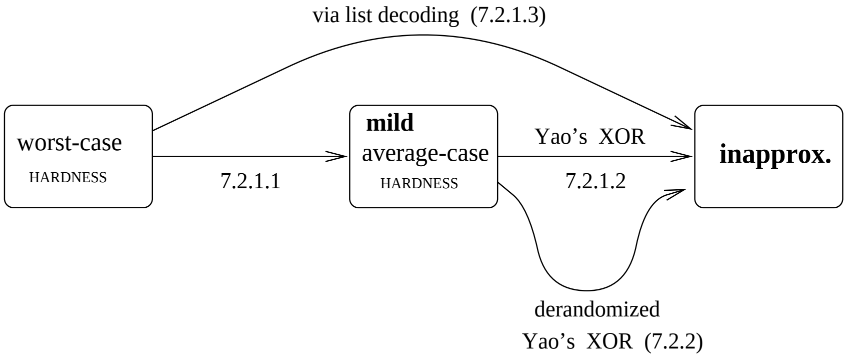


Figure 2: Proofs of hardness amplification: organization

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

The construction 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 at least 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.

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, the extended function is at least as hard on the worst-case as f, and by self-correction the extended function must be mildly hard in the average-case. Details follow.

Construction 11 (multi-variate extension)[8]: *For any function $f_n : \{0,1\}^n \to \{0,1\}$, finite field F, $H \subset F$ and integer m such that $|H|^m = 2^n$ and $|F| \geq m|H|$, 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 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. 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 $m = O(n/\log n)$ (yielding $|F| = \text{poly}(n)$). 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 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 $|\hat{f}(x)| \leq |x|$ 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 $\Pr[C'_{n'}(U_{n'}) \neq \hat{f}(U_{n'})] > (1/n')^2$.*

Theorem 12 completes the first step of the proof of Theorem 10, except that we desire a Boolean function rather than one that does not stretch its input. The extra step (of obtaining a Boolean function that is $(\text{poly}(n), n^{-3})$-inapproximable) may be taken by considering the bits in the output of the function (see Exercise 33).[9] That is, if $\hat{f}$ is hard to compute on an $(1/n')^2$ fraction of the n'-bit long inputs then the Boolean predicate that returns an indicated bit of $\hat{f}(x)$ must be mildly inapproximable.

Proof: 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 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 note that by making $m|H| = o(n^2)$ oracle calls to any circuit $C'_{n'}$ that satisfies $\Pr[C'_{n'}(U_{n'}) = \hat{f}_n(U_{n'})] > 1 - (1/n')^2 > 1 - (1/3m|H|)$, we can probabilistically recover the value of ($\hat{f}_n$ and thus) f_n on each input, with probability at least $2/3$. Using amplification and derandomization, 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 the theorem follows. ■

[8]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 $\text{poly}(|H|^m \log |F|)$-time.

[9]A quantitatively stronger bound can be obtained by noting that the proof of Theorem 12 actually establishes an error lower-bound of $\Omega((\log n')/(n')^2)$ and that $|\hat{f}(x)| = O(\log |x|)$.

Digest. The proof of Theorem 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 point, using oracle calls to circuits that, for every n', compute $\hat{f}$ correctly on a $1-(1/n')^2$ fraction of the n'-bit long inputs. We note 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., [3].)

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 (as in Section 1.2) the proof of the computational phenomenon is considerably more complex than the analysis of the information theoretic analogue.

Theorem 13 (Yao's XOR Lemma): *Let p be a polynomial and suppose that the Boolean function f is $(T, 1/p)$-inapproximable, for every polynomial T. Then the function $F(x_1, ..., x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$, is T'-inapproximable for every polynomial T'.*

Combining Theorems 12 and 13 (and Exercise 33), we complete the (first) proof of Theorem 10. Several different proofs of Theorem 13 are known. We choose using a proof that benefits most from the material already presented (in Section 1). This proof proceeds in two steps: 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, and next we establish the desired result by an application of Theorem 8. In fact, the first step is the main one, and we believe that it is of independent interest (and thus generalize it from Boolean functions to arbitrary ones).

Theorem 14 (The Direct Product Lemma): *Let p be a polynomial and $f : \{0,1\}^* \to \{0,1\}^*$. Suppose that for every family of polynomial-size circuits, $\{C_n\}_{n\in\mathbb{N}}$, and all sufficiently large $n \in \mathbb{N}$, it holds that $\mathsf{Pr}[C_n(U_n) \neq f(U_n)] > 1/p(n)$. 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 $t(n) = n \cdot p(n)$. Then, for every family of polynomial-size circuits, $\{C'_m\}_{m\in\mathbb{N}}$, it holds that $\mathsf{Pr}[C'_m(U_m) = P(U_m)] < \mu(m)$, where μ is a negligible function.*

Theorem 13 follows from Theorem 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. Applying Theorem 8, we infer that P' is T'-inapproximable for every polynomial T'. Lastly, we reduce the approximation of P' to the approximation of F (see Exercise 34), and Theorem 13 follows.

Proof of Theorem 14. As in the proof of Theorem 5, we show how to converts 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 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 will take a weighted majority vote among various answers, where the weights reflect our confidence in the correctness of the various answers.

We derive Theorem 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 statement of the lemma is not symmetric with respect to the two functions, guaranteeing a stronger (and in fact lossless) preservation of circuit sizes for one of the functions (which is arbitrarily chosen to be F_1).

Lemma 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 $\{C_m\}$*

$$\mathsf{Pr}[C_m(Y_m)=F_1(Y_m)] \leq \rho_1(m)$$

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

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

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

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

Theorem 14 is derived from Lemma 15 by using careful induction, which capitalizes on the asymmetry of Lemma 15. Specifically, 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 every polynomial s and a noticeable function ε (i.e., $\varepsilon(n) > 1/p(n)$ for some positive polynomial p), we prove by induction on i that circuits of size $s(n)$ cannot compute $P^{(i)}(U_{i\cdot n})$ with success probability greater than $(1-(1/p(n))^i + i \cdot \varepsilon(n)$. (The induction basis is guaranteed by the theorem's hypothesis.) The induction step is proved using Lemma 15 with $F_1 = P^{(i-1)}$ and $F_2 = f$ (along with $\rho_1((i-1)n) = (1-(1/p(n))^{i-1} + (i-1) \cdot \varepsilon(n)$, $s_1((i-1)n) = s(n)$, $\rho_2(n) = 1-(1/p(n))$ and $s_2(n) = \text{poly}(n/\varepsilon(n)) \cdot s(n)$). In particular, we use again the theorem's hypothesis regarding f, and note that $((1-(1/p(n))^{i-1} + (i-1) \cdot \varepsilon(n)) \cdot (1-(1/p(n)) + \varepsilon(n)$ is upper-bounded by $(1-(1/p(n))^i + i \cdot \varepsilon(n)$. Thus, no $s(n)$-size circuit can compute $P^{(t(n))}(U_{t(n)\cdot n})$ with success probability greater than $(1-(1/p(n))^{t(n)} + t(n) \cdot \varepsilon(n) = \exp(-n) + t(n) \cdot \varepsilon(n)$.

Proof of Lemma 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., $C_n(x) = (C_n(x)_1, C_n(x)_2)$):

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

The basic idea is that if the first factor is greater than $\rho_1(\ell(n))$ then we derive a circuit contradicting the lemma's hypothesis regarding F_1, whereas if the second factor is significantly greater than $\rho_2(n - \ell(n))$ then we derive a circuit contradicting the lemma's hypothesis regarding F_2. The basic idea for the latter case 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. This may work provided the condition holds with noticeable probability. The last caveat motivates a separate treatment of z's with noticeable $\mathsf{Pr}[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ and of the rest.

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. Then, we upper-bound the success probability of C by $\mathsf{Pr}[C(Y, Z) = F(Y, Z) \wedge Z \in G] + \varepsilon/2$, where the bound follows by observing that for any $z \notin G$:

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

Thus, using $\mathsf{Pr}[C(Y, z) = F(Y, z)] > \rho(n) = \rho_1(\ell) \cdot \rho_2(n - \ell) + \varepsilon$, we have

$$\mathsf{Pr}[C(Y, Z) = F(Y, Z) \wedge Z \in G] > \rho_1(\ell) \cdot \rho_2(n - \ell) + \frac{\varepsilon}{2}. \tag{8}$$

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 derive circuits violating the hypothesis concerning F_2. Actually, we prove something stronger (which we will actually need for the other case).

Claim 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 $\mathsf{Pr}[C(Y, z)_1 = F_1(Y)] > \rho_1(\ell)$, we obtain a circuit $C'(y) \stackrel{\text{def}}{=} C(y, z)_1$ that contradicts the lemma's hypothesis concerning F_1. $\square$

Using Claim 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 15.2: There exists a circuit C'' of size $s_2(n - \ell)$ such that

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

Proof: The second inequality is due to Eq. (8), and thus we focus on establishing the first one. We construct the circuit C'' as suggested in the foregoing outline. Specifically, we take a poly(n/ε)-large sample, denoted S, from the distribution $(Y, F_1(Y))$ and let $C''(z) \stackrel{\text{def}}{=} C(y, z)_2$, where (y, v) is a uniformly selected among the elements of S for which $C(y, z)_1 = v$ holds. Details follow.

Let S be a sequence of $m \stackrel{\text{def}}{=}$ poly(n/ε) 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 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 for 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 $m = \Omega(n^2/\varepsilon^3)$. 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. In particular, with probability greater than $1 - 2^{-n}$, it holds that

$$\frac{|\{(y,v) \in S_z : C(y,z)_2 = F_2(z)\}|}{|S_z|} \geq \Pr[C(Y,z)_2 = F_2(z) \mid C(Y,z)_1 = F_1(Y)] - \frac{\varepsilon}{2} \tag{9}$$

Thus, with positive probability, Eq. (9) holds for all $z \in G \subseteq \{0,1\}^{n-\ell}$. The circuit C'' computing F_2 is now defined as follows. A set $S = \{(y_i, v_i) : i = 1, ..., m\}$ satisfying Eq. (9) for all good z's is "hard-wired" into the circuit C''. (In particular, S_z is not empty for any good z.) 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 a standard averaging argument.) Using the definition of C'', Eq. (9), and Claim 15.1, we have:

$$\begin{aligned}
\Pr[C''(Z) = F_2(Z)] &\geq \sum_{z \in G} \Pr[Z = z] \cdot \Pr[C''(z) = F_2(z)] \\
&= \sum_{z \in G} \Pr[Z = z] \cdot \frac{|\{(y,v) \in S_z : C(y,z)_2 = F_2(z)\}|}{|S_z|} \\
&\geq \sum_{z \in G} \Pr[Z = z] \cdot \left(\Pr[C(Y,z)_2 = F_2(z) \mid C(Y,z)_1 = F_1(Y)] - \frac{\varepsilon}{2} \right) \\
&= \sum_{z \in G} \Pr[Z = z] \cdot \left(\frac{\Pr[C(Y,z)_2 = F_2(z) \wedge C(Y,z)_1 = F_1(Y)]}{\Pr[C(Y,z)_1 = F_1(Y)]} - \frac{\varepsilon}{2} \right) \\
&\geq \left(\sum_{z \in G} \Pr[Z = z] \cdot \frac{\Pr[C(Y,z) = F(Y,z)]}{\rho_1(\ell)} \right) - \frac{\varepsilon}{2}
\end{aligned}$$

where the last inequality is due to Claim 15.1. 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 14 involves invoking a quantitative lemma 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 15 (as well as Theorem 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 [11]. Indeed, a good lesson from the proof of Lemma 15 is that non-uniform circuits can "effectively sample" any distribution. Lastly, we mention that Theorem 5 (the amplification of one-way functions) and Theorem 13 (Yao's XOR Lemma) also have (tight) quantitative versions (see, e.g., [7, Sec. 2.3.2] and [11, Sec. 3], respectively).

2.1.3 List decoding and hardness amplification

In this subsection we provide an alternative proof of Theorem 10, showing that (for a suitable choice of parameters) combining Construction 11 with the inner-product construction (of Theorem 8) yields the desired result. Specifically, we claim the following.

Proposition 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 $\varepsilon(n) > n/2^n$. 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 11 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 $\mathsf{Pr}[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < \varepsilon'(n') \stackrel{\text{def}}{=} \varepsilon(n'/3)$.*

In particular, for some $\gamma > 0$, Proposition 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 $\mathsf{Pr}[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < 1/S'(n')$. Combining this with Theorem 8, 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 10 follows.

Proposition 16 is proven by observing that the transformation of f_n to $\hat{f}_n$ constitutes a "good" code (see [9, Text 12]) 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 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* efficient and supports implicit decoding with parameters q, ℓ, α *if it satisfies the following two conditions:*

1. Encoding: *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 $\Gamma(x) \in [q(|x|)]^{\ell(|x|)}$ as a mapping from $[\ell(|x|)]$ to $[q(|x|)]$.

2. Decoding: *There exists a polynomial p such that the following holds. For every $w : [\ell(N)] \to [q(N)]$ and $x \in \{0,1\}^N$ such that $\Gamma(x)$ is $(1-\alpha(N))$-close to w, there exists an oracle-aided[10] 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 trhe 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 only $p((\log N)/\alpha(N))$ queries to w, which may be a highly corrupted version of $\Gamma(x)$. On the other hand, the decoding condition does not refer to the complexity of obtaining the aforementioned oracle-aided circuits.

We mention that the transformation of f_n to $\hat{f}_n$ underlying Proposition 16 is efficient and supports implicit decoding with parameters q, ℓ, α such that $\ell(2^n) = \ell(|\langle f_n\rangle|) = |\langle f_n\rangle|^3 = 2^{3n}$,

[10] Oracle-aided 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 $p((\log N)/\alpha(N))$. We comment that we potentially consider also non-binary oracles, which return elements in $[q(N)]$.

$\alpha(2^n) = \varepsilon(n)$, and $q(2^n) = (n/\alpha(2^n))^3$. Furthermore, 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 $w : [\ell(2^n)] \rightarrow [q(2^n)]$, and the corresponding oracle-aided circuits can be constructed in probabilistic $p(n/\alpha(2^n))$-time.[11] These results are termed "list decoding" (with implicit representations). We stress that all these facts are highly non-trivial, but beyond the scope of the current text (and the interested reader is referred to [17]). Our focus is on showing that such efficient codes that supports implicit decoding suffice for worst-case to (strongly) average-case reductions.

Theorem 18 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S, and let $\varepsilon : \mathbb{N} \rightarrow [0,1]$. Consider $\ell : \mathbb{N} \rightarrow \mathbb{N}$ such that $n \mapsto \log_2 \ell(2^n)$ is a 1-1 map of the integers, and let $m(n) = \log_2 \ell(2^n)$. Suppose that the mapping $\Gamma : \{0,1\}^* \rightarrow \{0,1\}^*$ is efficient and supports implicit decoding with parameters q, ℓ, α such that $\alpha(N) = \varepsilon(\lfloor \log_2 N \rfloor)$. Define $g_n : [\ell(2^n)] \rightarrow [q(2^n)]$ such that $g_n(i) = \Gamma(\langle f_n \rangle)(i)$, where $\langle f_n \rangle$ denotes the 2^n-bit long description of the truth-table of f_n. Then, the function $g : \{0,1\}^* \rightarrow \{0,1\}^*$, 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 $S'(n') = \mathrm{poly}(\varepsilon(m^{-1}(n'))/n') \cdot S(m^{-1}(n'))$ 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. Regarding g's average-case hardness, consider a circuit $C' = C'_{n'}$ violating the conclusion of the theorem, let $n = m^{-1}(n')$, and recall that $\varepsilon'(n') = \varepsilon(n) = \alpha(2^n)$. Then, C' is $(1-\alpha(2^n))$-close to $g_n = \Gamma(\langle f_n \rangle)$, and the decoding condition of Γ asserts that we can recover each bit of $\langle f_n \rangle$ (i.e., evaluate f_n) by a circuit of size $p(n/\alpha(2^n)) \cdot S'(n') < S(n)$, in contradiction to the hypothesis. □

Comment. For simplicity, we formulated Definition 17 in a crude manner that suffices for the foregoing application. A more careful formulation of the decoding condition refers to codewords that are $(1-\alpha(N))$-close to the oracle $w : [\ell(N)] \rightarrow [q(N)]$ rather than $(1 - ((1/q(N)) + \alpha(N)))$-close to it.[12] Needless to say, the difference is insignificant in the case that $\alpha(N) \gg 1/q(N)$ (as in Proposition 16, where we used $q(N) = ((\log_2 N)/\alpha(N))^3$), 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 18 can be adapted to this context, and directly yields strongly inapproximable predicates. For details, see Exercise 35.

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.

[11] Needless to say, 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.

[12] Note that this is the "right" formulation, because 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 $N \gg \log q(N)$, we cannot hope to recover almost all N-bit long strings based on $\mathrm{poly}(q(N)\log N)$ bits of advice.

Theorem 19 *Suppose that there exists a decision problem $L \in \mathcal{E}$ 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 decides L on $\{0,1\}^n$ has size at least 2^{bk}. 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 19 can be used for deriving a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$) under the aforementioned assumption (see [9, Text 17]).

Theorem 19 follows as a special case of Proposition 16 (and the modification discussed right after it). An alternative proof, which uses different ideas that are of independent interest, will be briefly reviewed next. The starting point of this proof is a mildly inapproximable predicate, as provided by Theorem 12. However, here we cannot afford to apply Yao's XOR Lemma (i.e., Theorem 13), because the latter relates the circuit complexity of a *strongly* inapproximable predicate defined over poly(n)-bit long strings to the circuit complexity of a *mildly* inapproximable predicate defined over n-bit long strings. That is, 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, for some $S_F(\text{poly}(n))$ that 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 $S_F(m) = \exp(m^{\Omega(1)})$).

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.[13]

Teaching note: In continuation to Footnote 13, we note that there is a strong connection between the rest of this section and pseudorandom generators (see [9, Text 17]). On top of the aforementioned conceptual aspects, we will refer to pairwise independence generators, random walks on expanders , and even to the Nisan-Wigderson Construction.

The pivot of the proof is the notion of a hard region. 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. One important (and non-trivial) observation is that 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.

2.2.1 Hard regions

We actually generalize the notion of hard regions to arbitrary distributions. The important special case of uniform distributions is obtained by taking X_n to be U_n (i.e., the uniform distribution over $\{0,1\}^n$). In general, we only assume that $X_n \in \{0,1\}^n$.

[13] Indeed, this falls within the general paradigm discussed in [9, Text 17]. Furthermore, this suggestion provides another perspective on the connection between randomness and computational difficulty, which is the focus of much discussion in [9, Text 17].

Definition 20 (hard region relative to arbitrary distribution): *Let $f:\{0,1\}^* \to \{0,1\}$ be a Boolean predicate, $\{X_n\}$ 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 $\{X_n\}$ *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*
$$\mathsf{Pr}[C_n(X_n)=f(X_n)|X_n\in S] \leq \frac{1}{2}+\varepsilon(n).$$
- *We say that f has a* hard region of density $\rho(\cdot)$ *relative to $\{X_n\}$ 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 $\{X_n\}$ with respect to the above such that $\mathsf{Pr}[X_n\in S_n] \geq \rho(n)$.*

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 $\{U_n\}$ 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). But this trivial observation does not provide hard regions (with small advantage) for *mildly* inapproximable predicates. Providing such hard regions is the contents of the following theorem.

Theorem 21 (hard regions for mildly inapproximable predicates): *Let $f:\{0,1\}^*\to\{0,1\}$ be a Boolean predicate, $\{X_n\}$ be a probability ensemble, $s:\mathbb{N}\to\mathbb{N}$, and $\rho:\mathbb{N}\to[0,1]$ such that $\rho(n)>1/\mathrm{poly}(n)$. Suppose that, for every circuit C_n of size at most $s(n)$, it holds that $\mathsf{Pr}[C_n(X_n)= f(X_n)] \leq 1-\rho(n)$. Then, for every $\varepsilon:\mathbb{N}\to[0,1]$, the function f has a hard region of density $\rho'(\cdot)$ relative to $\{X_n\}$ 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 $\rho'(\cdot)$ relative to the uniform distribution (with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$).

Proof Sketch:[14] We start by proving a weaker statement; namely, that $\{X_n\}$ "dominates" an ensemble $\{Y_n\}$ such that f is strongly inapproximable on $\{Y_n\}$. For $\rho:\mathbb{N}\to[0,1]$, we say that $\{X_n\}$ ρ-dominates $\{Y_n\}$ if for every x it holds that $\mathsf{Pr}[X_n=x] \geq \rho(n)\cdot\mathsf{Pr}[Y_n=x]$. Fixing the function ρ (to the one provided by the theorem's hypothesis), in this case we also say that $\{Y_n\}$ is dominated by $\{X_n\}$. We say that $\{Y_n\}$ is critically dominated by $\{X_n\}$ if for every x either $\mathsf{Pr}[Y_n=x] = (1/\rho(n))\cdot\mathsf{Pr}[X_n=x]$ or $\mathsf{Pr}[Y_n=x]=0$.

The notion of domination and critical domination play a central role in the proof, which consists of two parts. In the first part (Claim 21.1), we prove the existence of a ensemble $\{Y_n\}$ that is dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Y_n\}$. In the second part (Claim 21.2), we prove that the existence of such a dominated ensemble implies the existence of an ensemble $\{Z_n\}$ that essentially is *critically* dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Z_n\}$. Finally, we note that such a critically dominated ensemble defines a hard region of f relative to $\{X_n\}$, and the theorem follows.

Claim 21.1: Under the hypothesis of the theorem it holds that there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that, for every $s'(n)$-size circuit C_n, it holds that
$$\mathsf{Pr}[C_n(Y_n)=f(Y_n)] \leq \frac{1}{2}+\frac{\varepsilon(n)}{2}. \tag{10}$$

Proof: We employ von Neumann's Min-Max Principle (cf. [18]) to a "game" between all critically dominated (by X_n) probability distributions and all possible $s'(n)$-size circuits.[15] We start by as-

[14] See details in [11, Apdx. A].

[15] We warn that this application of the min-max principle is somewhat non-straightforward.

suming, towards the contradiction, that for every distribution Y_n that is dominated by X_n there exists a $s'(n)$-size circuits C_n such that $\Pr[C_n(Y_n) = f(Y_n)] > 0.5+\varepsilon'(n)$, where $\varepsilon'(n) = \varepsilon(n)/2$. One key observation is that any distribution that is dominated by X_n can be written as a convex combination of critically dominated (by X_n) distributions. Considering an enumeration $Y_n^{(1)}, ..., Y_n^{(t)}$ of the critically dominated (by X_n) distributions, we conclude that for every distribution π on $[t]$ there exists a $s'(n)$-size circuits C_n such that

$$\sum_{i=1}^{t} \pi(i) \cdot \Pr[C_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n). \tag{11}$$

Consider a finite game between two players, where the first player selects a critically dominated (by X_n) distribution, and the second player selects a $s'(n)$-size circuit and obtains a payoff as determined by the corresponding success probability; that is, if the first player selects the i^{th} critically dominated distribution and the second player selects the circuit C then the payoff equals $\Pr[C(Y_n^{(i)}) = f(Y_n^{(i)})]$. Now, Eq. (11) means 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 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, D_n, on $s'(n)$-size circuits such that for every i it holds that

$$\Pr[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n), \tag{12}$$

where the probability refers both to the choice of the circuit D_n and to the random variable Y_n. Let $B_n = \{x : \Pr[D_n(x) = f(x)] \leq 0.5 + \varepsilon'(n)\}$. Then, $\Pr[X_n \in B_n] < \rho(n)$, because otherwise we reach a contradiction to Eq. (12) by defining Y_n such that $\Pr[Y_n = x] = \Pr[X_n = x]/\Pr[X_n \in B_n]$ if $x \in B_n$ and $\Pr[Y_n = x] = 0$ otherwise, and noting that Y_n is dominated by X_n and $\Pr[D_n(Y_n) = f(Y_n)] > 0.5 + \varepsilon'(n)$.[16] By employing standard amplification to D_n, we obtain a distribution D'_n over $\text{poly}(n/\varepsilon'(n)) \cdot s'(n)$-size circuits such that for every $x \in \{0,1\}^n \setminus B_n$ it holds that $\Pr[D'_n(x) = f(x)] > 1 - 2^{-n}$. It follows that there exists a $s(n)$-sized circuit C_n such that $C_n(x) = f(x)$ for every $x \in \{0,1\}^n \setminus B_n$, and it follows that $\Pr[C_n(X_n) = f(X_n)] \geq \Pr[X_n \in \{0,1\}^n \setminus B_n] > 1 - \rho(n)$, in contradiction to the theorem's hypothesis. The claim follows. □

Claim 21.2: If there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ and satisfies Eq. (10), then there exists a probability ensemble $\{Z_n\}$ that is ρ'-critically dominated by $\{X_n\}$ and satisfies Eq. (10) with $\varepsilon(n)/2$ replaced by $\varepsilon(n)$.

In other words, Claim 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 $2\varepsilon(\cdot)$, thus establishing the theorem. The proof of Claim 21.2 is by the Probabilistic Method (cf. [1]). Specifically, we select a set S_n at random by including each n-bit long string x with probability

$$p(x) \stackrel{\text{def}}{=} \frac{\rho(n) \cdot \Pr[Y_n = x]}{\Pr[X_n = x]} \leq 1 \tag{13}$$

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

[16] We use again the fact that any dominated distribution is a convex combination of critically dominated distributions.

circuits, showing that for each such circuit C_n, with probability $1-\exp(-s'(n)^2)$ over the choice of S_n, it holds that $|\mathsf{Pr}[C_n(X_n)=f(X_n)|X_n\in S_n] - \mathsf{Pr}[C_n(Y_n)=f(Y_n)]| < \varepsilon(n)/2$. For details see [11, Apdx. A]. □

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 13).

An alternative proof of Yao's XOR Lemma. Let f, p, and T be as in Theorem 13. Then, by Theorem 21, for $\rho'(n) = 1/3p(n)$ and $s'(n) = T(n)^{\Omega(1)}/\text{poly}(n)$, 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 $t(n) = n\cdot p(n)$ and F as in Theorem 13, with probability at least $1-(1-\rho'(n))^{t(n)} = 1-\exp(-\Omega(n))$, one of the $t(n)$ random n-bit 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 circuit C_n can approximate F 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

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

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 $C'_n(z) = C_n(x',z,x'')\oplus\sigma$), and the theorem follows. Actually, we derive a generalization of Theorem 13 asserting that *for any function T such that f is $(T,1/p)$-inapproximable it holds that F is T'-inapproximable for $T'(t(n)\cdot n) = s'(n) = T(n)^{\Omega(1)}/\text{poly}(n)$.*

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. This amplification utilizes a pairwise independence generator (see [9, Text 17]), G, that stretches $2n$-bit long seeds to sequences of n strings, each of length n.

Lemma 22 (derandomized XOR Lemma up to constant inapproximability): *Suppose that $f: \{0,1\}^*\to\{0,1\}$ is (T,ρ)-inapproximable, for $\rho(n) > 1/\text{poly}(n)$. 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 $\rho'(n') = \Omega(\min(n'\cdot\rho(n'/3),1))$.*

Needless to say, if $f\in\mathcal{E}$ then $F_1\in\mathcal{E}$.

Proof Sketch: Again, by Theorem 21, for $\alpha(n) = \rho(n)/3$ and $s'(n) = T(n)/\text{poly}(n)$, the function f has a hard region S of density α (relative to $\{U_n\}$) with respect to $s'(\cdot)$-size circuits and advantage 0.01. Consider the function $P_1(s) = (f(x_1),...,f(x_n))$, where $|s| = 2n$ and $(x_1,...,x_n) = G(s)$. By Exercise 36, with probability at least $\beta(n) \stackrel{\text{def}}{=} \min(n\cdot\alpha(n),1)/2$, at least one of the n strings output by $G(U_{2n})$ resides in S. Intuitively, we expect every $s'(n)$-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. Things are somewhat more involved (than in the non-derandomized case) because it is not clear what is the conditional distribution on the hard region.

For simplicity[17] (and without loss of generality), we assume that $\alpha(n) < 1/2n$, and note that in this case 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 claim that every $(s'(n) - \text{poly}(n))$-sized circuit fails to compute P_1 correctly with probability at least $\gamma(n) = 0.2\beta(n)$. 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,

$$\begin{aligned}
\Pr[C'_n(U_n) = f(U_n) | U_n \in S] &\geq \sum_{i=1}^{n} \frac{1}{n} \cdot \Pr[C_n(U_{2n}) = P_1(U_{2n}) | G(U_{2n})_i \in S] \\
&\geq \frac{1}{n} \cdot \frac{\Pr[C_n(U_{2n}) = P_1(U_{2n}) \wedge \exists i\, G_i(U_{2n})_i \in S]}{\max_i\{\Pr[G(U_{2n})_i \in S]\}} \\
&\geq \frac{1}{n} \cdot \frac{(1 - \gamma(n)) - (1 - \beta(n))}{\alpha(n)} \;=\; \frac{0.8\beta(n)}{n \cdot \alpha(n)} \;=\; 0.6
\end{aligned}$$

contradicting the fact that S is a hard region of f with respect to $s'(\cdot)$-size circuits and advantage 0.01. Employing the simple (warm-up) case discussed at the beginning of the proof of Theorem 7, it follows that, for $s''(n') = s(n'/3)/\text{poly}(n')$, every $s''(|s| + |r|)$-sized circuits fails to compute $(s, r) \mapsto b(P_1(s), r)$ with probability at least $\delta(|s| + |r|) \stackrel{\text{def}}{=} 0.24 \cdot \gamma(|r|)$. Thus, F_1 is $(s'', 2\delta)$-inapproximable, and the lemma follows. $\square$

The next lemma offers an amplification of constant inapproximability to strong inapproximability. Indeed, combining Theorem 12 with Lemmas 22 and 23, yields Theorem 19 (as a special case).

Lemma 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 a 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:[18] As in the proof of Lemma 22, we start with a hard region of density $\rho/3$ (and advantage $1/T'$) for f and focus on the analysis of 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 [9, Text 17]. 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

[17] The choice of some of the constants in the following argument is rather arbitrary. In general, assuming that $\alpha(n) < c/n$, for some constant $c \in (0,1)$, we can set $\beta(n) = (1 - (c/2)) \cdot n \cdot \alpha(n)$. This allows setting $\gamma(n) = c'\beta(n)$ for c' satisfying $(1 - c')(1 - (c/2)) \geq (0.5 + \varepsilon)$, where ε is the advantage in the definition of the hard region. (We have used $\varepsilon = 0.01$, $c = 0.5$ and $c' = 0.2$.) Finally, in moving from P_1 to F_1 we lose a factor that can be made arbitrarily close to 4.

[18] For details, see [15].

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 case the the x_i's that hit the hard region are drawn independently from 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 the Nisan-Wigderson Generator; 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 $n/O(1)$.[19] An analysis as the one applied to the Nisan-Wigderson Generator 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 the relevant property of G_2 is inherited by G.)

The actual analysis of the construction (via a guessing game presented in [15, 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 [15]. $\square$

Digest. Both Lemmas 22 and 23 are proved by first establishing corresponding "direct product" versions (i.e., derandomized versions of Theorem 14). We call the reader's attention to the seemingly crucial role of this step (especially in the proof of Lemma 23): We cannot treat the values $f(x_1), ...f(x_n)$ as independent (at least not for the generator G as postulated in the proof), 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 34 fails here, becuase the x_i's are not independent.

Notes

The notion of a one-way function was suggested by Diffie and Hellman [5]. The notion of weak one-way functions as well as the amplification of one-way functions (Theorem 5) were suggested by Yao [19]. A proof of Theorem 5 has first appeared in [6].

The concept of hard-core predicates was suggested by Blum and Micali [2]. 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 (Theorem 7) was suggested by Levin, and proven as such by Goldreich and Levin [10]. The proof presented here was suggested by Rackoff. We comment that the original proof has its own merits (cf., e.g., [12]).

The construction of canonical derandomizers and, specifically, the Nisan-Wigderson framework has been the driving force behind the study of inapproximable predicates in $\mathcal{E}$. Theorem 10 is due to [4], whereas Theorem 19 is due to [15]. Both results rely heavily of variants of Yao's XOR Lemma, to be reviewed next.

[19] 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 $s_S = \sigma_{i_1} \cdots \sigma_{i_n}$.

Like several other fundamental insights attributed to Yao's paper [19], Yao's XOR Lemma (Theorem 13) is not even stated in [19] but is rather due to Yao's oral presentations of his paper. The first published proof of Yao's XOR Lemma was given by Levin (see [11, Sec. 3]). Levin's proof is the only one known giving a tight quantitative analysis (on the decrease in the level of approximability), and the interested reader is referred to it (via the non-laconic presentation of [11, Sec. 3]). The proof presented in §2.1.2 is due to Goldreich, Nisan and Wigderson [11, Sec. 5].

The notion of a hard region and its applications to proving the original version of Yao's XOR Lemma as well as the first derandomization of it (Lemma 22) are due to Impagliazzo [14]. The second derandomization (Lemma 23) as well as Theorem 19 are due to Impagliazzo and Wigderson [15].

The connection between list decoding and hardness amplification (Section 2.1.3), yielding an alternative proof of Theorem 19, is due to Sudan, Trevisan, and Vadhan [17].

Hardness amplification for $\mathcal{NP}$ has been the subject of recent attention: An amplification of mild inapproximablity to strong inapproximablity is provided in [13], an indication to the impossibility of a worst-case to average-case reductions (at least non-adaptive ones) is provided in [3].

Exercises

Exercise 24 Prove that if one way-functions exist then there exists one-way functions that are length preserving (i.e., $|f(x)| = |x|$ for every $x \in \{0,1\}^n$).
(Hint: Clearly, for some polynomial p, it holds that $|f(x)| \leq p(|x|)$ for all x. Assume, without loss of generality that $n \mapsto p(n)$ is 1-1, and let $p^{-1}(m) = n$ if $p(n) \leq m < p(n+1)$. Define $f'(z) = f(x)0^{|z|-|f(x)|}$, where x is the $p^{-1}(|z|)$-bit long prefix of z.)

Exercise 25 Prove that if a function f is hard to invert in the sense of Definition 3 then it is hard to invert in the sense of Definition 1.
(Hint: consider a sequence of internal coin tosses that maximizes the probability in Eq. (1).)

Exercise 26 Assuming the existence of one-way functions, prove that there exists a weak one-way function that is not strongly one-way.

Exercise 27 (a universal one-way function) Using the notion of a universal machine, present a polynomial-time computable that is hard to invert (in the sense of Definition 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 28 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 $r^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$.
(Hint: 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 [9, Text 17].)

Exercise 29 Prove Theorem 8. In particular, provide a detailed presentation of the alternative procedure outlined in Footnote 5.

Exercise 30 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 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.

(Hint: 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 31 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 some problem in $\mathcal{E}$ does not have circuits of size S', where $S'(n) = S(n^\varepsilon)$ for some constant $\varepsilon > 0$.

(Hint: SAT is in $\mathcal{E}$.)

Exercise 32 For every function $f : \{0,1\}^n \to \{0,1\}$, present a linear-size circuit C_n such that $\Pr[C(U_n) = f(U_n)] \geq 0.5 + O(2^{-n})$. Furthermore, for every $t \leq 2^{n-1}$, present a circuit C_n of size $O(t \cdot n)$ such that $\Pr[C(U_n) = f(U_n)] \geq 0.5 + t \cdot 2^{-n}$. Warning: you may not assume that $\Pr[f(U_n) = 1] = 0.5$.

Exercise 33 Let $\hat{f}$ be as in the conclusion of Theorem 12. Prove that there exists a Boolean function g in $\mathcal{E}$ that is (p, ε)-inapproximable for every polynomial p and for $\varepsilon(n) = 1/n^3$.

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

Exercise 34 Let f be a Boolean function, and $b(y,r)$ denote the inner-product modulo 2 of the equal-length strings y and r. Suppose that $F'(x_1, ..., x_{t(n)}, r) \stackrel{\text{def}}{=} b(f(x_1) \cdots f(x_{t(n)}), r)$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $r \in \{0,1\}^{t(n)}$, is T-inapproximable for every polynomial T. Assuming that $n \mapsto t(n) \cdot n$ is 1-1, prove that $F(x) \stackrel{\text{def}}{=} F'(x, 1^{t'(|x|)})$, where $t'(t(n) \cdot n) = t(n)$, is T-inapproximable for every polynomial T.

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 $F'(x,r) = F(x') \oplus \oplus_{i:r_i=0} f(x'_i)$. Note that the equality holds regardless of the choice of the string $x'_i \in \{0,1\}^n$ for which $r_i = 0$. Also note that the suggested reduction requires knowledge of $\sigma = \oplus_{i:r_i=0} f(x'_i)$, but in our context the reduction may be performed by a small non-uniform circuit, which may incorporate the values of $f(z)$'s for a small number of z's. Indeed, for uniformly chosen $z_1, ..., z_{t(n)} \in \{0,1\}^n$, we use these z_i's as well as the $f(z_i)$'s as advice to the reduction. 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 $F(x') \oplus_{i:r_i=0} f(z_i)$.

Exercise 35 Consider a modification of Definition 17, in which the decoding condition 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 18 that refers to the modified definition (rather than to the original one).

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

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

3. Prove that the Hadamard Code allows implicit decoding under the modified definition (but not according to the original one).[20]

   (Hint: this is the actual contents of Theorem 8.)

Note that encoding the symbols of a non-binary code Γ that allows implicit decoding with the Hadamard code yields a binary code that allows implicit decoding. Note that efficient encoding is preserved only if $q(N) = \text{poly}(N)$.

Exercise 36 Let G be as in Lemma 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.

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

Exercise 37 (one-way functions versus inapproximable predicates) Prove that the existence of a non-uniformly hard one-way function (as in Definition 3) implies the existence of an exponential-time computable predicate that is T-inapproximable (as per Definition 9), for every polynomial T.

Guideline: Suppose first that the one-way function f is length-preserving and 1-1. Consider the corresponding function g and hard-core predicate b guaranteed by Theorem 7, and show that the Boolean function h such that $h(z) = b(g^{-1}(z))$ is polynomially inapproximable. 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 the i^{th} bit of the lexicographically first element in $f^{-1}(z) = \{x : f(x) = z\}$ equals σ. Note that h is computable in exponential-time, but is not (worst-case) computable in polynomial-time. Applying Theorem 10, we are done.

References

[1] N. Alon and J.H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992.

[2] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[3] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. In *Proc. 44th IEEE Symposium on Foundations of Computer Science*, pages 308–317, 2003.

[4] L. Babai, L. Fortnow, N. Nisan and A. Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[5] W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22 (Nov. 1976), pages 644–654.

[6] O. Goldreich. *Foundation of Cryptography – Class Notes.* Computer Science Dept., Technion, Israel, Spring 1989. Superseded by [7, 8].

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

[7] O. Goldreich. *Foundation of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[8] O. Goldreich. *Foundation of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[9] O. Goldreich. Expositions in Complexity Theory (various texts). Unpublished notes, December 2005. Availabe from the webpage `http://www.wisdom.weizmann.ac.il/~oded/cc-texts.html`

[10] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[11] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[12] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. Discrete Math.*, Vol. 13 (4), pages 535–570, 2000.

[13] A. Healy, S. Vadhan and E. Viola. Using nondeterminism to amplify hardness. In *36th ACM Symposium on the Theory of Computing*, pages 192–201, 2004.

[14] R. Impagliazzo. Hard-core Distributions for Somewhat Hard Problems. In *36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[15] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[16] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.

[17] , M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. *Journal of Computer and System Science*, Vol. 62, No. 2, pages 236–266, 2001.

[18] J. von Neumann, Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, pages 295–320, 1928.

[19] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.