Chapter 8

Pseudorandom Generators

Indistinguishable things are identical.[1]

G.W. Leibniz (1646–1714)

A fresh view at the *question of randomness* has been taken in the theory of computing: It has been postulated that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by any efficient procedure. The paradigm, originally associating efficient procedures with polynomial-time algorithms, has been applied also with respect to a variety of limited classes of such distinguishing procedures.

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

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

The archetypical case of pseudorandom generators refers to efficient generators that fool any feasible procedure; that is, the potential distinguisher is any probabilistic polynomial-time algorithm, which may

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

be more complex than the generator itself (which, in turn, has time-complexity bounded by a fixed polynomial). These generators are called general-purpose, because their output can be safely used in an efficient application. Such (general-purpose) pseudorandom generators exist if and only if one-way functions exist.

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

It is also beneficial to consider pseudorandom generators that fool space-bounded distinguishers and generators that exhibit some limited random behavior (e.g., outputting a pair-wise independent or a small-bias sequence).

8.1 Introduction

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf., [60]), initiated by Shannon [189], is rooted in probability theory and is focused at distributions that are not perfectly random. Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., there is no redundancy at all). Thus, perfect randomness is associated with a unique distribution – the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [143, 146]), due to Solomonov [196], Kolmogorov [137] and Chaitin [48], is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device; see §1.2.3.3). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object. Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable), and – by definition – one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory is rooted in complexity theory and is the focus of this chapter. This approach is explicitly aimed at providing a notion of randomness that nevertheless allows for an efficient (and deterministic) generation of random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" in one of the following four ways. In each of them Alice flips an unbiased coin and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess.
>
> In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2.
>
> In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2.
>
> The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess.
>
> In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept of pseudorandomness arises – a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This notion of pseudorandomness is indeed the most fundamental one, and much of this chapter is focused on it. Weaker notions of pseudorandomness arise as well – they refer to indistinguishability by weaker procedures such as space-bounded algorithms, constant-depth circuits, etc. Stretching this approach even further one may consider algorithms that are designed on purpose so not to distinguish even weaker forms of "pseudorandom" sequences from random ones (such algorithms arise naturally when trying to convert some natural randomized algorithm into deterministic ones; see Section 8.6).

The foregoing discussion has focused at one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). A natural approach is that the generation process has to be at least as efficient as the distinguisher (equiv., that the distinguisher is allowed at least as much resources as the generator). Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – these operating in polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. Such (general-purpose) pseudorandom generators allow to reduced the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography (see Section 8.3).

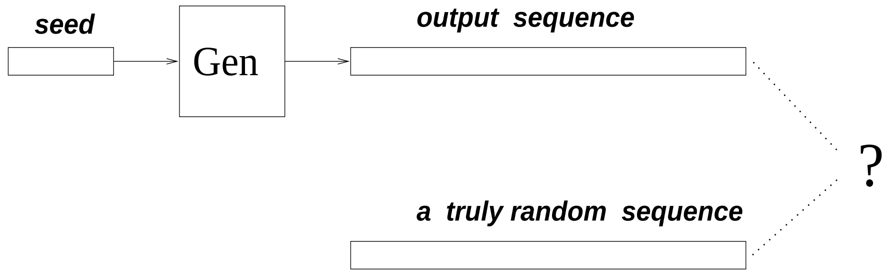


Figure 8.1: Pseudorandom generators – an illustration.

We stress that there are important reasons for considering also an alternative that seems less natural; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones), where the crucial step is replacing the random input of an algorithm by a pseudorandom input, which in turn can be generated based on a much shorter random seed. In particular, when derandomizing a probabilistic polynomial-time algorithm, the observer (to be fooled by the genewrator) is a fixed algorithm. In this case employing a more complex generator merely means that the complexity of the derived deterministic algorithm is dominated by the complexity of the generator (rather than by the complexity of the original randomized algorithm). Needless to say, allowing the generator to use more resources than the observer it tries to fool makes the task of designing pseudorandom generators easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators. The usefulness of this approach is demonstrated in Sections 8.4 through 8.6.

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

Organization. In Section 8.2 we present the general paradigm underlying the various notions of pseudorandom generators. The archetypical case of general-purpose pseudorandom generators is presented in Section 8.3. We then turn to the alternative notions of pseudorandom generators: Generators that suffice for the derandomization of complexity classes such as $\mathcal{BPP}$ are discussed in Section 8.4; Pseudorandom generators in the domain of space-bounded computations are discussed in Section 8.5; and special-purpose generators are discussed in Section 8.6. (For an alternative presentation, which focuses on general-purpose pseudorandom generators and provides more details on it, the reader is referred to [87, Chap. 3].)

Teaching note: If you can afford teaching only one of the alternative notions of pseudorandom generators, then we suggest teaching the notion of general-purpose pseudorandom generators (presented in Section 8.3). Our reasons being that this notion is relevant to computer science at large and that the technical material is relatively simpler. The chapter is organized to facilitate this option.

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

8.2 The General Paradigm

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

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

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that it is a deterministic algorithm that stretches short strings, called seeds, into longer output sequences. Specifically, it stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell : \mathbb{N} \to \mathbb{N}$ is called the stretch measure (or stretch function). In some settings the specific stretch measure is immaterial (e.g., see Section 8.3.4).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that it "fools" some non-trivial algorithms. That is, any algorithm taken from some class of interest cannot distinguish the output produced by the generator (when the generator is fed with a uniformly chosen seed) from a uniformly chosen sequence. Typically, we consider a class $\mathcal{D}$ of distinguishers and a class $\mathcal{F}$ of (threshold) functions, and require that the generator G satisfies the following: For any $D \in \mathcal{D}$, any $f \in \mathcal{F}$, and for all sufficiently large k's

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

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic.[2] The reader may think of such a distinguisher, D, as trying to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed as $U_{\ell(k)}$). The condition in Eq. (8.1) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases. The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

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

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

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

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

1. General-purpose pseudorandom generators correspond to the case that the generator itself runs in polynomial time and needs to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for more time than the generator. Thus, the same generator may be used safely in any efficient application.

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

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

8.3 General-Purpose Pseudorandom Generators

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

8.3.1 The basic definition

Loosely speaking, general-purpose pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by *any* efficient (i.e., polynomial-time) algorithm. Thus, the distinguisher is more complex than the generator: The generator is a fixed algorithm working within *some fixed* polynomial-time, whereas a potential distinguisher is *any* algorithm that runs in polynomial-time. Thus, for example, the distinguisher *may* always run in time cubic in the running-time of the generator. Furthermore, to facilitate the development of this theory, we allow the distinguisher to be probabilistic (whereas the generator remains deterministic as above). We require that such distinguishers cannot tell the output of the generator from a truly random string of similar length, or rather that the difference that such distinguishers may detect (or sense) is negligible. Here a negligible function is one that vanishes faster than the reciprocal of any positive polynomial.

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

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

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

Thus, Definition 8.1 is derived from the generic framework (presented in Section 8.2) by taking the class of distinguishers to be the set of all probabilistic polynomial-time algorithms, and taking the class of (noticeable) threshold functions to be the set of all functions that are the reciprocals of some positive polynomial.[3] The latter choice is naturally coupled with the association of efficient computation with polynomial-time algorithms: An event that occurs with noticeable probability occurs almost always when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

We note that Definition 8.1 does not make any requirement regarding the stretch function $\ell : \mathbb{N} \to \mathbb{N}$, except for the generic requirement that $\ell(k) > k$ for all k. Needless to say, the larger ℓ is the more useful is the pseudorandom generator. In Section 8.3.4 we show how to use any pseudorandom generator (even one with minimal stretch $\ell(k) = k+1$) in order to obtain a pseudorandom generator of any desired polynomial stretch function. But before going so, we rigorously discuss the "reduction in randomness" offered by pseudorandom generators, and the notion of computational indistinguishability underlying Definition 8.1.

8.3.2 The archetypical application

We note that "pseudo-random number generators" appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to Definition 8.1. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test (cf., [136]). We warn that the fact that a "pseudorandom number generator" passes some statistical tests, does not mean that it will pass a new test and that it will be good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses." In contrast, the approach encompassed in Definition 8.1 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 8.1, covers all possible efficient applications guaranteeing that for all of them pseudorandom sequences are as good as truly random ones. Indeed, any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. That is:

[3] Definition 8.1 requires that the distinguishing gap of certain algorithms must be smaller than the reciprocal of any positive polynomial for all but finitely many k's. Such functions are called *negligible*; see Footnote 2. The notion of negligible probability is robust in the sense that an event which occurs with negligible probability occurs with negligible probability also when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

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

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

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

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

Proposition 8.3 *Let A, ρ and G be as in Construction 8.2, and suppose that $\rho:\mathbb{N}\to\mathbb{N}$ is 1-1. Then, for every pair of probabilistic polynomial-time algorithms,* a finder F *and* a distinguisher D, *every positive polynomial p and all sufficiently long n's*

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

where $\Delta_{A,D}(x) \stackrel{\text{def}}{=} \mathsf{Pr}[D(x,A(x,U_{\rho(|x|)}))=1] - \mathsf{Pr}[D(x,A_G(x,U_{k(|x|)}))=1]$, and the probabilities are taken over the U_m's as well as over the coin tosses of F and D.

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

Proof: The proposition is proven by showing that any triplet (A, F, D) violating the claim can be converted into an algorithm D' that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. The key observation is that $\Delta_{A,D}(x)$ equals $\mathsf{Pr}[D(x, A(x, U_{\rho(n)})) = 1] - \mathsf{Pr}[D(x, A(x, G'(U_{k(n)}))) = 1]$, where $G'(s)$ is the $\rho(n)$-bit long prefix of $G(s)$. Details follow.

On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), algorithm D' first obtains $x \leftarrow F(1^n)$, where n can be obtained easily from $|r|$ (because ρ is 1-1 and $1^n \mapsto \rho(n)$ is computable via A). Next, D' obtains $y = A(x, r')$, where r' is the $\rho(|x|)$-bit long prefix of r. Finally D' outputs $D(x, y)$. Note that D' is implementable in probabilistic polynomial-time, and that

$$\begin{aligned} D'(U_{\rho(n)}) &= D(X_n, A(X_n, U_{\rho(n)}))\,, \text{ where } X_n \stackrel{\text{def}}{=} F(1^n) \\ D'(G'(U_{k(n)})) &= D(X_n, A(X_n, G'(U_{k(n)})))\,, \text{ where } X_n \stackrel{\text{def}}{=} F(1^n) \end{aligned}$$

It follows that $|\,\mathsf{Pr}[D'(U_{\ell(k(n))}) = 1] - \mathsf{Pr}[D'(G(U_{k(n)})) = 1]\,|$ equals $\mathsf{E}[\Delta_{A,D}(F(1^n))]$, which implies a weaker version of the proposition (referring to $\mathsf{E}[\Delta_{A,D}(F(1^n))]$ rather than to $\mathsf{E}[|\Delta_{A,D}(F(1^n))|]$).

In order to prove that $\mathsf{E}[|\Delta_{A,D}(F(1^n))|]$ (rather than to $\mathsf{E}[\Delta_{A,D}(F(1^n))]$) is negligible, we need to modify D' a little. We start by assuming, towards the contradiction, that $\mathsf{E}[|\Delta_{A,D}(F(1^n))|] > \varepsilon(n)$ for some non-negligible function ε. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), the modified algorithm D' first obtains $x \leftarrow F(1^n)$, as before. Next, using a sample of size $\text{poly}(n/\varepsilon(n))$, it approximates $p_U(x) \stackrel{\text{def}}{=} \mathsf{Pr}[D(x, A(x, U_{\rho(n)}) = 1]$ and $p_G(x) \stackrel{\text{def}}{=} \mathsf{Pr}[D(x, A(x, G'(U_{k(n)})) = 1]$ such that each probability is approximated to within a deviation of $\varepsilon(n)/8$ with negligible error probability (say, $\exp(-n)$). (Note that, so far, the actions of D' only depend on the length of its input r, which determines n.) If these approximations indicate that $p_U(x) \geq p_G(x)$ (equiv., that $\Delta_{A,D} \geq 0$) then D' outputs $D(x, A(x, r'))$ else it outputs $1 - D(x, A(x, r'))$, where r' is the $\rho(|x|)$-bit long prefix of r and we assume without loss of generality that the output of D is in $\{0, 1\}$. It follows that

$$\begin{aligned} &\mathsf{Pr}[D'(U_{\rho(n)}) = 1 | F(1^n) = x] \;-\; \mathsf{Pr}[D'(G'(U_{k(n)})) = 1 | F(1^n) = x] \\ &\geq \;\; |p_U(x) - p_G(x)| - \frac{\varepsilon(n)}{2} - \exp(-n) \end{aligned}$$

where the $\varepsilon(n)/2$ term is due to the maximal typical deviation (i.e., $\varepsilon(n)/4$) of our approximation of $p_U(x) - p_G(x)$, and the $\exp(-n)$ term is due to the rare case that our approximation errs by more than $\varepsilon(n)/4$. Thus, $\mathsf{Pr}[D'(U_{\ell(k(n))}) = 1] - \mathsf{Pr}[D'(G(U_{k(n)})) = 1]$ is lower-bounded by $\mathsf{E}[|\Delta_{A,D}(F(1^n))|] - (\varepsilon(n)/2) - \exp(-n) > \varepsilon(n)/3$, and the proposition follows. ■

Conclusion. Analogous arguments are applied whenever one wishes to prove that an efficient randomized process (be it an algorithm as above or a multi-party computation) preserves its behavior when one replaces true randomness by pseudorandomness as defined above. Thus, given a pseudorandom generator with a large stretch function, *one can considerably reduce the randomness complexity in any efficient application.*

8.3.3 Computational Indistinguishability

In this section we spell-out (and study) the definition of computational indistinguishability that underlies Definition 8.1. The general definition of computational indistinguishability refers to *arbitrary* probability ensembles, where a probability ensemble is an infinite sequence of random variables $\{Z_n\}_{n\in\mathbb{N}}$ such that each Z_n ranges over strings of length bounded by a polynomial in n. We say that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \stackrel{\rm def}{=} |\Pr[A(X_n)=1] - \Pr[A(Y_n)=1]|$ is a negligible function in n. That is:

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

$$|\Pr[D(X_n)=1] - \Pr[D(Y_n)=1]| \; < \; \frac{1}{p(n)} \tag{8.4}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm D. The l.h.s. of Eq.* (8.4)*, when viewed as a function of n, is often called the* distinguishing gap of D *where $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ is understood from the context.*

That is, we can think of D as somebody who wishes to distinguish two distributions (based on a sample given to it), and think of 1 as D's verdict that the sample was drawn according to the first distribution. Saying that the two distributions are computationally indistinguishable means that if D is a feasible procedure then its verdict is not really meaningful (because the verdict is almost as often 1 when the input is drawn from the first distribution as when the input is drawn from the second distribution). We comment that the absolute value in Eq. (8.4) can be omitted without affecting the definition (see Exercise 8.3), and we will often do so without warning.

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

Non-triviality of Computational Indistinguishability. Clearly, any two probability ensembles that are statistically close[4] are computationally indistinguishable. Needless to say, this is a trivial case of computational indistinguishability, which is due to information theoretic reasons. In contrast, as noted in Section 8.2, there exist probability ensembles that are statistically far apart and yet are computationally

[4]Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are said to be statistically close if for every positive polynomial p and sufficient large n the variation distance between X_n and Y_n (i.e., $\frac{1}{2}\sum_z |\Pr[X_n = z] - \Pr[Y_n = z]|$) is bounded above by $1/p(n)$.

indistinguishable (see Exercise 8.1). However, at least one of the probability ensembles in Exercise 8.1 is *not* polynomial-time constructible. One non-trivial case of computational indistinguishability in which both ensembles are polynomial-time constructible is provided by the definition of pseudorandom generators (see Exercise 8.4). As we shall see (in Theorem 8.11), the existence of one-way functions implies the existence of pseudorandom generators, which in turn implies the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable. We mention that this sufficient condition is also necessary (see Exercise 8.5).

Indistinguishability by Multiple Samples

The definition of computational indistinguishability (i.e., Definition 8.4) refers to distinguishers that obtain a single sample from one of the two probability ensembles (i.e., $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$). A more general definition refers to distinguishers that obtain several independent samples from such an ensemble.

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

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

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

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

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

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

Proof Sketch:[5] To prove that a sequence of independently drawn samples of one distribution is indistinguishable from a sequence of independently drawn samples

[5] For more details see [87, Sec. 3.2.3].

from the other distribution, we consider *hybrid* sequences such that the i^{th} hybrid consists of i samples taken from the first distribution and the rest taken from the second distribution. The "homogeneous" sequences (which we wish to prove to be computational indistinguishable) are the extreme hybrids (i.e., the first and last hybrids considered above). The key observation is that distinguishing the extreme hybrids (towards the contradiction hypothesis) means distinguishing neighboring hybrids, which in turn yields a procedure for distinguishing single samples of the two original distributions (contradicting the hypothesis that these two distributions are indistinguishable by a single sample). Details follow.

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

$$\begin{aligned}
&\mathsf{E}_{i\in\{0,\ldots,s(n)-1\}}\left[\mathsf{Pr}[D(H_n^i)=1]-\mathsf{Pr}[D(H_n^{i+1})=1]\right] \\
&= \frac{1}{s(n)}\cdot\sum_{i=0}^{s(n)-1}\left(\mathsf{Pr}[D(H_n^i)=1]-\mathsf{Pr}[D(H_n^{i+1})=1]\right) \qquad (8.5)\\
&= \frac{1}{s(n)}\cdot\left(\mathsf{Pr}[D(H_n^0)=1]-\mathsf{Pr}[D(H_n^{s(n)})=1]\right) \;=\; \frac{\delta(n)}{s(n)}
\end{aligned}$$

Using D, we obtain a distinguisher D' of single samples: Given a single sample, D' selects $i \in \{0, ..., s(n)-1\}$ at random, generates i samples from the first distribution and $s(n)-i-1$ samples from the second distribution, and invokes D with the $s(n)$-samples sequence obtained when placing the input sample in location $i+1$. Thus, the construction of D' relies on the hypothesis that both proabbility ensembles are polynomial-time constructible. In analyzing D', observe that when the single sample (i.e., the input to D') is taken from the first (resp., second) distribution, algorithm D' invokes D on the $i+1^{\text{st}}$ hybrid (resp., i^{th} hybrid). Thus, the distinguishing gap of D' is captured by Eq. (8.5), and the claim follows. $\square$

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

1. *The extreme hybrids collide with the complex ensembles*: this property is essential because what we want to prove (i.e., the indistinguishability of the

complex ensembles) relates to the complex ensembles.

2. *Neighboring hybrids are easily related to the basic ensembles*: this property is essential because what we know (i.e., the indistinguishability of the basic ensembles) relates to the basic ensembles. We need to be able to translate our knowledge (i.e., computational indistinguishability) of the basic ensembles to knowledge (i.e., computational indistinguishability) of any pair of neighboring hybrids. Typically, it is required to efficiently transform strings in the range of a basic distribution into strings in the range of a hybrid, so that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighboring hybrid. (In the proof of Proposition 8.6, the hypothesis that both X and Y are polynomial-time constructible is instrumental for such an efficient transformation.)

3. *The number of hybrids is small* (i.e., polynomial): this property is essential in order to deduce the computational indistinguishability of extreme hybrids from the computational indistinguishability of each pair of neighboring hybrids. Typically, the provable "distinguishability gap" is inversely proportional to the number of hybrids. Indeed, see Eq. (8.5).

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

8.3.4 Amplifying the stretch function

Recall that the definition of pseudorandom generators (i.e., Definition 8.1) makes a minimal requirement regarding their stretch; that is, it is only required that the length of the output of such generators is longer than their input. Needless to say, we seek pseudorandom generators with a significant stretch. It turns out (see Construction 8.7) that pseudorandom generators of any stretch function and in particular of stretch $\ell_1(k) \stackrel{\rm def}{=} k+1$, are easily converted into pseudorandom generators of any desired (polynomially bounded) stretch function, ℓ. (On the other hand, since pseudorandom generators are required (in Definition 8.1) to run in polynomial time, their stretch must be polynomially bounded.) Thus, when talking about the existence of pseudorandom generators, as in Definition 8.1, we may ignore the stretch function.

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

$$G(s) \stackrel{\rm def}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)} \tag{8.6}$$

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

Needless to say, G is polynomial-time computable and has stretch ℓ.

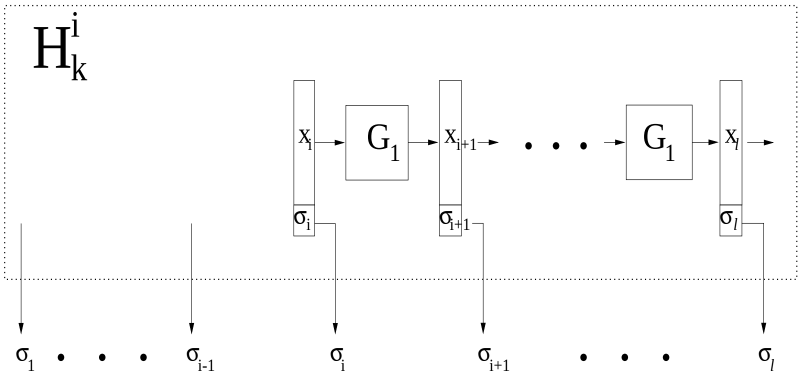


Figure 8.2: Analysis of stretch amplification – the $i^{\rm th}$ hybrid.

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

Proof Sketch:[6] The proposition is proven using the *hybrid technique*, presented and discussed in Section 8.3.3. Here (for $i = 0, ..., \ell(k)$) we consider the hybrid distributions H_k^i, defined by

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

where $U_i^{(1)}$ and $U_k^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^k$, respectively), and $P_j(x)$ denotes the j-bit long prefix of $G(x)$. (See Figure 8.2.) The extreme hybrids (i.e., H_k^0 and H_k^k) correspond to $G(U_k)$ and $U_{\ell(k)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G_1(U_k)$ and U_{k+1}. Details follow.

Suppose that algorithm D distinguishes H_k^i from H_k^{i+1} (with some gap $\delta(k)$). Denoting the first $|x|-1$ bits (resp., last bit) of x by $F(x)$ (resp., $L(x)$), we may write $P_j(s) \equiv (L(G_1(s)), P_{j-1}(F(G_1(s))))$ and

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

Then, incorporating the generation of $U_i^{(1)}$ and the evaluation of $P_{\ell(k)-i-1}$ into the distinguisher D, we distinguish $(F(G_1(U_k^{(2)})), L(G_1(U_k^{(2)}))) \equiv G_1(U_k)$ from

[6]For more details see [87, Sec. 3.3.3].

$(F(U_{k+1}^{(2')}), L(U_{k+1}^{(2')})) \equiv U_{k+1}$, in contradiction to the pseudorandomness of G_1. Specifically, on input $x \in \{0,1\}^{k+1}$, we uniformly select $r \in \{0,1\}^i$ and output $D(r \cdot L(x) \cdot P_{\ell(k)-i-1}(F(x)))$. Thus, the probability we output 1 on input $G_1(U_k)$ (resp., U_{k+1}) equals $\mathsf{Pr}[D(H_k^i) = 1]$ (resp., $\mathsf{Pr}[D(H_k^{i+1}) = 1]$). A final detail refers to the question which i to use. As usual (when the hybrid technique is used), a random i (in $\{0, ..., k-1\}$) will do. $\square$

8.3.5 Constructions

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

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

Proof Sketch:[7] The $|s|$-bit long prefix of $G(s)$ is uniformly distributed, because f is 1-1 and onto $\{0,1\}^{|s|}$. Hence, the proof boils down to showing that distinguishing $f(s)b(s)$ from $f(s)\cdot\sigma$, where σ is a random bit, yields contradiction to the hypothesis that b is a hard-core of f (i.e., that $b(s)$ is *unpredictable* from $f(s)$). Intuitively, such a distinguisher also distinguishes $f(s)b(s)$ from $f(s) \cdot \overline{b(s)}$, where $\overline{\sigma} = 1 - \sigma$, but distinguishing $f(s) \cdot b(s)$ from $f(s) \cdot \overline{b(s)}$ yields an algorithm for predicting $b(s)$ based on $f(s)$. Details follow.

We start with any potential distinguisher D, and let

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

We may assume, without loss of generality, that $\delta(k)$ is non-negative (for infinitely many k's). Using $G(U_k) = f(U_k) \cdot b(U_k)$ and $U_{k+1} \equiv f(U_k) \cdot Z$, where $Z = b(U_k)$ with probability 1/2 and $Z = \overline{b(U_k)}$ otherwise, we have

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

Consider an algorithm A that, on input y, uniformly selects $\sigma \in \{0,1\}$, invokes $D(y\sigma)$, and outputs σ if $D(y\sigma) = 1$ and $\overline{\sigma}$ otherwise. Then

$$\mathsf{Pr}[A(f(U_k)) = b(U_k)] \;=\; \mathsf{Pr}[D(f(U_k) \cdot \sigma) = 1 \wedge \sigma = b(U_k)]$$

[7]For more details see [87, Sec. 3.3.4].

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

which equals $(1+2\delta(k))/2$. The proposition follows. $\square$

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

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

Digest. The key point in the proof of Proposition 8.9 is showing that the (rather obvious) unpredictability of the output of G implies its pseudorandomness. The fact that (next bit) unpredictability and pseudorandomness are equivalent, in general, is proven explicitly in the alternative proof of Theorem 8.10 provided next.

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

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)}\,, \tag{8.7}$$

where $x_0 = s$ and $x_i\sigma_i = f(x_{i-1})b(x_{i-1})$ for $i = 1,...,\ell(|s|)$. Denoting by $f^i(x)$ the value of f iterated i times on x (i.e., $f^i(x) = f^{i-1}(f(x))$ and $f^0(x) = x$), we rewrite Eq. (8.7) as follows

$$G(s) \stackrel{\text{def}}{=} b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))\,. \tag{8.8}$$

The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a (random)[8] prefix of Z_k fails to predict the next bit of Z_k with probability non-negligibly higher than $1/2$. Specifically, we need to establish the following two results.

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

[8] For simplicity, we define unpredictability as referring to prefices of a random length (distributed uniformly in $\{0,...,|Z_k|-1\}$).

Clearly, pseudorandomness implies polynomial-time unpredictability, but here we actually need the other direction, which is less obvious. Still, using a hybrid argument, one can show that (next-bit) unpredictability implies indistinguishability from the uniform ensemble. For details see Exercise 8.9.

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

   Using the fact that f induces a permutation over $\{0,1\}^n$, observe that the $(j+1)$-bit long prefix of $G'(U_k)$ is distributed identically to $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$. Thus, an algorithm that predicts the $j+1^{\text{st}}$ bit of $G'(U_n)$ based on the j-bit long prefix of $G'(U_n)$ yields an algorithm that guesses $b(U_n)$ based on $f(U_n)$. For details see Exercise 8.11.

Needless to say, G is a pseudorandom generator if and only if G' is a pseudorandom generator (see Exercise 8.10). We mention that Eq. (8.8) is often referred to as the Blum-Micali Construction.[9]

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

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

To show that the existence of pseudorandom generators imply the existence of one-way functions, consider a pseudorandom generator G with stretch function $\ell(k) = 2k$. For $x, y \in \{0,1\}^k$, define $f(x,y) \stackrel{\text{def}}{=} G(x)$, and so f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_k)$ from U_{2k} by trying to invert and checking the result: Inverting f on its range distribution refers to the distribution $G(U_k)$, whereas the probability that U_{2k} has inverse under f is negligible.

The interesting direction of the proof of Theorem 8.11 is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_k)$ may not be pseudorandom, and so Construction 8.9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea underlying the construction is to hash $f(U_k)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions. (This is done after guaranteeing, that the logarithm of the probability mass of a value of $f(U_k)$ is

[9] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. Indeed, this construction originates in [37].

typically close to the entropy of $f(U_k)$.)[10] But "hashing $f(U_k)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $k' < k$. This foils the entire point of stretching the k-bit seed. Thus, a second idea underlying the construction is to compensate for the $k - k'$ loss by extracting these many bits from the seed U_k itself. This is done by hashing U_k, and the point is that the $(k - k')$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

8.3.6 Non-uniformly strong pseudorandom generators

Recall that we said that truly random sequences can be replaced by pseudorandom ones without affecting any efficient computation. The specific formulation of this assertion, presented in Proposition 8.3, refers to randomized algorithms that take a "primary input" and use a secondary "random input" in their computation. Proposition 8.3 asserts that it is infeasible to find a primary input for which the replacement of a truly random secondary input by a pseudorandom one affects the final output of the algorithm in a noticeable way. This, however, does not mean that such primary inputs do not exist (but rather that they are hard to find). Consequently, Proposition 8.3 falls short of yielding a (worst-case)[11] "derandomization" of a complexity class such as $\mathcal{BPP}$. To obtain such results, we need a stronger notion of pseudorandom generators, presented next. Specifically, we need pseudorandom generators that can fool all polynomial-size circuits (cf. §1.2.4.1), and not merely all probabilistic polynomial-time algorithms.[12]

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

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

---

[10] Specifically, given an arbitrary one-way function f', one first constructs f by taking a "direct product" of sufficiently many copies of f'. For example, for $x_1, ..., x_{k^{2/3}} \in \{0,1\}^{k^{1/3}}$, we let $f(x_1, ..., x_{k^{2/3}}) \stackrel{\text{def}}{=} f'(x_1), ..., f'(x_{k^{2/3}})$.

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

[12] Needless to say, strong pseudorandom generators in the sense of Definition 8.12 satisfy the basic definition of a pseudorandom generator (i.e., Definition 8.1); see Exercise 8.12.

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

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

Proof Sketch: Given any $L \in \mathcal{BPP}$ and any $\varepsilon > 0$, we let A denote the decision procedure for L and G denote a non-uniformly strong pseudorandom generator stretching n^ε-bit long seeds into poly(n)-long sequences (to be used by A as secondary input when processing a primary input of length n). We thus obtain an algorithm $A' = A_G$ (as in Construction 8.2). We note that A and A' may differ in their decision on at most finitely many inputs, because otherwise we can use these inputs (together with A) to derive a (non-uniform) family of polynomial-size circuits that distinguishes $G(U_{n^\varepsilon})$ and $U_{\mathrm{poly}(n)}$, contradicting the the hypothesis regarding G. (Specifically, in terms of the proof of Proposition 8.3, the finder F consists of a non-uniform family of polynomial-size circuits that print the "problematic" primary inputs that are hard-wired in them, and the corresponding distinguisher D' is thus also non-uniform.) Incorporating the finitely many "bad" inputs into A', we derive a probabilistic polynomial-time algorithm that decides L while using randomness complexity n^ε.

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

We comment that stronger results regarding derandomization of $\mathcal{BPP}$ are presented in Section 8.4.

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

8.3.7 Other variants and a conceptual discussion

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

8.3.7.1 Stronger notions

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

Fooling stronger distinguishers. One strengthening of Definition 8.1 amounts to explicitly quantifying the resources (and success gaps) of distinguishers. We chose to bound these quantities as a function of the length of the seed (i.e., k), rather than as a function of the length of the string that is being examined (i.e., $\ell(k)$). For a class of time bounds $\mathcal{T}$ (e.g., $\mathcal{T} = \{t(k) \stackrel{\text{def}}{=} 2^{c\sqrt{k}}\}_{c\in\mathbb{N}}$) and a class of noticeable functions (e.g., $\mathcal{F} = \{f(k) \stackrel{\text{def}}{=} 1/t(k) : t \in \mathcal{T}\}$), we say that a pseudorandom generator, G, is $(\mathcal{T},\mathcal{F})$-strong if for any probabilistic algorithm D having running-time bounded by a function in $\mathcal{T}$ (applied to k)[13], for any function f in $\mathcal{F}$, and for all sufficiently large k's

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

An analogous strengthening may be applied to the definition of one-way functions. Doing so reveals the weakness of the construction that underlies the proof of Theorem 8.11: It only implies that for some $\varepsilon > 0$ ($\varepsilon = 1/5$ will do), for any $\mathcal{T}$ and $\mathcal{F}$, the existence of "$(\mathcal{T},\mathcal{F})$-strong one-way functions" implies the existence of $(\mathcal{T}',\mathcal{F}')$-strong pseudorandom generators, where $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k^\varepsilon)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k)\cdot f(k^\varepsilon) : f \in \mathcal{F}\}$. What we *would like* to have is an analogous result with $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k)\cdot f(k) : f \in \mathcal{F}\}$.

Pseudorandom Functions. Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined in Appendix C.3.3) are even more powerful: They allow *efficient direct access* to a huge pseudorandom sequence, which is not even feasible to scan bit-by-bit. Put in other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). We mention that pseudorandom functions can be constructed from any pseudorandom generator (see Appendix C.3.3), and found many applications in cryptography (see Appendix C.3.3, C.5.2, and C.6.2). Pseudorandom functions have been used to derive negative results in computational learning theory [214] and in complexity theory (cf., Natural Proofs [176]).

8.3.7.2 Conceptual Discussion

Whoever does not value preoccupation with thoughts, can skip this chapter.

Robert Musil, The Man without Qualities, Chap. 28

We highlight several conceptual aspects of the foregoing computational approach to randomness. Some of these aspects are common to other instantiation of the general paradigm (esp., the one presented in Section 8.4).

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

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

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

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The general paradigm of pseudorandomness relies on the fact that putting computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Thus, the pivot of the entire approach is the computational difficulty of distinguishing pseudorandom distributions from truly random ones. Furthermore, many of the constructions of pseudorandom generators rely on either conjectures or facts regarding computational difficulty (i.e., that certain computations that are hard for certain classes). For example, one-way functions were used to construct general-purpose pseudorandom generators (i.e., those working in polynomial-time and fooling all polynomial-time observers). Analogously, as we shall see in Section 8.4.3, the fact that parity function is hard for polynomial-size constant-depth circuits can be used to generate (highly non-uniform) sequences that fool such circuits.

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions (cf. Sections 8.3.5 and 8.4.2). We wish to highlight the intuitive appeal of this connection.

8.4 Derandomization of time-complexity classes

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

8.4.1 Definition

Recall that in order to "derandomize" a probabilistic polynomial-time algorithm A, we first obtain a functionally equivalent algorithm A_G (as in Construction 8.2) that has (significantly) smaller randomness complexity. Algorithm A_G has to maintain A's input-output behavior on all (but finitely many) inputs. Thus, the set of the relevant distinguishers (considered in the proof of Theorem 8.13) is the set of all possible circuits obtained from A by hard-wiring each of the possible inputs. Such a circuit, denoted C_x, emulates the execution of algorithm A on input x, when using the circuit's input as the algorithm's internal coin tosses (i.e., $A(x,r) = C_x(r)$). Furthermore, the size of C_x is polynomial in the running-time of A on input x, and the length of the input to C_x is linear in the running-time of A (on input x).[14] For simplicity, let's say that the size of C_x is quadratic in the running-time of A on input x. Thus, the pseudorandom generator in use (i.e., G) needs to fool each of these possible circuits. Recalling that we may allow the generator to run in exponential time (in the length of its own input)[15], we arrive at the following definition.

[14] Indeed, if algorithm A is represented as a Turing machine then C_x has size that is at most quadratic (and in fact even almost-linear [167]) in the running-time of A on input x, which in turn means that C_x has size that is at most quadratic (or almost linear) in the length of its own input. We note that most sources use the fictitious convention by which the circuit size equals the length of its input, which can be justified by considering a suitably padded input.

[15] Actually, in Definition 8.14 we allow the generator to run in time $\mathrm{poly}(2^k \ell(k))$, rather than $\mathrm{poly}(2^k)$. This is done in order not to rule out trivially generators of super-exponential stretch (i.e., $\ell(k) = 2^{\omega(k)}$). However (see Exercise 8.13), the condition in Eq. (8.9) does not allow for super-exponential stretch, and so in retrospect the two formulations are equivalent (because $\mathrm{poly}(2^k \ell(k)) = \mathrm{poly}(2^k)$ for $\ell(k) = 2^{O(k)}$).

Definition 8.14 (pseudorandom generator for derandomizing BPTIME($\cdot$))[16]: *Let $\ell : \mathbb{N} \to \mathbb{N}$ be a 1-1 function. A* canonical derandomizer of stretch ℓ *is a deterministic algorithm G of time complexity upper-bounded by* $\text{poly}(2^k \cdot \ell(k))$ *such that for every circuit D_k of size $\ell(k)^2$ it holds that*

$$|\Pr[D_k(G(U_k)) = 1] - \Pr[D_k(U_{\ell(k)}) = 1]| < \frac{1}{6} \tag{8.9}$$

The circuits D_k are potential distinguishers, which are given inputs of length $\ell(k)$. When seeking to derandomize an algorithm A of time-complexity t, the aforementioned $\ell(k)$-bit long inputs represent possible random-inputs of A when invoked on a generic (primary) input of length $n = t^{-1}(\ell(k))$. That is, letting $D_k(r) = A(x, r)$ for some choice of $x \in \{0,1\}^n$, where $|r| = t(n) = \ell(k)$, and Eq. (8.9) implies that $A_G(x)$ maintains the majority vote of $A(x)$. The straightforward deterministic emulation of A_G takes time $2^k \cdot (\text{poly}(2^k \cdot \ell(k)) + t(n))$, which is upper-bounded by $\text{poly}(2^k \cdot \ell(k)) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. The following proposition is easy to establish.

Proposition 8.15 *If there exists a canonical derandomizer of stretch ℓ then, for every time-constructible $t : \mathbb{N} \to \mathbb{N}$, it holds that* $\text{BPTIME}(t) \subseteq \text{DTIME}(T)$*, where* $T(n) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$.

Proof Sketch: Just follow the proof of Theorem 8.13, noting that the adequate value of k (i.e., $k = \ell^{-1}(t(n))$) can be determined easily (e.g., by invoking $G(1^i)$ for $i = 1, ..., k$, using the fact that $\ell : \mathbb{N} \to \mathbb{N}$ is 1-1). Note that the complexity of the deterministic procedure is dominated by the 2^k invocations of $A_G(x, s) = A(x, G(s))$, where $s \in \{0,1\}^{\ell^{-1}(t(|x|))}$, and each of these invocations takes time $\text{poly}(2^k \cdot \ell(k)) + t(n) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. $\square$

The goal. In light of Proposition 8.15, we seek canonical derandomizers with stretch that is as big as possible. The stretch cannot be super-exponential (in fact $\ell(k) = O(2^k)$), because a circuit of size $O(2^k \cdot \ell(k))$ may violate Eq. (8.9) (see Exercise 8.13) whereas for $\ell(k) = \omega(2^k)$ it holds that $O(2^k \cdot \ell(k)) < \ell(k)^2$. Thus, our goal is to construct canonical derandomizer with stretch $\ell(k) = 2^{\Omega(k)}$. Such canonical derandomizers will allow for a "full derandomization of $\mathcal{BPP}$":

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

Proof: Using Proposition 8.15, we get $\text{BPTIME}(t) \subseteq \text{DTIME}(T)$, where $T(n) = \text{poly}(2^{\ell^{-1}(t(n))} \cdot t(n)) = \text{poly}(t(n))$. $\blacksquare$

[16] Fixing a model of computation, we denote by BPTIME(t) the class of decision problems that are solvable by a randomized algorithm of time complexity t that has two-sided error 1/3.

Reflections. We stress that a canonical derandomizer G was defined in a way that allows it to have time complexity t_G that is larger than the size of the circuits that it fools (i.e., $t_G(k) > \ell(k)^2$ is allowed). Furthermore, $t_G(k) > 2^k$ was also allowed. Thus, if indeed $t_G(k) = 2^{\Omega(k)}$ (as is the case in Section 8.4.2) then *$G(U_k)$ can be distinguished from $U_{\ell(k)}$ in time $2^k \cdot t_G(k) = \mathrm{poly}(t_G(k))$* (greater than $\ell(k)^2$), by trying all possible seeds. In contrast, for a general-purpose pseudorandom generator G (as discussed in Section 8.3) it holds that $t_G(k) = \mathrm{poly}(k)$, while *for every polynomial p it holds that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ in time $p(t_G(k))$*.

8.4.2 Construction

The fact that canonical derandomizers are allowed to be more complex than the corresponding distinguisher makes *some* of the techniques of Section 8.3 inapplicable in the current context. On the other hand, the techniques developed below are inapplicable to Section 8.3. Amazingly enough, the pseudorandomness (or rather the next-bit unpredictability) of the following generators hold even when the "observer" is given the seed (capitalizing on the fact that the observer's time-complexity does not allow running the generator).

As in Section 8.3.5, the construction surveyed below transforms computational difficulty into pseudorandomness, except that here both computational difficulty and pseudorandomness are of a somewhat different form than in Section 8.3.5. Specifically, here we use Boolean predicates that are computable in exponential-time but are T-inapproximated for some exponential function T (see Definition 7.9 in Section 7.2). That is, for constants $c, \varepsilon > 0$ and all but finitely many m, the (residual) predicate $f : \{0,1\}^m \to \{0,1\}$ is computable in time 2^{cm} but for any circuit C of size $2^{\varepsilon m}$ it holds that $\mathsf{Pr}[C(U_m) = f(U_m)] < \frac{1}{2} + 2^{-\varepsilon m}$. (Needless to say, $\varepsilon < c$.) Recall that such predicates exist under the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity (see Theorem 7.19 for an exact formulation). With these preliminaries, we turn to the construction of canonical derandomizers with exponential stretch.

Construction 8.17 (The Nisan-Wigderson Construction):[17] *Let $f : \{0,1\}^m \to \{0,1\}$ and $S_1, ..., S_\ell$ be a sequence of m-subsets of $\{1, ..., k\}$. Then, for $s \in \{0,1\}^k$, we let*

$$G(s) \stackrel{\text{def}}{=} f(s_{S_1}) \cdots f(s_{S_\ell}) \tag{8.10}$$

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

Letting k vary and $\ell, m : \mathbb{N} \to \mathbb{N}$ be functions of k, we wish G to be a canonical derandomizer and $\ell(k) = 2^{\Omega(k)}$. Obvious necessary conditions for this to happen include the requirement that the sets be distinct and hence $m(k) = \Omega(k)$; consequently, f must be computable in exponential-time. Furthermore, the sequence of

[17] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. This construction originates in [160, 163].

sets $S_1, ..., S_{\ell(k)}$ must be constructible in poly(2^k) time. Intuitively, it is desirable to use a set system with small pairwise intersections (because this restricts the overlap among the various inputs to which f is applied), and a function f that is strongly inapproximable (i.e., T-inapproximble for some exponential function T). Interestingly, these conditions are essentially sufficient.

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

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

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

For any $\gamma > 0$, a function S as in Condition 2 does exist (see Exercise 8.14), with some $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$. Combining such S with Theorems 7.19 and 8.18, we obtain a canonical derandomizer with exponential stretch based on the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.[18] Combining this with Theorem 8.16, we get the first item of the following theorem.

Theorem 8.19 (Derandomization of BPP, revisited):

1. *Suppose that there exists a set $S \in \mathcal{E}$ having almost-everywhere exponential circuit complexity* (i.e., there exists a constant $\varepsilon > 0$ such that, for all but finitely many m's, any circuit that correctly decides S on $\{0,1\}^m$ has size at least $2^{\varepsilon m}$). *Then, $\mathcal{BPP} = \mathcal{P}$.*
2. *Suppose that for every polynomial p there exists a set $S \in \mathcal{E}$ having circuit complexity that is almost-everywhere greater than p. Then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon > 0}$DTIME(t_ε), where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{n^\varepsilon}$.*

Part 2 is proved (in Exercise 8.18) by using a generalization of Theorem 8.18, which in turn is provided in Exercise 8.17. We note that Part 2 of Theorem 8.19 superseeds Theorem 8.13 (see Exercise 7.14). The two parts of Theorem 8.19 exhibit two extreme cases: Part 1 (often referred to as the "high end") assumes an extremely strong circuit lower-bound and yields "full derandomization" (i.e., $\mathcal{BPP} = \mathcal{P}$), whereas Part 2 (often referred to as the "low end") assumes an extremely weak

[18] Specifically, starting with a function having circuit complexity at least $\exp(\varepsilon_0 m)$, we apply Theorem 7.19 obtaining a T-inapproximble predicate for $T(m) = 2^{\varepsilon m}$, where the constant $\varepsilon \in (0, \varepsilon_0)$ depends on the constant ε_0. Next, we set $\gamma = \varepsilon/2$ and invoke Exercise 8.14, which determines $\alpha, \beta > 0$ such that $\ell(k) = 2^{\alpha k}$ and $m(k) = \beta k$. In fact, $\beta = \gamma/2$ and $\alpha = \gamma\beta/10$, hence $(2\alpha/\beta) + \gamma < \varepsilon$.

circuit lower-bound and yields weak but meaningfull derandomization. Intermediate results (relying on intermediate lower-bound assumptions) can be obtained analogous to Exercise 8.18, but tight trade-offs are obtained differently (cf., [210]).

Proof of Theorem 8.18: Using the time complexity bounds on f and S, it follows that G can be computed in exponential time. Out focus is on showing that $\{G(U_k)\}$ cannot be distinguished from $\{U_{\ell(k)}\}$ by circuits of size $\ell(k)^2$; that is, that G satisfies Eq. (8.9). In fact, we will prove that this holds for $G'(s) = s \cdot G(s)$; that is, G fools such circuits even if they are given the seed as auxiliary input. (Indeed, these circuits are smaller than the running time of G, and so they cannot just evaluate G on the given seed.)

We start with the intuition underlying the proof. As a warm-up suppose that the sets (i.e., $S(k,i)$'s) used in the construction are disjoint. In such a case (which is indeed impossible becuase $k < \ell(k) \cdot m(k)$), the pseudorandomness of $G(U_k)$ would follow easily from the inapproximablity of f, because in this case G consists of applying f to non-overlaping parts of the seed (see Exercise 8.15). In the actual construction being analyzed here, the sets (i.e., $S(k,i)$'s) are not disjoint but have relatively small pairwise intersection, which means that G applies f on parts of the seed that have relatively small overlap. Intuitively, such small overlaps guarantee that the values of f on the corresponding inputs are "computationally independent" (i.e., having the value of f at one input does not help to approximate the value of f at another input). This intuition will be backed by showing that the former values can be computed at a relatively small computational cost. With this intuition in mind, we now turn to the actual proof.

The proof that G' fools circuits of size $\ell(k)^2$ utilizes the relation between pseudorandomness and unpredictability. Specifically, as detailed in Exercise 8.16, any circuit that distinguishes $G'(U_k)$ from $U_{\ell(k)+k}$ with gap $1/6$, yields a next-bit predictor of similar size that succeeds in predicting the next bit with probability at least $\frac{1}{2} + \frac{1}{6\ell'(k)} > \frac{1}{2} + \frac{1}{7\ell(k)}$, where the $\ell'(k) = \ell(k) + k < (1 + o(1))\ell(k)$ factor is introduced by the hybrid technique (cf. Eq. (8.5)). Furthermore, given the non-uniform setting of the current proof, we may fix a bit location $i + 1$ for prediction, rather than analyzing the prediction at a random bit location. Indeed, $i \geq k$ must hold, because the first k bits of $G'(U_k)$ are uniformly distributed. In the rest of the proof, we transform such a predictor into a circuit that approximates f better than allowed by the hypothesis (regarding the inapproximability of f).

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

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

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

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

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

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

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

We obtain the desired circuit C, which depends on the index j and the string s' that are fixed as in the foregoing analysis. On input $x\in\{0,1\}^m$, the circuit computes the values $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$, invokes C'' on input x and these values, and outputs the answer as a guess for $f(x)$. That is,

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

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

8.4.3 Variants and a conceptual discussion

The Nisan–Wigderson Construction (Construction 8.17) is actually a general framework, which can be instantiated in various ways. We start this section by briefly reviewing some of these instantiations and end it with a conceptual discussion regarding derandomization.

Derandomization of constant-depth circuits. Using (Construction 8.17 with) a different setting of parameters and the parity function in the role of the inapproximable predicate (i.e., inapproximable by "small" constant-depth circuits), one can obtain pseudorandom generators that fool "small" constant-depth circuits (see [160]). The analysis of the modified construction proceeds very much like the proof of Theorem 8.18. One important observation is that incorporating the (straightforward) circuits that compute $f(\pi(x)_{S_i})$ into the distinguishing circuit only increases its depth by two levels. The resulting pseudorandom generator, which use a seed of polylogarithmic length (equiv., $\ell(k) = \exp(k^{1/O(1)})$), can be used for derandomizing $\mathcal{RAC}^0$ (i.e., random $\mathcal{AC}^0$), analogously to Theorem 8.16. In other words, we can *deterministically* approximate, in quasi-polynomial-time and up-to an additive error, the fraction of inputs that satisfy a given (constant-depth) circuit. Specifically, for any constant d, given a depth-d circuit C, one can approximate the fraction of the inputs that satisfy C (i.e., cause C to evaluate to 1) to within any *additive constant error*[19] in time $\exp(\text{poly}(\log |C|))$, where the polynomial depends on d. Providing a deterministic polynomial-time aproximation, even in the case $d = 2$ (i.e., CNF/DNF formulae) is an open problem.

Derandomization of probabilistic proof systems. A different (and more surprising) instantiation of Construction 8.17 utilizes predicates that are hard for small circuits having oracle access to $\mathcal{NP}$. The result is a pseudorandom generator robust against two-move public-coin interactive proofs (which are as powerful as constant-round interactive proofs). The key observation is that the above proof provides a black-box procedure for approximating the underlying predicate when given oracle access to a distinguisher (and this procedure in valid also in case the distinguisher is a non-deterministic machine). Thus, under suitably strong (and yet plausible) assumptions, constant-round interactive proofs collapse to $\mathcal{NP}$. We note that a stronger result, which deviates from the foregoing framework, has been subsequently obtained (cf. [155]).

An even more radical instantiation of Construction 8.17 was used to obtain explicit constructions of randomness extractors (see Section D.4). In addition to the foregoing observation, one also utilizes the fact that the generator itself uses the predicate as a black-box. Further discussion appears in Section D.4.

A conceptual discussion regarding derandomization

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

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

Taking a closer look at the proof of (the underlying) Theorem 8.16, we note that a randomized algorithm A of time complexity t is emulated by a deterministic algorithm A' of time complexity $t' = \mathrm{poly}(t)$. Further noting that $A' = A_G$ invokes A and the canonical derandomizer G for a number of times that must exceed t, we infer that $t' > t^2$ must hold. Thus, derandomization via (Part 1 of) Theorem 8.19 is not really for free.

More importantly, we note that derandomization is not possible in various distributed settings, when both parties may protect their conflicting interests by employing randomization. Notable examples include most cryptographic primitives (e.g., encryption) as well as most types of probabilistic proof systems (e.g., PCP). For further discussion see Chapter 9 and Appendix C. Additional settings where randomness makes a difference (either between impossibility and possibility or between prohibited and affordable cost) include distributed computing (see [15]), communication complexity (see [138]), parallel architectures (see [141]), sampling (see Section D.3), and property testing (see Section 10.1.2).

8.5 Space Pseudorandom Generators

In the previous two sections we have considered generators the output of which is indistinguishable by any efficient procedures. The latter were modeled by time-bounded computations; specifically, polynomial-time computations. A finer characterization of time-bounded computations is obtained by considering their space-complexity (i.e., restricting the space-complexity of time-bounded computations). In contrast to the definitions of pseudorandom generators that were considered in Sections 8.3 and 8.4, the existence of pseudorandom generators that fool space-bounded distinguishers can be established without relying on computational assumptions.

8.5.1 Definitional issues

Unfortunately, natural notions of space-bounded computations are quite subtle, especially when non-determinism or randomization are concerned (see Sections 5.3 and 6.1.4). Two major issues are *time bounds* and *access to the random tape*.

1. Time bound: The question is whether or not one restricts the space-bounded machines to run in time-complexity that is at most exponential in the space-complexity.[20] Recall that such an upper-bound follows automatically in the deterministic case, and can be assumed without loss of generality in the non-deterministic case *but not in the randomized case*.

   Indeed, as in Section 6.1.4, we do postulate the aforementioned time-bound.

2. Access to the random tape: The question is whether whether the space-bounded machine has one-way or two-way access to the randomness tape.

[20] Alternatively, one can ask whether these machines must always halt or only halt with probability approaching 1. It can be shown that the only way to ensure "absolute halting" is to have time-complexity that is at most exponential in the space-complexity.

(Allowing two-way access means that the randomness is recorded for free; that is, without being accounted for in the space-bound; see discussions in Sections 5.3 and 6.1.4.) Recall that one-way access to the randomness tape corresponds to the natural model of on-line randomized machine (which determines its moves based on its internal coin tosses).

Again, as in Section 6.1.4, we consider one-way access.[21]

In accordance with the resulting definition of randomized space-bounded computation, we consider space-bounded distinguishers that have a one-way access to the input sequence that they examine. Since all known constructions remain valid also when these distinguishers are non-uniform (and since non-uniform distinguishers arise anyhow in derandomization), we use this stronger notion here.[22]

In the context of non-uniform algorithms that have one-way access to their input, we may assume, without loss of generality, that the running-time of such algorithms equals the length of their input, denoted $\ell = \ell(k)$. Thus, we define a non-uniform machine of space $s:\mathbb{N}\to\mathbb{N}$ as a family, $\{D_k\}_{k\in\mathbb{N}}$, of directed layered graphs such that D_k has at most $2^{s(k)}$ vertices at each layer, and labeled directed edges from each layer to the next layer.[23] Each vertex has two (possibly parallel) outgoing directed edges, one labeled 0 and the other labeled 1, and there is a single vertex in the first layer of D_k. The result of the computation of such a machine, on an input of adequate length (i.e., length ℓ where D_k has $\ell+1$ layers), is defined as the vertex (in last layer) reached when following the sequence of edges that are labelled by the corresponding bits of the input. That is, on input $x = x_1 \cdots x_\ell$, for $i = 1, ..., \ell$, we move from the vertex reached in the i^{th} layer by using the outgoing edge labelled x_i (thus reaching a vertex in the $i+1^{\text{st}}$ layer). Using a fixed partition of the vertices of the last layer, this defines a natural notion of decision (by D_k); that is, we write $D_k(x) = 1$ if on input x machine D_k reached a vertex that belongs to the first part of the aforementioned partition.

Definition 8.20 (Indistinguishability by space-bounded machines):

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

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

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

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

[22] We note that these non-uniform space-bounded distinguishers correspond to branching programs of width that is exponential in the space-bound. Furthermore, these branching programs read their input in a fixed predetermined order (which is determined by the designer of the generator).

[23] Note that the space bound of the machine is stated in terms of a parameter k, rather than in terms of the length of its input. In the sequel this parameter will be set to the length of a seed to a pseudorandom generator. We warn that our presentation here is indeed non-standard for this area. To compensate for this, we will also state the consequences in the standard format.

- *A probability ensemble, $\{X_k\}_{k\in\mathbb{N}}$, is called (s,ε)-*pseudorandom *if for any (non-uniform) $s(\cdot)$-space-bounded machine, the distinguishability-gap of the machine between $\{X_k\}_{k\in\mathbb{N}}$ and a uniform ensemble (of length $|X_k|$) is at most $\varepsilon(\cdot)$.*
- *A deterministic algorithm G of stretch function ℓ is called a (s,ε)-*pseudorandom generator *if the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is (s,ε)-pseudorandom.*

8.5.2 Two constructions

In contrast to the case of pseudorandom generators that fool time-bounded distinguishers, pseudorandom generators that fool space-bounded distinguishers can be established without relying on any computational assumption. The following two constructions exhibit two extreme cases of a general trade-off between the length of the seed and the stretch function of the generator.[24] We start with an attempt to maximize the stretch.

Theorem 8.21 (exponential stretch with quadratic length seed): *For every space constructible function $s:\mathbb{N}\to\mathbb{N}$, there exists a $(s,2^{-s})$-pseudorandom generator of stretch function $\ell(k)=2^{k/O(s(k))}$. Furthernore, the generator works in space linear in the length of the seed, and in time linear in the stretch function.*

In other words, we have a generator that takes a random seed of length $k=O(t\cdot m)$ and produce sequences of length 2^t that look random to any m-space-bounded machine. In particular, using a random seed of length $k=O(m^2)$, one can produce sequences of length 2^m that look random to any m-space bounded machine. Thus, *one may replace random sequences used by any space-bounded computation, by sequences that are efficiently generated from random seeds of length quadratic in the space bound.* The common instantiation is for log-space machines. In §8.5.2.2, we apply Theorem 8.21 (and its underlying ideas) for the derandomization of space complexity classes such as $\mathcal{BPL}$ (i.e., the log-space analogue of $\mathcal{BPP}$).

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

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

In other words, we have a generator that takes a random seed of length $k=O(m)$ and produce sequences of length $\mathrm{poly}(m)$ that look random to any m-space-bounded machine. Thus, one may *convert any randomized computation utilizing*

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

polynomial-time and linear-space into a functionally equivalent randomized computation of similar time and space complexities that uses only a linear number of coin tosses.

8.5.2.1 Overviews of the proofs of Theorems 8.21 and 8.22

In both cases, we describe the construction by starting with an adequate distinguisher and showing how the input distribution it examines can be modified (from the uniform one into a pseudorandom one) without the distinguisher noticing the difference.

Overview of the proof of Theorem 8.21.[25] Theorem 8.21 is proven by using the "mixing property" of pairwise independent hash functions (see Section D.2). A family of functions H_n which map $\{0,1\}^n$ to itself is called *mixing* if for every pair of subsets $A, B \subseteq \{0,1\}^n$ for all but very few (i.e., $\exp(-\Omega(n))$ fraction) of the functions $h \in H_n$,

$$\mathsf{Pr}[U_n \in A \wedge h(U_n) \in B] \approx \frac{|A|}{2^n} \cdot \frac{|B|}{2^n} \tag{8.11}$$

where the approximation is up to an additive term of $\exp(-\Omega(n))$. (See the generalization of Lemma D.4.)

Given a $s(k)$-space distinguisher D_k as in Definition 8.20, we set $n \stackrel{\text{def}}{=} \Theta(s(k))$ and $\ell' \stackrel{\text{def}}{=} \ell(k)/n < 2^{s(k)}$, and consider an auxiliary "distinguisher" D'_k that is a directed layered graph with ℓ' layers and $2^{s(k)}$ vertices in each layer. In D'_k, each vertex has directed edges going to each vertex of the next layer and these edges are *labeled with* (possibly empty) *subsets of* $\{0,1\}^n$ such that these subsets form a partition of $\{0,1\}^n$. The graph D'_k simulates D_k in the obvious manner; that is, the computation of D'_k on an input of length $\ell(k) = \ell' \cdot n$ is defined by breaking the input into consecutive blocks of length n and following the path of edges that are labeled by the subsets containing the corresponding block. Now, for each pair of neighboring vertices, u and v (in layers i and $i+1$, respectively), consider the label, $L_{u,v} \subseteq \{0,1\}^n$, of the edge going from u to v. Similarly, for a vertex w at layer $i+2$, we consider the label $L'_{v,w}$ of the edge from v to w. By Eq. (8.11), for all but very few few of $h \in H_n$,

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

where "very few" and $\approx$ are as in Eq. (8.11). Thus, replacing the coins in the second block (i.e., used in transitions from layer $i+1$ to layer $i+2$) with the value of h applied to the outcomes of the coins used in the first block (i.e., in transitions from layer i to $i+1$), approximately maintains the probability that D'_k moves from u to w via v. The same (with "few" being $2^{3s(k)} \cdot \ell'$ times larger here)[26] holds for every triple of vertices in any three layers as in the foregoing discussion. The point is that we can use the same h in all these approximations. Thus, at the cost

[25] A detailed proof appears in [161].

[26] Note that "very few" means an $\exp(-\Omega(n))$ fraction and that $n = \Omega(s(k))$ and $\ell' < \exp(s(k))$.

of extra $|h|$ random bits, we can reduce the number of true random coins used in transitions on D'_k by a factor of 2, without significantly affecting the final decision of D'_k. In other words, at the cost of extra $|h|$ random bits, we can effectively contract the distinguisher to half its length. That is, fixing a good h (i.e., one that provides a good approximation to all $2^{3s(k)} \cdot \ell'$ relevant pairs of sets), we can replace the 2-edge paths in D'_k by edges in a new distinguisher D''_k such that r is in the set that labels the edge u–w in D''_k if and only if, for some v, the string r is in the label of the edge u–v in D'_k and $h(r)$ is in the label of the edge v–w (also in D'_k).

Repeating the process for a logarithmic (in D'_k's length) number of times we obtain a distinguisher that only examines n bits, at which point we stop. In total, we have used $\log_2(\ell(k)/O(s(k))) < \log_2 \ell(k)$ random hash functions, which means that we can generate a sequence that fools the original D_k using a seed of length $n + \log_2 \ell(k) \cdot \log_2 |H_n|$ (see Exercise 8.19). Using $n = \Theta(s(k))$ and an adequate family H_n (e.g., Construction D.3) yields the claimed seed length of $O(s(k) \cdot \log_2 \ell(k)) = k$. □

Overview of the proof of Theorem 8.22.[27] Theorem 8.22 is proven by using a suitable randomness extractor (as defined in §D.4.1.1), which is indeed a much more powerful tool than hashing functions. The basic idea is that when D_k is at some distant layer, say at layer t, it typically "knows" little about the random choices that led it there. That is, D_k has only $s(k)$ bits of memory, which leaves out $t - s(k)$ bits of "uncertainty" (or randomness) regarding the previous moves. Thus, much of the randomness that led D_k to its current state may be "re-used" (or "recycled"). To re-use these bits we need to extract *almost* uniform distribution on strings of sufficient length out of the aforementioned distribution over $\{0,1\}^t$ that has entropy[28] at least $t - s(k)$. Furthermore, such an extraction requires some – yet relatively few – truly random bits. In particular, using $k' = \Omega(\log t)$ bits towards this end, the extracted bits are $\exp(-\Omega(k'))$ away from uniform.

One important point is how to use the foregoing argument repeatedly. Towards this end, we break the k-bit long seed into two parts, denoted $r' \in \{0,1\}^{k/2}$ and $(r_1, ..., r_{3\sqrt{k}})$, where $|r_i| = \sqrt{k}/6$, and set $n = k/3$. Intuitively, r' will be used for determining the first n steps, and it will be re-used (or recycled) together with r_i for determining the steps $i \cdot n + 1$ through $(i+1) \cdot n$. Looking at layer $i \cdot n$, we consider the information regarding r' that is known to D_k (at layer $i \cdot n$). Typically, the conditional distribution of r', given that we reached a specific vertex at layer $i \cdot n$ has (min-)entropy greater than $0.99 \cdot (t - s(k))$. Using r_i (as a seed of an extractor applied to r'), we can extract $0.9 \cdot ((k/2) - s(k) - o(k)) > k/3 = n$ bits that are almost-random with respect to D_k, and use these bits for determining the next n steps. Hence, using k random bits we are produce a sequence[29] of length

[27] A detailed proof appears in [164].

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

[29] Specifically, using an extractor of the form $\mathrm{Ext} : \{0,1\}^{\sqrt{k}/6} \times \{0,1\}^{k/2} \to \{0,1\}^{k/3}$, we map

$(1+3\sqrt{k}) \cdot n > k^{3/2}$ that fools machines of space bound, say, $s(k) = k/10$. That is, we obtained a $(s, 2^{-\Omega(\sqrt{s})})$-pseudorandom generator of stretch function $\ell(k) = k^{3/2}$.

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

1. On input $s \in \{0,1\}^k$, obtain $G_1(s)$, and parse it into consecuetive blocks, each of length $m = s_1(k)/O(1)$, denoted $r_1, ..., r_t$, where $t = \ell_1(k)/m$.

2. Output the $t \cdot \ell_2(m)$-bit long sequence $G_2(r_1) \cdots G_2(r_t)$.

Note that $|G(s)| = \ell_1(k) \cdot \ell_2(m)/m$, which for $s_1(k) = \Theta(k)$ yields $|G(s)| = \ell_1(k) \cdot \ell_2(\Omega(k))/O(k)$. We claim that G is a (s, ε)-pseudorandom generator, for $s(k) = \min(s_1(k)/2, s_2(\Omega(s_1(k)))$ and $\varepsilon(k) = \varepsilon_1(k) + \ell_1(k) \cdot \varepsilon_2(\Omega(s_1(k))$. The proof uses a hybrid argument, which focuses on the intermediate distribution $G_2(U_m^{(1)}) \cdots G_2(U_m^{(t)})$. The key claim is that the intermediate distribution is $(s_1/2, \varepsilon_1)$-indistinguishable from $G(U_k)$, and it is proven by converting a potential distinguisher into a distinguisher of $U_m^{(1)} \cdots U_m^{(t)}$ and $G_1(U_k)$ by invoking G_2 on the corresponding m-bit long blocks (of the $\ell(k)$-bit long input). For this reason, it crucial that G_2 can be evaluate on m-bit long strings using space at most $s_1(k)/2$, which is guaranteed by our setting of $m = s_1(k)/O(1)$ and the hypothesis that G_2 works in linear space. $\square$

8.5.2.2 Derandomization of space-complexity classes

As a direct application of Theorem 8.21, we obtain that $\mathcal{BPL} \subseteq \textsc{Dspace}(\log^2)$, where $\mathcal{BPL}$ denotes the log-space analogue of $\mathcal{BPP}$ (see Definition 6.9). (Recall that $\mathcal{NL} \subseteq \textsc{Dspace}(\log^2)$, but it is not known whether or not $\mathcal{BPL} \subseteq \mathcal{NL}$.)[30] A stronger derandomization result can be obtained by a finer analysis of the proof of Theorem 8.21.

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

Thus, $\mathcal{BPL} \supseteq \mathcal{RL}$ is placed in a class not known to contain $\mathcal{NL}$. Another such result was subsequently obtained in [183]: Randomized log-space can be simulated in deterministic space $o(\log^2)$; specifically, in space $\log^{3/2}$. We mention that the archetypical problem of $\mathcal{RL}$ has been recently proved to be in $\mathcal{L}$ (see Section 5.2).

Overview of the proof of Theorem 8.23.[31] Looking at the proof of Theorem 8.21, we note that the question of whether or not a specific hash function

the seed $(r', r_1, ..., r_{3\sqrt{k}})$ to the output sequence $(r', \mathrm{Ext}(r_1, r'), ..., \mathrm{Ext}(r_{3\sqrt{k}}, r'))$.

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

[31]A detailed proof appears in [162].

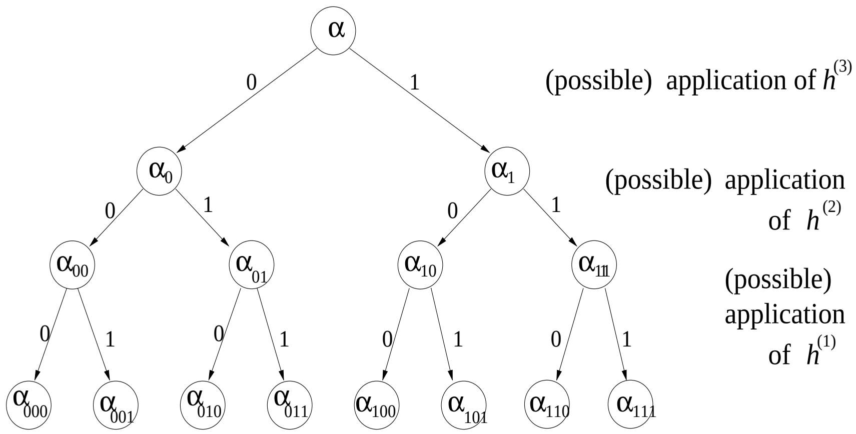
$h \in H_n$ is good for a specific D'_k can be determined in space that is linear in $n = |h|/2$ and logarithmic in the size of D'_k. Indeed, the time complexity of this decision procedure is exponential in its space complexity. It follows that we can find a good $h \in H_n$, for a given D'_k, within these complexities (by scanning through all possible $h \in H_n$). Once a good h is found, we can also construct the corresponding graph D''_k (in which edges represent 2-edge paths in D'_k), again within the same complexity. Actually, it will be more instructive to note that we can determine a step (i.e., an edge-traversal) in D''_k by making two steps (edge-traversals) in D'_k.

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

$$\alpha, h^{(1)}(\alpha), h^{(2)}(\alpha), h^{(1)}(h^{(2)}(\alpha)), ..., h^{(1)}(h^{(2)}(\cdots h^{(i)}(\alpha)\cdots)).$$

(See Figure 8.3.) Thus, for $n = \Theta(\log|D'_k|)$, given D'_k and a pair (u, v) of source and sink in D'_k (which reside in the first and last layer, respectively), we can (deterministically) approximate the probability that a random walk starting at u reaches v in $O(\log|D'_k|)^2$-space and $\text{poly}(|D'_k|)$-time. The approximation can be made accurate up to a factor of $1 \pm (1/\text{poly}(|D'_k|))$. We conclude the proof by recalling the connection between such an approximation and the derandomization of $\mathcal{BPL}$.

The computation of a log-space probabilistic machine M, on input x, can be represented by a directed layer graph $G_{M,x}$ of size $\text{poly}(|x|)$. Specifically, the probability that M accepts x equals the probability that a random walk starting at the single vertex of the first layer of $G_{M,x}$ reaches some vertex in the last layer that represents an accepting configuration. Setting $k = \Theta(\log|x|)$ and $n = \Theta(k)$, the graph $G_{M,x}$ coincides with the graph D_k referred to at the beginning of the proof of Theorem 8.21, and D'_k is obtained from D_k by an "n-layer contraction" (see ibid.). Combining this with the foregoing analysis, we conclude that the probability that M accepts x can be deterministically approximated in $O(\log|x|)^2$-space and $\text{poly}(|x|)$-time. The theorem follows. $\square$



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

Figure 8.3: Derandomization of $\mathcal{BPL}$ – the generator for $i = 3$.

8.6 Special Purpose Generators

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

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

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

8.6.1 Pairwise-Independence Generators

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

A t-wise independence generator of block-size $b : \mathbb{N} \to \mathbb{N}$ (and stretch function ℓ) is an efficient deterministic algorithm (e.g., one that works in time polynomial in the output length) that expands a k-bit long random seed into a sequence of $\ell(k)/b(k)$ strings, each of length $b(k)$, such that any t blocks are uniformly and independently distributed in $\{0,1\}^{t \cdot b(k)}$. In case $t = 2$, we call the generator pairwise independent. We note that this condition holds even if the inspected t blocks are selected adaptively (see Exercise 8.20)

8.6.1.1 Constructions

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

Proposition 8.24 (t-wise independence generator):[32] *Let t be a fixed integer and $b, \ell, \ell' : \mathbb{N} \to \mathbb{N}$ such that $b(k) = k/t$, $\ell'(k) = \ell(k)/b(k) > t$ and $\ell'(k) \le 2^{b(k)}$. Let $\alpha_1, ..., \alpha_{\ell'(k)}$ be distinct elements of this field. For $s_0, s_1, ..., s_{t-1} \in \{0,1\}^{b(k)}$, let*

$$G(s_0, s_1, ..., s_{t-1}) \stackrel{\text{def}}{=} \left(\sum_{j=0}^{t-1} s_j \alpha_1^j \,,\, \sum_{j=0}^{t-1} s_j \alpha_2^j \,, ..., \sum_{j=0}^{t-1} s_j \alpha_{\ell'(k)}^j \right) \tag{8.12}$$

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

That is, given a seed that consists of t elements of $\mathrm{GF}(2^{b(k)})$, the generator outputs a sequence of $\ell'(k)$ such elements. To make the above generator totally explicit, we need an explicit representation of $\mathrm{GF}(2^{b(k)})$, which requires an irreducible polynomial of degree $b(k)$ over $\mathrm{GF}(2)$. For specific values of $b(k)$, a good representation does exist: Specifically, for $d \stackrel{\text{def}}{=} b(k) = 2 \cdot 3^e$ (with e being an integer), the polynomial $x^d + x^{d/2} + 1$ is irreducible over $\mathrm{GF}(2)$. The proof of Proposition 8.24 is left as an exercise (see Exercise 8.21).

[32] The common parameterization of t-wise independence generator is as follows. Given parameters b and $\ell' \le 2^b$, and a uniformly distributed seed of length $t \cdot b$, one efficiently and deterministically generates a random sequence of ℓ' strings, each of length b, that are t-wise independent.

An alternative construction for the case of $t = 2$ is obtained by using (random) affine transformations (as possible seeds). In fact, better performance (i.e., shorter seed length) is obtained by using affine transformations defined by Toeplitz matrices. A Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s).

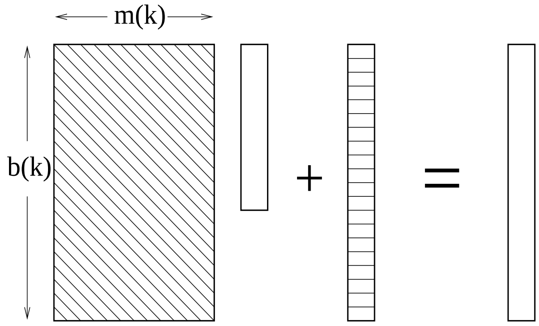


Figure 8.4: An affine transformation defined by a Toeplitz matrix.

Proposition 8.25 (Alternative pairwise independence generator, see Figure 8.4):[33] *Let $b, \ell, \ell', m : \mathbb{N} \to \mathbb{N}$ such that $\ell'(k) = \ell(k)/b(k)$ and $m(k) = \lceil \log_2 \ell'(k) \rceil = k - 2b(k) + 1$. Associate $\{0,1\}^n$ with the n-dimensional vector space over* GF(2)*, and let $v_1, ..., v_{\ell'(k)}$ be distinct vectors in the $m(k)$-dimensional vector space over* GF(2)*. For $s \in \{0,1\}^{b(k)+m(k)-1}$ and $r \in \{0,1\}^{b(k)}$, let*

$$G(s, r) \stackrel{\text{def}}{=} (T_s v_1 + r \, , \, T_s v_2 + r \, , ..., \, T_s v_{\ell'(k)} + r) \tag{8.13}$$

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

That is, given a seed that represents an affine transformation defined by an $b(k)$-by-$m(k)$ Toeplitz matrix, the generator outputs a sequence of $\ell'(k) \leq 2^{m(k)}$ strings, each of length $b(k)$. Note that $m(k) = k - 2b(k) + 1$, and that the stretching property requires $\ell'(k) > k/b(k)$. The proof of Proposition 8.25 is left as an exercise (see Exercise 8.22).

A stronger notion of efficient generator. We note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed, $s \in \{0,1\}^k$, and a block location $i \in [\ell'(k)]$ (in bianry), outputs the i^{th} block of the corresponding output (i.e., the i^{th} block of $G(s)$).

[33]The common parameterization of this pairwise independence generator is as follows. Given parameters b and ℓ', and a uniformly chosen seed of length $2b + \lceil \log_2 \ell' \rceil - 1$, one efficiently and deterministically generates a random sequence of ℓ' strings, each of length b, that are pairwise independent.

8.6.1.2 Applications

Pairwise independence generators do suffice for a variety of applications (cf., [220, 149]). In particular, we mention the application to sampling discussed in Section D.3, and the celebrated derandomization of the Maximal Independent Set algorithm. The latter uses the fact that the analysis of the "target" randomized algorithm only relies on the hypothesis that some objects are selected in pairwise independent manner. Thus, such weak generators do suffice to fool distinguishers that are derived from some natural and interesting algorithms.

Referring to Eq. (8.12), we remark that for constant $t \geq 2$, the cost of derandomization (i.e., going over all 2^k possible seeds) is exponential in the block-size (because $b(k) = \Omega(k)$), which in turn also bounds the number of blocks (because $\ell'(k) \leq 2^{b(k)}$). Note that if a larger number of blocks is needed, we can artificially increase the block-length in order to allow for it (i.e., allow $\ell'(k) = 2^{b(k)} = \exp(k/t)$), and in this case the cost of derandomization will be polynomial in the number of blocks. Thus, *whenever the analysis of a randomized algorithm can be based on a constant amount of independence* between (feasibly-many) random choices, each made inside a feasible domain, *a feasible derandomization is possible.*[34] On the other hand, the relationship $\ell(k) = \exp(k/t)$ is the best possible; that is, one cannot produce from a seed of length k an $\exp(k/O(1))$-long sequence of non-constantly independent random bits. In other words, t-wise independent generators of (any block-length and) stretch ℓ require a seed of length $\Omega(t \cdot \log \ell)$. In the next subsection we will see that meaningful approximations may be obtained with much shorter seeds.

8.6.2 Small-Bias Generators

Trying to go beyond constant-independence in derandomizations (while using seeds of length that is logarithmic in the length of the pseudorandom sequence) was the original motivation (and remain an important application) of the notion of small-bias generators. Still, small-bias generators are interesting for their own sake, and in particular they fool "global tests" that look at the entire output sequence and not merely at a fixed number of positions in it (as the limited independence generators). Specifically, small-bias generators generate a sequence of bits that fools any linear test (i.e., a test that computes a fixed linear combination of the bits).

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

$$2 \cdot \left| \mathsf{Pr}[\oplus_{i \in S} \zeta_i = 1] - \frac{1}{2} \right| = |\mathsf{Pr}[\oplus_{i \in S} \zeta_i = 1] - \mathsf{Pr}[\oplus_{i \in S} \zeta_i = 0]|. \tag{8.14}$$

[34] We stress that it is important to have the cost of derandomization be polynomial in the length of the produced pseudorandom sequence, because the latter is typically polynomially-related to the length of the input to the algorithm we wish to derandomize.

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

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

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

8.6.2.1 Constructions

Efficient small-bias generators with exponential stretch and exponentially vanishing bias are know.

Theorem 8.26 (small-bias generators):[35] *For some universal constant $c>0$, let $\ell:\mathbb{N}\to\mathbb{N}$ and $\varepsilon:\mathbb{N}\to[0,1]$ such that $\ell(k)\le\varepsilon(k)\cdot\exp(k/c)$. Then, there exists an ε-bias generator with stretch function ℓ operating in time polynomial in the length of its output.*

Three simple constructions of small-bias generators that satisfy Theorem 8.26 are known (see [9]). One of these constructions is based on Linear Feedback Shift Registers. Loosely speaking, the first half of the seed, denoted $f_0f_1\cdots f_{(k/2)-1}$, is interpreted as a (non-degenerate) feedback rule[36], the other half, denoted $s_0s_1\cdots s_{(k/2)-1}$, is interpreted as "the start sequence", and the output sequence, denoted $r_0r_1\cdots r_{\ell(k)-1}$, is obtained by setting $r_i=s_i$ for $i<k/2$ and $r_i=\sum_{j=0}^{(k/2)-1} f_j\cdot r_{i-(k/2)+j}$ for $i\ge k/2$. (See Figure 8.5 and Exercise 8.26.)

As in Section 8.6.1.1, we note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed and a bit location $i\in[\ell(k)]$ (in bianry), outputs the $i^{\rm th}$ bit of the corresponding output.

8.6.2.2 Applications

An archetypical application of small-bias generators is for generating random checks for fast string equality. The key obsererevation is that checking whether or not $x=y$ is probabilistically reducible to checking whether the inner product modulo 2 of x and r equals the inner product modulo 2 of y and r, where r is generated by a small-bias generator. One advantage of this reduction is that only few bits (i.e., the seed of the generator and the result of the inner product) needs to be "communicated

[35] Here the common parameterization differs from ours merely in the point of view: Rather than saying that the functions ℓ and ε should satisfy $\ell(k)\le\varepsilon(k)\cdot\exp(k/c)$, one says that given the desired parameters ℓ and ε the seed length k is set to $O(\log(\ell/\varepsilon))$. We also comment that using [9] the constant in the O-notation is merely 2 (i.e., $k\approx 2\log_2(\ell/\varepsilon)$), whereas using [158] $k\approx\log_2\ell+4\log_2(1/\varepsilon)$.

[36] That is, $f_0=1$ and $f(z)\stackrel{\rm def}{=}z^{k/2}+\sum_{j=0}^{(k/2)-1} f_j\cdot z^j$ is an irreducible polynomial over GF(2).

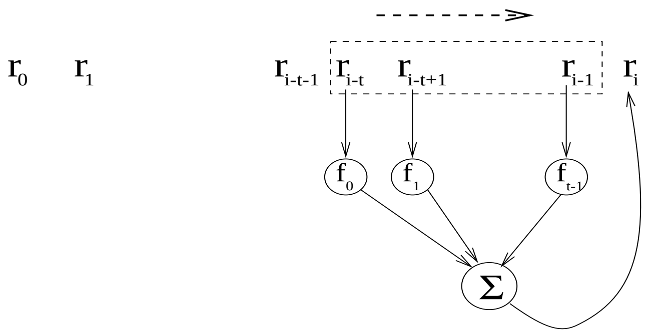


Figure 8.5: The LFSR small-bias generator (for $t = k/2$).

between x and y" (see Exercise 8.24). A related advantage (i.e., low randomness complexity) underlies the application of small-bias generators in §9.3.2.2.

Small-bias generators have been used in a variety of areas (e.g., inapproximation, structural complexity, and applied cryptography; see references in [86, Sec 3.6.2]). In addition, they seem an important tool in the design of various types of "pseudorandom" objects; see next.

Approximate independence generators. As hinted at the beginning of this section, small-bias is related to approximate limited independence.[37] Actually, even a restricted type of ε-bias (in which only subsets of size $t(k)$ are required to have bias upper-bounded by ε) implies that any $t(k)$ bits in the said sequence are $2^{t(k)/2} \cdot \varepsilon(k)$-close to $U_{t(k)}$, where here we refer to the variation distance (i.e., Norm-1 distance) between the two distributions. (The max-norm of the difference is bounded by $\varepsilon(k)$.)[38] Combining Theorem 8.26 and the foregoing upper-bound, and relying on the linearity of the construction in Proposition 8.24, we obtain generators with $\exp(k)$ stretch that are approximately $t(k)$-independent, for some non-constant $t(k)$; see Exercise 8.29. Specifically, for $k = O(t(k) + \log(1/\varepsilon(k)) + \log\log \ell(k))$ (equiv., for $\ell(k) = 2^{2^{k/O(1)}}$, $t(k) = k/O(1)$, and $\varepsilon(k) = 2^{-k/O(1)}$), one may obtain generators with stretch function ℓ, producing bit sequences in which any $t(k)$ positions are at most $\varepsilon(k)$-away from uniform (in variation distance). In the corresponding result for the max-norm distance, it suffices to have $k = O(\log(t(k)/\varepsilon(k) + \log\log \ell(k))$. Thus, whenever the analysis of a randomized algorithm can be based on a logarithmic amount of (almost) independence between feasibly-many binary random choices, a feasible derandomization is possible (by using an adequate generator of logarithmic seed length).

Extensions to non-binary choices were considered in various works (see references in [86, Sec 3.6.2]). Some of these works also consider the related problem of constructing small "discrepancy sets" for geometric and combinatorial rectangles.

[37] We warn that, unlike in the case of perfect independence, here we refer only to the distribution on fixed bit locations. See Exercise 8.23 for further discussion.

[38] Both bounds are derived from the Norm2 bound on the difference vector (i.e., the difference between the two probability vectors). For details, see Exercise 8.25.

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

8.6.3 Random Walks on Expanders

In this section we review generators that produce a sequence of values by taking a random walk on a large graph (called an expander) having small degree but good "mixing" properties. Thus, producing a sequence of length ℓ over 2^b values, requires a random seed of length $b + (\ell - 1) \cdot \log_2 d$, where d is the degree of the said graph (of 2^b vertices). This should be compared against the randomness needed for generating a sequence of ℓ independent samples from $\{0,1\}^b$ (or taking a random walk on a clique of size 2^b). It will turn out that the pseudorandom sequence (generated by the said random walk on an expander) *behaves analogously to a truly random sequence with respect to hitting any fixed subset of* $\{0,1\}^b$. Let us start by defining this property, or rather define the hitting problem.

Definition 8.27 (the hitting problem): *A distribution on sequences over* $\{0,1\}^b$ *is* (ε, δ)-hitting *if for any* (target) *set* $T \subseteq \{0,1\}^b$ *of cardinality at least* $\varepsilon \cdot 2^b$*, with probability at least* $1 - \delta$*, at least one of the elements of a sequence drawn from this distribution hits* T.

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

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

We are interested in explicit constructions of such graphs, by which we mean that there exists a polynomial-time algorithm that on input N (in binary), a vertex $v \in G_N$ and an index $i \in \{1, ..., d\}$, returns the i^{th} neighbor of v. (We also require that the set S for which G_N's exist is sufficiently "tractable" – say that given any $n \in \mathbb{N}$ one may efficiently find an $s \in S$ such that $n \leq s < 2n$.) Several explicit constructions of expanders are known (see Section E.2.2). Below, we rely on the fact that for every $\varepsilon > 0$, there exist d and $\lambda < \varepsilon \cdot d$ such that there exists an explicit construction of a (d,λ)-expander over $\{2^b : b \in \mathbb{N}\}$.[39] The relevant (to us) fact about expanders is stated next.

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

$$\rho \cdot \left(\rho + (1-\rho)\cdot\frac{\lambda}{d}\right)^{\ell-1} \tag{8.15}$$

Thus, a random walk on an expander is "pseudorandom" with respect to the hitting property (i.e., when we consider hitting the set $V \setminus W$); that is, a set of density $\varepsilon = 1-\rho$ is hit with probability $1-\delta$, where $\delta = (1-\varepsilon)\cdot(1-\varepsilon+(\lambda/d)\cdot\varepsilon)^{\ell-1} < (1-(1-(\lambda/d))\cdot\varepsilon)^{\ell}$. A proof of an upper-bound that is weaker than Eq. (8.15) is outlined in Exercise 8.30. Using Theorem 8.28 and an explicit $(2^t, \overline{\lambda}\cdot 2^t)$-expander, we get

Proposition 8.29 (The Expander Random Walk Generator):[40]

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

$$G(v_0, i_1, ..., i_{\ell'(k)-1}) \stackrel{\rm def}{=} (v_0, v_1, ..., v_{\ell'(k)-1}), \tag{8.16}$$

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

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

The stretch of G is optimized at $b(k) \approx k/2$ (and $\ell'(k) = k/2t$), but optimizing the stretch is not necessaily the goal in all applications. Expander random-walk generators have been used in a variety of areas (e.g., PCP and inapproximability (see [27, Sec. 11.1]), cryptography (see [87, Sec. 2.6]), and the design of various types of "pseudorandom" objects (see, in particular, Section D.3)).

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

[40] The common parameterization starts with parameters b and ℓ'. Given a uniformly chosen seed of length $b + O(\ell'-1)$, one can efficiently and deterministically generate a random sequence of ℓ' strings, each of length b, which is (ε,δ)-hitting for any $\varepsilon > 0$ and $\delta = (1-\Omega(\varepsilon))^{\ell'}$.

TYPE	distinguisher's resources	generator's resources	stretch (i.e., $\ell(k)$)	comments
gen.-purpose	$p(k)$-time, $\forall$ poly. p	poly(k)-time	poly(k)	Assumes OW[41]
derand. BPP	$2^{k/O(1)}$-time	$2^{O(k)}$-time	$2^{k/O(1)}$	Assumes EvEC[41]
space-bounded	$s(k)$-space	$O(k)$-space	$2^{k/O(s(k))}$	runs in time
robustness	$k/O(1)$-space	$O(k)$-space	poly(k)	poly$(k) \cdot \ell(k)$
t-wise indepen.	"t-wise"	poly$(k) \cdot \ell(k)$-time	$2^{k/O(t)}$	(e.g., pairwise)
small bias	"ε-bias"	poly$(k) \cdot \ell(k)$-time	$2^{k/O(1)} \cdot \varepsilon(k)$	
expander	"hitting"	poly$(k) \cdot \ell(k)$-time	$\ell'(k) \cdot b(k)$	
rand. walk	$(0.5, 2^{-\ell'(k)/O(1)})$-hitting for $\{0,1\}^{b(k)}$, with $\ell'(k) = ((k - b(k))/O(1)) + 1$.			

Figure 8.6: Pseudorandom generators at a glance

Chapter Notes

Figure 8.6 depicts some of the notions of pseudorandom generators discussed in this chapter. We highlight a key distinction between the case of general-purpose pseudorandom generators (treated in Section 8.3) and the other cases (cf. Sections 8.4 and 8.5): in the former case the distinguisher is more complex than the generator, whereas in the latter cases the generator is more complex than the distinguisher. Specifically, in the general-purpose case the generator runs in (some *fixed*) polynomial-time and needs to withstand *any* probabilistic polynomial-time distinguisher. In fact, some of the proofs presented in Section 8.3 utilize the fact that the distinguisher can invoke the generator on seeds of its choice. In contrast, the Nisan-Wigderson Generator, analyzed in Theorem 8.18 (of Section 8.4), runs more time than the distinguishers that it tries to foil, and the proof relies on this fact in an essential manner. Similarly, the space complexity of the space-resilient generators presented in Section 8.5 is higher than the space-bound on the distinguishers that they foil.

The general paradigm of pseudorandom generators. Our presentation, which views vastly different notions of pseudorandom generators as incarnations of a general paradigm, has emerged mostly in retrospect. We note that, while the historical technical development of the various notions was mostly unrelated, the case of general-purpose pseudorandom generators served as a source of inspiration to most of the other cases. In particular, the concept of computational indistinguishability, the connection between hardness and pseudorandomness, and the equivalence between pseudorandomness and unpredictability, appeared first in the context of general-purpose pseudorandom generators (and inspired the development of "generators for derandomization" and "generators for space bounded machines"). Indeed, the development of the special-purpose generators (see Section 8.6) was unrelated to all of these.

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

General-purpose pseudorandom generators. The concept of *computational indistinguishability*, which underlies the entire computational approach to randomness, was suggested by Goldwasser and Micali [104] in the context of defining secure encryption schemes. Indeed, computational indistinguishability plays a key role in cryptography (see Appendix C). The general formulation of computational indistinguishability is due to Yao [221]. Yao also observed (using the hybrid technique of [104]) that defining pseudorandom generators as producing sequences that are computationally indistinguishable from the corresponding uniform distribution is equivalent to defining such generators as producing unpredictable sequences. The latter definition originates in the earlier work of Blum and Micali [37].

Blum and Micali [37] pioneered the rigorous study of pseudorandom generators and, in particular, their construction based on some simple intractability assumption (in their case, the intractability of Discrete Logarithm problem over prime fields). Their work also introduces basic paradigms that were used in all subsequent improvements (cf., e.g., [221, 113]). We refer to the trasformation of computational difficulty into pseudorandomness, the use of hard-core predicates (defined in [37]), and the iteration paradigm (cf. Eq. (8.8)).

Theorem 8.11 (by which pseudorandom generators exist if and only if one-way functions exist) is due to Håstad, Impagliazzo, Levin and Luby [113], building upon the hard-core predicate of [95] (see Theorem 7.7). Unfortunately, the current proof of Theorem 8.11 is very complicated and unfit for presentation in a book of the current nature. Presenting a simpler and tighter (cf. §8.3.7.1) proof is indeed an important research project.

Pseudorandom functions (further discussed in Appendix C.3.3) were defined and first constructed by Goldreich, Goldwasser and Micali [91]. We advocate the study of a general theory of pseudorandom objects initiated in [92].

Derandomization of time-complexity classes. As observed by Yao [221], a non-uniformly strong notion of pseudorandom generators yields improved derandomization of time-complexity classes. A key observation of Nisan [160, 163] is that whenever a pseudorandom generator is used this way, it suffices to require that the generator runs in time exponential in its seed length, and so the generator may have running-time greater than the distinguisher (representing the algorithm to be derandomized). This observation underlines the construction of Nisan and Wigderson [160, 163], and is the basis for further improvements culminating in [121]. Part 1 of Theorem 8.19 (i.e., the so-called "high end" derandomization of $\mathcal{BPP}$) is due to Impagliazzo and Wigderson [121], whereas Part 2 (the "low end") is from [163].

The Nisan–Wigderson Generator [163] was subsequently used in several ways transcending its original presentation. We mention its application towards fooling non-deterministic machines (and thus derandomizing constant-round interactive proof systems) and to the construction of randomness extractors [208].

Space Pseudorandom Generators. As stated in the first paper on the subject of space-resilient pseudorandom generators [4][42], this research direction was inspired by the derandomization result obtained via use of general-purpose pseudorandom generators. The latter result (necessarily) depends on intractability assumptions, and so the objective was to find classes of algorithms for which derandomization is possible without relying on intractability assumptions. (This objective was achieved before for the case of constant-depth circuits.) Fundamentally different constructions of space pseudorandom generators were given in several works, but are superseeded by the two incomparable results mentioned in Section 8.5.2: Theorem 8.21 (a.k.a Nisan's Generator [161]) and Theorem 8.22 (a.k.a the Nisan–Zuckerman Generator [164]). These two results have been "interpolated" in [11]. Theorem 8.23 ($\mathcal{BPL} \subseteq \mathcal{SC}$) was proved by Nisan [162].

Special Purpose Generators. The various generators presented in Section 8.6 were not inspired by any of the other types of pseudorandom generator (nor even by the generic notion of pseudorandomness). Pairwise-independence generator were explicitly suggested in [51] (and are implicit in [47]). The generalization to t-wise independence (for $t \geq 2$) is due to [6]. Small-bias generators were first defined and constructed by Naor and Naor [158], and three simple constructions were subsequently given in [9]. The Expander Random Walk Generator was suggested by Ajtai, Komlos, and Szemerédi [4], who discovered that random walks on expander graphs provide a good approximation to repeated independent attempts for hitting any arbitrary fixed subset of sufficient density (within the vertex set). The analysis of the hitting property of such walks was subsequently improved, culminating in the bound cited in Theorem 8.28, which is taken from [125, Cor. 6.1].

(The foregoing historical notes do not mention several technical contributions that played an important role in the development of the area. For further details, the reader is referred to [86, Chap. 3]. In fact, the current chapter is a revision of [86, Chap. 3], providing more details for the main topics, and omitting [86, Sec. 3.6.4 and 3.6.5] (a revision of which appears in Appendix D).)

Exercises

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

Guideline: Use the Probabilistic Method (cf. [10]). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probabity that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\Pr[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/100}$. Next, using a union bound, prove the existence of a set $S \subset \{0,1\}^n$ of size

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

$2^{n/2}$ such that no circuit of size $2^{n/100}$ can distinguish a uniformly distributed element of S from a uniformly distributed element of $\{0,1\}^n$, where distinguishing means with a probability gap of at least $2^{-n/100}$.

Exercise 8.2 Let A be a probabilistic polynomial-time algorithm solving the search associated with the NP-relation R, and let A_G be as in Construction 8.2. Prove that it is infeasible to find an x on which A_G outputs a wrong solution; that is, assuming for simplicity that A has error probability $1/3$, prove that on input 1^n it is infeasible to find an $x \in \{0,1\}^n \cap L_R$ such that $\Pr[(x, A_G(x)) \notin R] > 0.4$, where $L_R \stackrel{\text{def}}{=} \{x : \exists y\,(x,y) \in R\}$.
(Hint: For x that violates the claim, it holds that $|\Pr[(x, A(x)) \notin R] - \Pr[(x, A_G(x)) \notin R]| > 0.06$.)

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

Exercise 8.4 Show that the existence of pseudorandom generators implies the existence of polynomial-time constructible probability ensembles that are statistically far apart and yet are computationally indistinguishable.
(Hint: lower-bound the statistical distance between $G(U_k)$ and $U_{\ell(k)}$, where G is a pseudorandom generator with stretch ℓ.)

Exercise 8.5 Prove that the sufficient condition in Exercise 8.4 is in fact necessary.[43] Recall that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are said to be statistically far apart if, for some positive polynomial p and all sufficiently large n, the variation distance between X_n and Y_n is greater than $1/p(n)$. Using the following three steps, prove that the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable implies the existence of pseudorandom generators.

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

   (Hint: Consider $\overline{X}_n = (X_n^{(1)}, ..., X_n^{(t(n))})$ and $\overline{Y}_n = (Y_n^{(1)}, ..., Y_n^{(t(n))})$, where the $X_n^{(i)}$'s (resp., $Y_n^{(i)}$'s) are independent copies of X_n (resp., Y_n), and $t(n) = O(n \cdot p(n)^2)$.)

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

   (Hint: Let S_0 and S_1 be sampling algorithms such that $X_n \equiv S_0(U_{\text{poly}(n)})$ and $Y_n \equiv S_1(U_{\text{poly}(n)})$. Consider the generator $G(\sigma, r) = (\sigma, S_\sigma(r))$, and the distribution Z_n that equals (U_1, X_n) with probability $1/2$ and (U_1, Y_n) otherwise. Note that in $G(U_1, U_{\text{poly}(n)})$ the first bit is almost determined by the rest, whereas in Z_n the first bit is statistically independent of the rest.)

[43] This exercise follows [84], which in turn builds on [113].

3. Using a false entropy generator, obtain one in which the excess entropy is $\sqrt{k}$, and using the latter construct a pseudorandom generator.

   (Hint: Use the ideas presented at the end of Section 8.3.5 (i.e., the discussion of the interesting direction of the proof of Theorem 8.11).)

Exercise 8.6 Prove that *if* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are computationally indistinguishable and A is a probabilistic polynomial-time algorithm then* $\{A(X_n)\}_{n\in\mathbb{N}}$ *and* $\{A(Y_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*
(Hint: If D distinguishes the latter ensembles then D' such that $D'(z) \stackrel{\text{def}}{=} D(A(z))$ distinguishes the former.)

Exercise 8.7 In continuation to Exercise 8.6, show that the conclusion may not hold in case A is not computationally bounded. That is, show that there exists computationally indistinguishable ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, and an exponential-time algorithm A such that $\{A(X_n)\}_{n\in\mathbb{N}}$ and $\{A(Y_n)\}_{n\in\mathbb{N}}$ are *not* computationally indistinguishable.
(Hint: For any pair of ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, consider the Boolean function f such that $f(z) = 1$ if and only if $\mathsf{Pr}[X_n = z] > \mathsf{Pr}[Y_n = z]$. Show that $|\mathsf{Pr}[f(X_n) = 1] - \mathsf{Pr}[f(Y_n) = 1]|$ equals the statistical difference between X_n and Y_n. Consider an adequate (approximate) implementation of f (e.g., approximate $\mathsf{Pr}[X_n = z]$ and $\mathsf{Pr}[Y_n = z]$ up to $\pm 2^{-2|z|}$), and use Exercise 8.1.)

Exercise 8.8 For G_1 and ℓ as in Construction 8.7, consider $G(s) \stackrel{\text{def}}{=} G_1^{\ell(|s|)}(s)$, where $G_1^i(x)$ denotes G_1 iterated i times on x (i.e., $G_1^i(x) = G_1^{i-1}(G_1(x))$ and $G_1^0(x) = x$). Prove that G is a pseudorandom generator of stretch ℓ. Reflect on the advantages of Construction 8.7 over the current construction.
(Hint: Use a hybrid argument, with the i^{th} hybrid being $G_1^i(U_{\ell(k)-i})$. Note that $G_1^{i+1}(U_{\ell(k)-(i+1)}) = G_1^i(G_1(U_{\ell(k)-i-1}))$ and $G_1^i(U_{\ell(k)-i}) = G_1^i(U_{|G_1(U_{\ell(k)-i-1})|})$, and use Exercise 8.6.)

Exercise 8.9 (pseudorandom versus unpredictability) Prove that a probability ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is pseudorandom if and only if it is unpredictable. For simplicity, we say that $\{Z_k\}_{k\in\mathbb{N}}$ is (next-bit) unpredictable if for every probabilistic polynomial-time algorithm A it holds that $\mathsf{Pr}_i[A(F_i(Z_k)) = B_{i+1}(Z_k)]$ is negligible, where $i \in \{0, ..., |Z_k| - 1\}$ is uniformly distributed, and $F_i(z)$ (resp., $B_{i+1}(z)$) denotes the i-bit prefix (resp., $i + 1^{\text{st}}$ bit) of z.

Guideline: Show that pseudorandomness implies polynomial-time unpredictability; that is, polynomial-time predictability violates pseudorandomness (because the uniform ensemble is unpredictable regardless of computing power). Use a hybrid argument to prove that unpredictability implies pseudorandomness. Specifically, the i^{th} hybrid consists of the i-bit long prefix of Z_k followed by $|Z_k| - i$ uniformly distributed bits. Thus, distinguishing the extreme hybrids (which correspond to Z_k and $U_{|Z_k|}$) implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability. For the last step, use an argument as in the proof of Proposition 8.9.

Exercise 8.10 Prove that a probability ensemble is unpredictable (from left to right) if and only if it is unpredictable from right to left (or in any other canonical order).
(Hint: use Exercise 8.9, and note that an ensemble is pseudorandom if and only if its reverse is pseudorandom.)

Exercise 8.11 Let f be 1-1 and length preserving, and b be a hard-core predicate of f. For any polynomial ℓ, prove that $\{G'(U_k)\}$ is unpredictable (in the sense of Exercise 8.9), where $G'(s) \stackrel{\rm def}{=} b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$.

Guideline: Suppose towards the contradiction that, for a uniformly distributed $j \in \{0,...,\ell(k)-1\}$, given the j-bit long prefix of $G'(U_k)$ an algorithm A' can predict the $j+1^{\rm st}$ bit of $G'(U_k)$. That is, given $b(f^{\ell(k)-1}(s)) \cdots b(f^{\ell(k)-j}(s))$, algorithm A' predicts $b(f^{\ell(k)-(j+1)}(s))$, where s is uniformly distributed in $\{0,1\}^k$. Consider an algorithm A that given $y = f(x)$ approximates $b(x)$ by invoking A' on input $b(f^{j-1}(y)) \cdots b(y)$, where j is uniformly selected in $\{0,...,\ell(k)-1\}$. Analyze the success probability of A using the fact that f induces a permutation over $\{0,1\}^n$, and thus $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$ is distributed identically to $b(f^{\ell(k)-1}(U_k)) \cdots b(f^{\ell(k)-j}(U_k)) \cdot b(f^{\ell(k)-(j+1)}(U_k))$.

Exercise 8.12 Prove that if G is a strong pseudorandom generator in the sense of Definition 8.12 then it a pseudorandom generator in the sense of Definition 8.1.
(Hint: consider a sequence of internal coin tosses that maximizes the probability in Eq. (8.2).)

Exercise 8.13 Show that there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (8.9), provided $\ell(k) > k$.
(Hint: The circuit may incorporate all values in the range of G and deciding by comparing its input to these values.)

Exercise 8.14 (constructing a set system for Theorem 8.18) For every $\gamma > 0$, show a construction of a set system S as in Condition 2 of Theorem 8.18, with $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$.

Guideline: We assume, without loss of generality, that $\gamma < 1$, and set $m(k) = (\gamma/2) \cdot k$ and $\ell(k) = 2^{\gamma m(k)/10}$. We construct the set system $S_1, ..., S_{\ell(k)}$ in iterations, selecting S_i as the first $m(k)$-subset of $[k]$ that has sufficiently small intersections with each of the previous sets $S_1, ..., S_{i-1}$. The existence of such a set S_i can be proved using the Probabilistic Method (cf. [10]). Specifically, for a fixed $m(k)$-subset S', the probability that a random $m(k)$-subset has intersection greater than $\gamma m(k)$ with S' is upper-bounded[44] by $2^{-\gamma m(k)/10}$, because the expected intersection size is $(\gamma/2) \cdot m(k)$. Thus, with positive probability a random $m(k)$-subset has intersection at most $\gamma m(k)$ with each of the previous $i - 1 < \ell(k) = 2^{\gamma m(k)/10}$ subsets. Note that we construct S_i in time $\binom{k}{m(k)} \cdot (i-1) \cdot m(k) < 2^k \cdot \ell(k) \cdot k$, and thus S is computable in time $k2^k \cdot \ell(k)^2 < 2^{2k}$.

[44] Applying the standard Chernoff Bound yields an upper-bound of $\exp(-(\gamma/2)^2 \cdot m(k))$, which suffices for $\ell(k) = 2^{\gamma^2 m(k)/10}$. However, using a Multiplicative Chernoff Bound yields an upper-bound of $\exp(-(\gamma/2) \cdot m(k)/3)$.

Exercise 8.15 Suppose that the sets S_i's in Construction 8.17 are disjoint and that $f : \{0,1\}^m \to \{0,1\}$ is T-inapproximable. Prove that for every circuit C of size $T - O(1)$ it holds that $|\mathsf{Pr}[C(G(U_k)) = 1] - \mathsf{Pr}[C(U_\ell) = 1]| < \ell/T$.

Guideline: Prove the contrapositive using Exercise 8.16. Note that the values of the $i + 1^{\text{st}}$ bit of $G(U_k)$ is statistically independent of the values of the first i bits of $G(U_k)$, and thus predicting it yields an approximator for f. Indeed, such an approximator can be obtained by fixing the the first i bits of $G(U_k)$ via an averaging argument.

Exercise 8.16 In continuation to Exercise 8.9, show that if there exists a circuit of size s that distinguishes Z_n from U_ℓ with gap δ, then there exists an $i < \ell = |Z_n|$ and a circuit of size $s + O(1)$ that given an i-bit long prefix of Z_n guesses the $i + 1^{\text{st}}$ bit with success probability at least $\frac{1}{2} + \frac{\delta}{\ell}$.
(Hint: defining hybrids as in Exercise 8.9, note that, for some i, the given circuit distinguishes the i^{th} hybrid from the $i + 1^{\text{st}}$ hybrid with gap at least δ/ℓ.)

Exercise 8.17 (Theorem 8.18, generalized) Let $\ell, m, m', T : \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k)^2 + \widetilde{O}(\ell(k)2^{m'(k)}) < T(m(k))$. Suppose that the following two conditions hold:

1. There exists an exponential-time computable function $f : \{0,1\}^* \to \{0,1\}$ that is T-inapproximable.
2. There exists an exponential-time computable function $S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|S(k,i)| = m(k)$ for every k and $i = 1, ..., \ell(k)$, and $|S(k,i) \cap S(k,j)| \leq m'(k)$ for every k and $i \neq j$.

Prove that using G as defined in Construction 8.17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.
(Hint: following the proof of Theorem 8.18, just note that the circuit constructed for approximating $f(U_{m(k)})$ has size $\ell(k)^2 + \ell(k) \cdot \widetilde{O}(2^{m'(k)})$ and success probability at least $(1/2) + (1/7\ell(k))$.)

Exercise 8.18 (Part 2 of Theorem 8.19) Prove that if for every polynomial T there exists a T-inapproximable predicate in $\mathcal{E}$ then $\mathcal{BPP} \subseteq \cap_{\varepsilon > 0}\text{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{n^\varepsilon}$.
(Hint: For any p-time algorithm, apply Exercise 8.17 using $\ell(k) = p(k^{1/\varepsilon})$, $m(k) = \sqrt{k}$ and $m'(k) = O(\log k)$. Revisit Exercise 8.14 in order to obtain a set system as required in Exercise 8.17 (for these parameters), and use Theorem 7.10.)

Exercise 8.19 Provide an explicit description of the generator outlined in the proof of Theorem 8.21.
(Hint: for $r \in \{0,1\}^n$ and $h^{(1)}, ..., h^{(t)} \in H_n$, the generaor outputs a 2^t-long sequence of n-bit strings such that the i^{th} block equals $h'(r)$, where h' is a composition of some of the $h^{(j)}$'s.)

Exercise 8.20 (adaptive t-wise independence tests) Prove that the output of a t-wise independence generator is indistinguishable to any test than examines t of the blocks, even if the examined blocks are selected adaptively (i.e., the location of the i^{th} block is determined based on the contents of the previously inspected blocks).

Guideline: First show that, without loss of generality, it suffices to consider deterministic (adaptive) tester. Next, show that the probability that such a tester sees any fixed sequence of t values at locations selected adaptively in the generator's output is $2^{-t \cdot b(k)}$, where $b(k)$ is the block length.

Exercise 8.21 (t-wise independence generator) Prove that G as defined in Proposition 8.24 produces a t-wise independent sequence over $\mathrm{GF}(2^{b(k)})$.

Guideline: For every t fixed indices $i_1, ..., i_t \in [\ell'(k)]$, consider the distribution of $G(U_k)_{i_1,...,i_t}$ (i.e., the projection of $G(U_k)$ on locations $i_1, ..., i_t$). Show that for every sequence of t possible values $v_1, ..., v_t \in \mathrm{GF}(2^{b(k)})$, there exists a unique seed $s \in \{0,1\}^k$ such that $G(s)_{i_1,...,i_t} = (v_1, ..., v_t)$.

Exercise 8.22 (pairwise independence generators) As a warm-up, consider a construction analogous to the one in Proposition 8.25, where the seed specifies an affine $b(k)$-by-$m(k)$ transformation. That is, for $s \in \{0,1\}^{b(k) \cdot m(k)}$ and $r \in \{0,1\}^{b(k)}$, where $k = b(k) \cdot m(k) + b(k)$, let

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

where A_s is an $b(k)$-by-$m(k)$ matrix specified by the string s. Show that G as in Eq. (8.17) is a pairwise independence generator of block-size b and stretch ℓ. (Note that a related construction appears in the proof of Theorem 7.7; see also Exercise 7.5.) Next, show that G as in Eq. (8.13) is a pairwise independence generator of block-size b and stretch ℓ.

Guideline: The following description applies to both constructions. First note that for every fixed $i \in [\ell'(k)]$, the i^{th} element in the sequence $G(U_k)$ is uniformly distributed in $\{0,1\}^{b(k)}$. Actually, show that for every fixed $s \in \{0,1\}^{k-b(k)}$, it holds that $G(s, U_{b(k)})_i$ is uniformly distributed in $\{0,1\}^{b(k)}$. Next note that it suffices to show that, for every $j \neq i$, conditioned on the value of the i^{th} element in $G(U_k)$, the j^{th} element is uniformly distributed in $\{0,1\}^{b(k)}$. The key technical detail is to show that for any non-zero vector $v \in \{0,1\}^{m(k)}$ it holds that $A_{U_{k-b(k)}} v$ (resp., $T_{U_{k-b(k)}} v$) is uniformly distributed in $\{0,1\}^{b(k)}$. This is easy in case of a random $b(k)$-by-$m(k)$ matrix, and can be proven also for a random Toeplitz matrix.

Exercise 8.23 (adaptive t-wise independence tests, revisited) In contrast to Exercise 8.20, we note that almost uniform distribution on any fixed t bit locations does not imply that an adaptive test that inspects t locations cannot detect "non-uniformity" (i.e., a "non random behavior" of the inspected sequence). Specifically, present a distribution over 2^{t-1}-bit long strings in which each $t-1$ fixed bit positions are $t \cdot 2^{-(t-1)}$-close to uniform, but some test that adaptively inspects t positions can distinguish this distribution from the uniform one with constant gap. (Hint: Modify the uniform distribution over $((t-1) + 2^{t-1})$-bit long strings such that the first $t-1$ locations indicate a bit position (among the rest) that is set to zero.)

Exercise 8.24 Suppose that G is an ε-bias generator with stretch ℓ. Show that equality between the $\ell(k)$-bit strings x and y can be probabilistically checked by comparing the inner product modulo 2 of x and $G(s)$ to the inner product modulo 2 of y and $G(s)$, where $s \in \{0,1\}^k$ is selected uniformly.

(Hint: reduce the problem to the special case in which $y = 0^{\ell(k)}$.)

Exercise 8.25 (bias versus statistical difference from uniform) Let X be a random variable assuming values in $\{0,1\}^t$. Prove that if X has bias at most ε over any non-empty set then the statistical difference between X and U_t is at most $2^{t/2} \cdot \varepsilon$, and that for every $x \in \{0,1\}^t$ it holds that $\mathsf{Pr}[X = x] = 2^{-t} \pm \varepsilon$.

Guideline: Consider the probability function $p : \{0,1\}^t \to [0,1]$ defined by $p(x) \stackrel{\text{def}}{=} \mathsf{Pr}[X = x]$, and let $\delta(x) \stackrel{\text{def}}{=} p(x) - 2^{-t}$ denote the deviation of p from the uniform probability function. Viewing the set of real functions over $\{0,1\}^t$ as a 2^t-dimensional vector space, we consider two orthonormal bases for this space. The first basis is of the (Kroniker) functions $\{k_\alpha\}_{\alpha \in \{0,1\}^t}$ such that $k_\alpha(x) = 1$ if $x = \alpha$ and $k_\alpha(x) = 0$ otherwise. The second basis is of the (normalize Fourier) functions $\{f_S\}_{S \subseteq [t]}$ defined by $f_S(x) \stackrel{\text{def}}{=} 2^{-t/2} \prod_{i \in S} (-1)^{x_i}$ (where $f_\emptyset \equiv 2^{-t/2}$).[45] Note that the bias of X over any $S \neq \emptyset$ equals $|\sum_x p(x) \cdot 2^{t/2} f_S(x)|$, which in turn equals $2^{t/2} |\sum_x \delta(x) f_S(x)|$. Thus, for every S (including the empty set), we have $|\sum_x \delta(x) f_S(x)| \leq 2^{-t/2}\varepsilon$, which means that the representation of δ in the normalize Fourier basis is by coefficients that have each an absolute value of at most $2^{-t/2}\varepsilon$. Thus, the Norm-2 of this vector of coefficients is bounded by $\sqrt{2^t \cdot (2^{-t/2}\varepsilon)^2} = \varepsilon$, and all claims follow by noting that they refer to norms of δ according to the Kroniker basis. In particular, Norm-2 is preserved under orthonormal bases, the max-norm is upper-bounded by Norm-2, and Norm-1 is upper-bounded by $\sqrt{2^t}$ times the value of the Norm-2.

Exercise 8.26 (The LFSR small-bias generator (following [9])) Using the following guidelines (and letting $t = k/2$), analyze the construction outlined following Theorem 8.26 (and depicted in Figure 8.5):

1. Prove that $r_i = \sum_{j=0}^{t-1} c_j^{(i)} \cdot s_j$, where $c_j^{(i)}$ is the coefficient of z^j in the (degree $t-1$) polynomial obtained by reducing z^i modulo the polynomial $f(z)$ (i.e., $z^i \equiv \sum_{j=0}^{t-1} c_j^{(i)} z^j \pmod{f(z)}$).

   (Hint: Recall that $z^t \equiv \sum_{j=0}^{t-1} f_j z^j \pmod{f(z)}$, and thus $z^i \equiv \sum_{j=0}^{t-1} f_j z^{i-t+j} \pmod{f(z)}$. Note the correspondance to $r_i = \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j}$.)

2. For any non-empty $S \subseteq \{0, ..., \ell(k) - 1\}$, evaluate the bias of the sequence $r_0, ..., r_{\ell(k)-1}$ over S, where f is a random irreducible polynomial of degree t and $s = (s_0, ..., s_{t-1}) \in \{0,1\}^t$ is uniformly distributed. Specifically:

   (a) For a fixed f and random $s \in \{0,1\}^t$, prove that $\sum_{i \in S} r_i$ has non-zero bias if and only if $f(z)$ divides $\sum_{i \in S} z^i$.

   (Hint: Note that $\sum_{i \in S} r_i = \sum_{j=0}^{t-1} \sum_{i \in S} c_j^{(i)} s_j$, and use Item 1.)

[45] Verify that both bases are indeed orthogonal (i.e., $\sum_x k_\alpha(x) k_\beta(x) = 0$ for every $\alpha \neq \beta$ and $\sum_x f_S(x) f_T(x) = 0$ for every $S \neq T$) and normal (i.e., $\sum_x k_\alpha(x)^2 = 1$ and $\sum_x f_S(x)^2 = 1$).

(b) Prove that the probability that a random irreducible polynomial of degree t divides $\sum_{i\in S} z^i$ is $\Theta(\ell(k)/2^t)$.

(Hint: A polynomial of degree n can be divided by at most n/d different irreducible polynomials of degree d. On the other hand, the number of irreducible polynomials of degree d over GF(2) is $\Theta(2^d/d)$.)

Conclude that for random f and s, the sequence $r_0, ..., r_{\ell(k)-1}$ has bias $O(\ell(k)/2^t)$.

Note that an implementation of the LFSR generator requires a mapping of random $k/2$-bit long seeds to almost random irreducible polynomials of degree $k/2$. Such a mapping can be constructed in $\exp(k)$ time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A more efficient mapping that uses a $O(k)$-bit long seek is described in [9, Sec. 8].

Exercise 8.27 (limitations on small-bias generators) Let G be an ε-bias generator with stretch ℓ, and view G as a mapping from $\text{GF}(2)^k$ to $\text{GF}(2)^{\ell(k)}$. As such, each bit in the output of G can be viewed as a polynomial in the k input variables (each ranging in GF(2)). Prove that if $\varepsilon(k) < 1$ and each of these polynomials has degree at most d then $\ell(k) \leq \sum_{i=1}^{d} \binom{k}{i}$.

Guideline: First note that, without loss of generality, all polynomials have a free term equal to zero. Then, consider the vector space spanned by all d-monomials over k variables (i.e., monomial having at most d variables). Since $\varepsilon(k) < 1$, the polynomials representing the output bits of G must correspond to a sequence of independent vectors in this space. Derive the following corollaries:

1. If $\varepsilon(k) < 1$ then $\ell(k) < 2^k$.

2. If $\varepsilon(k) < 1$ and $\ell(k) > k$ then G cannot be a linear transformation.

Note that $G(s) = (s, b(s))$, where $b(s_1, ..., s_k) = \sum_{i=1}^{k/2} s_i s_{(k/2)+i} \bmod 2$, is an ε-biased generator with $\varepsilon(k) = \exp(-\Omega(k))$.

(Hint: Focusing on bias over sets that include the last output bit, prove that without loss of generality it suffices to analyze the bias of $b(U_k)$.)

Exercise 8.28 (a sanity check for pseudorandomness) The following fact is suggested as a sanity check for candidate pseudorandom generators with respect to space-bounded machines. The fact (to be proven as an exercise) is that, for every $\varepsilon(\cdot)$ and $s(\cdot)$ such that $s(k) \geq 1$ for every k, if G is (s, ε)-pseudorandom (as per Definition 8.20), then G is an ε-bias generator.

Exercise 8.29 (approximate t-wise independent generators (following [158])) Combining Theorem 8.26 and relying on the linearity of the t-wise independent generator of Eq. (8.12), construct a generator producing ℓ-bit long sequences in which any t positions are at most ε-away from uniform (in variation distance), while using a seed of length $O(t + \log(1/\varepsilon) + \log\log \ell)$. (For max-norm a seed of length $O(\log(t/\varepsilon) + \log\log \ell)$ suffices.)

Guideline: First note that, for any t, ℓ' and b, the transformation of Eq. (8.12) can be implemented by a fixed linear (over GF(2)) transformation of a $t \cdot b$-bit seed into an ℓ-bit bit sequence, where $\ell = \ell' \cdot b$. It follows that there exists a fixed GF(2)-linear transformation T of a random seed of length $t \cdot b$, where $b = \log_2 \ell$, into a t-wise independent bit sequence of the length ℓ (i.e., $T\,U_{t \cdot b}$ is t-wise independent over $\{0,1\}^\ell$). Thus, every t rows of T are linearly independent. The key observation is that when we replace the aforementioned random seed by an ε'-biased sequence, every $i \le t$ positions in the output sequence have bias at most ε' (because they define a non-zero linear test on the bits of the ε'-biased sequence). Note that the length of the new seed (used to produce ε'-biased sequence of length $t \cdot b$) is $O(\log tb/\varepsilon')$. Applying Exercise 8.25, we conclude that any t positions are at most $2^{t/2} \cdot \varepsilon'$-away from uniform (in variation distance). Recall that this was obtained using a seed of length $O(\log(t/\varepsilon') + \log\log \ell)$, and the claim follows by using $\varepsilon' = 2^{-t/2} \cdot \varepsilon$.

Exercise 8.30 (A version of the Expander Random Walk Theorem) Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ stays in W is at most $(\rho + (\lambda/d)^2)^{\ell/2}$. In fact, prove a more general claim that refers to the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$. The claimed upper-bound is

$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell-1} \sqrt{\rho_i + (\lambda/d)^2}, \tag{8.18}$$

where $\rho_i \stackrel{\rm def}{=} |W_i|/|V|$.

Guideline: View the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in the graph and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first trasformation shrinks the component that is orthogonal to the uniform distribution (which is the first eigenvalue of the adjacency matrix of the expander), whereas the second trasformation shrinks the component that is in the direction of the uniform distribution. For further details, see §E.2.1.3.

Exercise 8.31 Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ visits W more than $\alpha\ell$ times is smaller than $\binom{\ell}{\alpha\ell} \cdot (\rho + (\lambda/d)^2)^{\alpha\ell/2}$. For example, for $\alpha = 1/2$ and $\lambda/d < \sqrt{\rho}$, we get an upper-bound of $(32\rho)^{\ell/4}$. We comment that much better bounds can be obtained (cf. [82]).
(Hint: Use a union bound on all possible sequences of $m = \alpha\ell$ visits, and upper-bound the probability of visiting W in steps $j_1, ..., j_m$ by applying Eq. (8.18) with $W_i = W$ if $i \in \{j_1, ..., j_m\}$ and $W = V$ otherwise.)

Chapter 9

Probabilistic Proof Systems

A proof is whatever convinces me.

Shimon Even (1935–2004)

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last couple of decades. In this chapter, we concentrate on three such proof systems: *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. These proof systems share a common (untraditional) feature – they carry a probability of error; yet, this probability is explicitly bounded and can be reduced by successive application of the proof system. The gain in allowing this untraditional relaxation is substantial, as demonstrated by the three results mentioned in the summary.

Summary: The association of efficient procedures with *deterministic* polynomial-time procedures is the basis for viewing NP-proof systems as the canonical formulation of proof systems (with efficient verification procedures). Allowing *probabilistic* verification procedures and, moreover, ruling by statistical evidence gives rise to various types of probabilistic proof systems. These probabilistic proof systems carry a probability of error (which is explicitly bounded), but they offer various advantages over the traditional (deterministic and errorless) proof systems.

Randomized and interactive verification procedures, giving rise to *interactive proof systems*, seem much more powerful than their deterministic counterparts. In particular, such interactive proof systems exist for any set in $\mathcal{PSPACE} \supseteq \text{co}\mathcal{NP}$ (e.g., for the set of unsatisfied propositional formulae), whereas it is widely believed that some sets in $\text{co}\mathcal{NP}$ do *not* have NP-proof systems (i.e., $\mathcal{NP} \neq \text{co}\mathcal{NP}$). We stress that a "proof" in this context is not a fixed and static object, but rather a randomized (and dynamic) process in which the verifier interacts with the prover.

Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly.

Such randomized and interactive verification procedures allow for the introduction of *zero-knowledge proofs*, which are of great conceptual and practical interest (especially in cryptography). Loosely speaking, zero-knowledge proofs are interactive proofs that yield nothing (to the verifier) beyond the fact that the assertion is indeed valid. For example, a zero-knowledge proof that a certain propositional formula is satisfiable does not reveal a satisfying assignment to the formula nor any partial information regarding such an assignment (e.g., whether the first variable can assume the value `true`). Thus, zero-knowledge proofs exhibit an extreme contrast between being convinced of the validity of a statement and learning anything in addition (while receiving such a convincing proof). Under reasonable complexity assumptions (i.e., assuming the existence of one-way functions), every set in $\mathcal{NP}$ has a zero-knowledge proof system.

NP-proofs can be efficiently transformed into a (redundant) form that offers a trade-off between the number of (randomly) examined locations in the resulting proof and the confidence in its validity. It particular, it is known that any set in $\mathcal{NP}$ has an NP-proof system that supports probabilistic verification such that the error probability decreases exponentially with the number of bits read from the alleged proof. These redundant NP-proofs are called *probabilistically checkable proofs* (or PCPs). In addition to their conceptually fascinating nature, PCPs have played a key role in the study of the complexity of approximation problems.

Introduction and Preliminaries

The glory attached to the creativity involved in finding proofs, makes us forget that it is the less glorified procedure of verification that gives proofs their value. Conceptually speaking, proofs are secondary to the verification procedure; whereas technically speaking, proof systems are defined in terms of their verification procedures.

The notion of a verification procedure presumes the notion of computation and furthermore the notion of efficient computation. This implicit stipulation is made explicit in the definition of $\mathcal{NP}$ (cf. Definition 2.5), in which efficient computation is associated with (deterministic) polynomial-time algorithms.[1] Thus, NP provides the ultimate formulation of proof systems (with efficient verification procedures)

[1] Recall that the formulation of NP-proof systems explicitly restricts the length of proofs to be polynomial in the length of the assertion. Thus, verification is performed in a number of steps that is polynomial in the length of the assertion. We comment that deterministic proof systems that allow for longer proofs (but require that verification is efficient in terms of the length of the alleded proof) can be modeled as NP-proof systems by adequate padding (of the assertion).

as long as one associates efficient procedures with *deterministic* polynomial-time algorithms. However, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures. In particular:

- Interactive proof systems, which employ randomized and interactive verification procedures, seem much more powerful than their deterministic counterparts.
- Such interactive proof systems allow for the introduction of zero-knowledge proofs, which are of great theoretical and practical interest.
- NP-proofs can be efficiently transformed into a (redundant) form that offers a trade-off between the number of randomly examined locations in the alleged proof and the confidence in its validity.

In all these cases, explicit bounds are imposed on the computational complexity of the verification procedure, which in turn is personified by the notion of a verifier. Furthermore, in all these proof systems, the verifier is allowed to toss coins and rule by statistical evidence. Thus, all these proof systems carry a probability of error; yet, this probability is explicitly bounded and, furthermore, can be reduced by successive application of the proof system.

One important convention. When presenting a proof system, we state all complexity bounds in terms of the length of the assertion to be proven (which is viewed as an input to the verifier). Namely, when we say "polynomial-time" we mean time that is polynomial in the length of this assertion. Actually, as will become evident, this is *the* natural choice in all the cases that we consider. Note that this convention is consistent with the definition of NP-proof systems (cf. Definition 2.5), because $\mathrm{poly}(|(x,y)|) = \mathrm{poly}(|x|)$ for $|y| = \mathrm{poly}(|x|)$.

Notational Conventions. Denote by `poly` the set of all integer functions bounded by a polynomial and by `log` the set of all integer functions bounded by a logarithmic function (i.e., $f \in \texttt{log}$ iff $f(n) = O(\log n)$). All complexity measures mentioned in the subsequent exposition are assumed to be constructible in polynomial-time.

Organization. In Section 9.1 we present the basic definitions and results regarding interactive proof systems. The definition of an interactive proof systems is the starting point for a discussion of zero-knowledge proofs, which is provided in Section 9.2. Section 9.3, which presents the basic definitions and results regarding probabilistically checkable proofs (PCP), can be read independently of the other sections.

9.1 Interactive Proof Systems

In light of the growing acceptability of randomized and distributed computations, it is only natural to associate the notion of efficient computation with probabilistic and interactive polynomial-time computations. This leads naturally to the notion of an interactive proof system in which the verification procedure is interactive and randomized, rather than being non-interactive and deterministic. Thus, a "proof" in this context is not a fixed and static object, but rather a randomized (dynamic) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly. The foregoing discussion, as well as the definition provided in Section 9.1.1, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proof systems). Before turning to the actual definition, we highlight and further discuss some of the foregoing issues.

A static object versus an interactive process. Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements that are either self-evident or are derived from previous statements via self-evident rules. Actually, both conceptually and technically, it is more accurate to substitute the phrase "self-evident" by the phrase "commonly agreed" (because, at the last account, self-evidence is a matter of common agreement). In fact, in the formal study of proofs (i.e., logic), the commonly agreed statements are called *axioms*, whereas the commonly agreed rules are referred to as *derivation rules*. We highlight *a key property of mathematics proofs: proofs are viewed as fixed* (*static*) *objects*. In contrast, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, a proof is not a fixed object but rather a process by which the validity of an assertion is established. For example, the cross-examination of a witness in court is considered a proof in law, and failure to answer a rival's claim is considered a proof in daily discussions. The latter convention (by which failure to answer a claim is considered a proof) is common in philosophy, politics, and often even in Science. Furthermore, some technical "proofs by contradiction" take a similar attitude by emulating an imaginary debate with a potential generic skeptic.

We note that, in mathematics, proofs are often considered more fundamental than their consequence (i.e., the theorem). In contrast, in many daily situations, proofs are considered secondary (in importance) to their consequence.

The aforementioned daily attitudes will be adequate in the current chapter, where proofs are the object of study. We will be interested in modeling proofs that can be verified by automated procedures, which are designed to check the validity of potential proofs and are oblivious of additional features such as beauty, insight, etc. Thus, we view proofs merely as a vehicle for the verification of the validity of the claimed assertion. In fact, this attitude gets to an extreme in the case of zero-knowledge proofs, where we actually require that the proofs themselve be useless beyond being convincing of the validity of the claimed assertion. We note that the proof systems that we study refer to mundane theorems (e.g., asserting that a *specific* propositional formula is not satisfiable or that a party sent a message as instructed by a predetermined protocol). We stress that the (meta) theorems that we shall state regarding these proof systems will be proven in the traditional mathematical sense.

Prover and Verifier. The notion of a prover is implicit in all discussions of proofs, be it in mathematics or in other situations: the prover is the (sometimes hidden or transcendental) entity providing the proof. In contrast, the notion of a verifier tends to be more explicit in such discussions, which typically emphasize the *verification process*, or in other words the role of the verifier. Both in mathematics and in daily situations, proofs are defined in terms of the verification procedure. The verification procedure is considered to be relatively simple, and the burden is placed on the party/person supplying the proof (i.e., the prover). The asymmetry between the complexity of the verification task and the complexity of the theorem-proving task is captured by the definition of NP-proof systems (i.e., verification is required to be efficient whereas $\mathcal{P} \neq \mathcal{NP}$ implies that in some cases finding adequate proofs is infeasible).

We highlight the "distrustful attitude" towards the prover, which underlies any proof system. If the verifier trusts the prover then no proof is needed. Hence, whenever discussing a proof system one considers a setting in which the verifier is not trusting the prover, and furthermore is skeptic of anything that the prover says.

Completeness and Soundness. Two fundamental properties of a proof system (i.e., of a verification procedure) are its *soundness* (or *validity*) and *completeness*. The soundness property asserts that the verification procedure cannot be "tricked" into accepting false statements. In other words, *soundness* captures the verifier's ability to protect itself from being convinced of false statements (no matter what the prover does in order to fool it). On the other hand, *completeness* captures the ability of some prover to convince the verifier of true statements (belonging to some predetermined set of true statements). Note that both properties are essential to the very notion of a proof system.

We note that not every set of true statements has a "reasonable" proof system in which each of these statements can be proven (while no false statement can be "proven"). This fundamental fact is given a precise meaning in results such as *Gödel's Incompleteness Theorem* and Turing's theorem regarding the *undecidability of the Halting Problem*. This section is devoted to the study of a liberal notion of "reasonable proof systems".

9.1.1 Definition

Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion. Specifically, the verifier employs a probabilistic polynomial-time strategy. It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). On the other hand, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter what strategy is being employed by the prover. (The error probability can be reduced by running such a proof system several times.)

Formally, a strategy for a party describes the *party's next move* (i.e., its next message or its final decision) *as a function of the common input* (i.e., the aforementioned assertion), *its internal coin tosses, and all messages it has received so far.* That is, we assume that each party records the outcomes of its past coin tosses as well as all the messages it has received, and determines its moves based on these. Thus, an interaction between two parties, employing strategies A and B respectively, is determined by the common input, denoted x, and the randomness of both parties, denoted r_A and r_B. Assuming that A takes the first move (and B takes the last one), the corresponding interaction transcript (on common input x and randomness r_A and r_B) is $\alpha_1, \beta_1, ..., \alpha_t, \beta_t$, where $\alpha_i = A(x, r_A, \beta_1, ..., \beta_{i-1})$ and $\beta_i = B(x, r_B, \alpha_1, ..., \alpha_i)$. The corresponding final decision of A is defined as $A(x, r_A, \beta_1, ..., \beta_t)$.

We say that a party employs a probabilistic polynomial-time strategy if its next move can be computed in a number of steps that is *polynomial in the length of the common input.* In particular, this means that, on input common input x, the strategy may only consider a polynomial in $|x|$ many messages, which are each of poly($|x|$) length.[2] Intuitively, if the other party exceeds an a priori (polynomial in $|x|$) bound on the total length of the messages that it is allowed to send, then the execution is suspended. Referring to the foregoing strategies, we say that A is a probabilistic polynomial-time strategy if, for every i and $r_A, \beta_1, ..., \beta_i$, the value of $A(x, r_A, \beta_1, ..., \beta_i)$ can be computed in time polynomial in $|x|$. Again, in proper use, it must hold that $|r_A|, t$ and the $|\beta_i|$'s are all polynomial in $|x|$.

Definition 9.1 (Interactive Proof systems – IP):[3] *An* interactive proof system for a set *S is a two-party game, between a* verifier *executing a probabilistic polynomial-time strategy* (denoted V) *and a* prover *that executes a computationally unbounded strategy* (denoted P)*, satisfying the following two conditions:*

- Completeness: *For every $x \in S$, the verifier V always accepts after interacting with the prover P on common input x.*
- Soundness: *For every $x \notin S$ and every strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$ after interacting with P^* on common input x.*

We denote by $\mathcal{IP}$ the class of sets having interactive proof systems.

The error probability (in the soundness condition) can be reduced by successive applications of the proof system. (This is easy to see in the case of sequential repetitions, but holds also for parallel repetitions; see Exercise 9.1.) In particular, repeating the proving process for k times, reduces the probability that the verifier is fooled (i.e., accepts a false assertion) to 2^{-k}, and we can afford doing so for any $k = \text{poly}(|x|)$. Still, we have relaxed the requirements from the verification procedure by allowing it to interact with the prover, toss coins and risk some (bounded) error probability. (Variants on the basic definition are discussed in Section 9.1.3.)

[2] Needless to say, the number of internal coin tosses fed to a polynomial-time strategy must also be bounded by a polynomial in the length of x.

[3] We follow the convention of specifying strategies for both the verifier and the prover. An alternative presentation only specifies the verifier's strategy, while rephrasing the completeness condition as follows: *There exists a prover strategy P so that, for every $x \in S$, the verifier V always accepts after interacting with P on common input x.*

The role of randomness. Randomness is essential to the power of interactive proofs; that is, if randomness is not allowed, then interactive proof systems collapse to NP-proof systems. The reason being that, in case the verifier is deterministic, the prover can predict the verifier's part of the interaction. Thus, the prover can just supply its own answers to the verifier's (predictable) questions, and the verifier can just check that these answers are convincing. Actually we establish that soundness error (and not merely randomized verification) is essential to the power of interactive proof systems (i.e., their ability to reach beyond NP-proofs).

Proposition 9.2 *Suppose that S has an interactive proof system (P, V) with* no soundness error*; that is, for every $x \notin S$ and every potential strategy P^*, the verifier V rejects with probability* one *after interacting with P^* on common input x. Then $S \in \mathcal{NP}$.*

Proof: We may assume, without loss of generality, that V is deterministic (by just fixing arbitrarily the contents of its random-tape and noting that both (perfect) completeness and perfect (i.e., errorless) soundness still hold). Since V is deterministic, the prover may predict each message sent by V, and thus a sequence of optimal prover's messages (i.e., a sequence of messages leading V to accept x) can be (pre)determined (without interacting with V) *based solely on the common input x*. (In contrast, in the case that V is randomized, its random coins may effect the messages that it sends, which in turn may effect the optimal prover's responses.) Thus, $x \in S$ if and only if there exists a sequence of (prover's) messages that make (the deterministic) V accept x. It follows that $S \in \mathcal{NP}$. ■

Indeed, the punch-line of the foregoing proof is that the prover gains nothing from interacting with an easily predictable verifier (i.e., a verifier that determines its messages in deterministic polynomial-time based on the common input and the prover's prior messages). The prover can just produce the entire interaction by itself (and send it to the verifier for verification). *The moral is is that there is no point to interact with a party whose moves are easily predictable.* This moral represents the prover's point of view (regarding deterministic verifiers). Certainly, from the verifier's point of view it is beneficial to interact with the prover, because the latter is computationally stronger (and thus its moves may not be easily predictable by the verifier even in case they are predictable in an information theoretic sense).

9.1.2 The Power of Interactive Proofs

We have seen that randomness is essential to the power of interactive proof systems in the sense that without randomness interactive proofs are not more powerful than NP-proofs. Indeed, the power of interactive proof arises from the combination of randomization and interaction. We first demonstrate this point by a simple proof system for a specific coNP-set that is not known to have an NP-proof system, and next prove the celebrated result $\mathcal{IP} = \mathcal{PSPACE}$, which suggests that interactive proofs are much stronger than NP-proofs.

9.1.2.1 A simple example

> *One day on the Olympus, bright-eyed Athena claimed that Nectar poured out of the new silver-coated jars tastes less good than Nectar poured out of the older gold-decorated jars. Mighty Zeus, who was forced to introduce the new jars by the practically oriented Hera, was annoyed at the claim. He ordered that Athena be served one hundred glasses of Nectar, each poured at random either from an old jar or from a new one, and that she tell the source of the drink in each glass. To everybody's surprise, wise Athena correctly identified the source of each serving, to which the Father of the Gods responded "my child, you are either right or extremely lucky." Since all gods knew that being lucky was not one of the attributes of Pallas-Athena, they all concluded that the impeccable goddess was right in her claim.*

The foregoing story illustrates the main idea underlying the interactive proof for Graph Non-Isomorphism, presented in Construction 9.3. Informally, this interactive proof system is designed for proving dissimilarity of two given objects (in the foregoing story these are the two brands of Nectar, whereas in Construction 9.3 these are two non-isomorphic graphs). We note that, typically, proving similarity between objects is easy, because one can present a mapping (of one object to the other) that demonstrates this similarity. In contrast, proving dissimilarity seems harder, because in general there seems to be no succinct proof of dissimilarity. More generally, it is typically easy to prove the existence of an easily verifiable structure in the given object by merely presenting this structure, but proving the non-existence of such a structure seems hard. Formally, membership in an NP-set is proved by presenting an NP-witness, but it is not clear how to prove the non-existence of such witness. Indeed, recall that the common belief is that co$\mathcal{NP} \neq \mathcal{NP}$.

Recall that two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called isomorphic if there exists a 1-1 and onto mapping, ϕ, from the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$. The ("edge preserving") mapping ϕ, if existing, is called an *isomorphism* between the graphs. The following protocol allows proving that two graphs are not isomorphic. (We note that it is not known whether or not such a statement can be proven via an NP-proof system.)

Construction 9.3 (Interactive proof for Graph Non-Isomorphism):

- Common Input: *A pair of graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Suppose, without loss of generality, that $V_1 = \{1, 2, ..., |V_1|\}$, and similarly for V_2.*
- Verifier's first step (V1): *The verifier selects at random one of the two input graphs, and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly $\sigma \in \{1, 2\}$, and a random permutation π from the set of permutations over the vertex set V_σ. The verifier constructs a graph with vertex set V_σ and edge set*

  $$E \stackrel{\text{def}}{=} \{\{\pi(u), \pi(v)\} : \{u, v\} \in E_\sigma\}$$

  and sends (V_σ, E) to the prover.

- Motivating Remark: *If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient algorithm) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic, then a random isomorphic copy of one graph is distributed identically to a random isomorphic copy of the other graph.*
- Prover's step: *Upon receiving a graph, $G' = (V', E')$, from the verifier, the prover finds a $\tau \in \{1,2\}$ such that the graph G' is isomorphic to the input graph G_τ. (If both $\tau = 1, 2$ satisfy the condition then τ is selected arbitrarily. In case no $\tau \in \{1,2\}$ satisfies the condition, τ is set to 0). The prover sends τ to the verifier.*
- Verifier's second step (V2): *If the message, τ, received from the prover equals σ (chosen in Step V1) then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).*

The verifier's strategy in Construction 9.3 is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's strategy, but this is not required. The motivating remark justifies the claim that Construction 9.3 constitutes an interactive proof system for the set of pairs of non-isomorphic graphs.[4] Recall that the latter is a co$\mathcal{NP}$-set (which is not known to be in $\mathcal{NP}$).

9.1.2.2 The full power of interactive proofs

The interactive proof system of Construction 9.3 refers to a specific coNP-set that is not known to be in $\mathcal{NP}$. It turns out that interactive proof systems are powerful enough to prove membership in *any* coNP-set (e.g., prove that a graph is not 3-colorable). Thus, assuming that $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$, this establishes that interactive proof systems are more powerful than NP-proof systems. Furthermore, the class of sets having interactive proof systems coincides with the class of sets that can be decided using a polynomial amount of work-space.

Theorem 9.4 (The IP Theorem): $\mathcal{IP} = \mathcal{PSPACE}$.

Recall that it is widely believed that $\mathcal{NP}$ is a *proper* subset of $\mathcal{PSPACE}$. Thus, under this conjecture, interactive proofs are more powerful than NP-proofs.

Sketch of the Proof of Theorem 9.4

Theorem 9.4, was established using algebraic methods (see details below). In particular, the following approach – unprecedented in complexity theory – was employed:

[4] In case G_1 is not isomorphic to G_2, no graph can be isomorphic to both G_1 and G_2. On the other hand, if G_1 and G_2 are isomorphic then for every G' the number of isomorphisms between G_1 and G' equals the number of isomorphisms between G_2 and G'. It follows that in this case G' yields no information about the σ chosen by V, and so no prover may convince V with probability exceeding $1/2$.

In order to demonstrate that a particular set is in a particular class, an arithmetic generalization of the Boolean problem is presented, and (elementary) algebraic methods are applied for showing that the arithmetic problem is solvable within the class. Following is a sketch of the proof. We first show that co$\mathcal{NP} \subseteq \mathcal{IP}$, by presenting an interactive proof system for the co$\mathcal{NP}$-complete set of non-satisfiable CNF formulae. Next we extend this proof system to obtain one for the $\mathcal{PSPACE}$-complete set of non-satisfiable Quantified Boolean Formulae. Finally, we observe that $\mathcal{IP} \subseteq \mathcal{PSPACE}$.

Teaching note: We present a sketch of the proof that focuses on the main ideas, and omit various implementation details (which can be found in [150, 191]). Furthermore, we devote most of the presentation to establishing that co$\mathcal{NP} \subseteq \mathcal{IP}$, and recommend doing the same in class.

Arithmetization of Boolean (CNF) formulae: Given a Boolean (CNF) formula, we replace the Boolean variables by integer variables, OR-clauses by sums, and the top level conjunction by a product. Then we sum over all 0-1 assignments to these variables. For example, the Boolean formula

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee \neg x_4)$$

is replaces by the arithmetic expression

$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

and the Boolean formula is non-satisfiable if and only if the sum of the arithmetic expression, taken over all choices of $x_1, x_2, ..., x_{17} \in \{0,1\}$, equals 0. Observe that the arithmetic expression is a low degree polynomial over the integers (i.e., its degree is upper-bounded by the number of clauses). Also observe that, for any Boolean formula, the value of the corresponding arithmetic expression (for any choice of $x_1, ..., x_n \in \{0,1\}$) resides within the interval $[0, v^m]$, where v is the maximum number of variables in a clause, and m is the number of clauses. Summing over all 2^n possible 0-1 assignments, where $n < vm$ is the number of variables, the result resides in $[0, 2^n v^m]$.

Moving to a Finite Field: Whenever we check equality between two integers in $[0, M]$, it suffices to check equality mod q, where $q > M$. The benefit is that the arithmetic is now in a finite field (mod q) and so certain things are "nicer" (e.g., uniformly selecting a value). Thus, proving that a CNF formula is not satisfiable reduces to proving equality of the following form

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi(x_1, ..., x_n) \equiv 0 \pmod{q}, \tag{9.1}$$

where ϕ is a low degree multi-variate polynomial.

The actual protocol: stripping summations in iterations. We strip off summations in iterations. In each iteration the prover is supposed to supply the univariate polynomial representing the expression in one (currently stripped) variable. (By the foregoing observation, this is a low degree polynomial and so has a short description.) The verifier checks that the polynomial (say, p) is of low degree, and that it corresponds to the current value (say, v) being claimed (i.e., $p(0)+p(1)\equiv v$). Next, the verifier randomly instantiates the variable (i.e., selects uniformly $r\in \mathrm{GF}(q)$)[5], yielding a new value to be claimed for the resulting expression (i.e., the verifier computes $v\leftarrow p(r)$, and expects a proof that the residual expression equals v). The verifier sends the uniformly chosen instantiation (i.e., r) to the prover, and the proceed to the next iteration (which refers to the residual expression and to the value v). At the end of the last iteration, the verifier has a closed form expression (i.e., an expression without formal summations), which can be easily checked against the claimed value.

A single iteration (revisited): The i^{th} iteration is aimed at proving a claim of the form

$$\sum_{x_i=0,1}\cdots\sum_{x_n=0,1}\phi(r_1,...,r_{i-1},x_i,x_{i+1},...,x_n)\equiv v_{i-1}\pmod{q}, \tag{9.2}$$

where $v_0=0$, and $r_1,...,r_{i-1}$ and v_{i-1} are as determined in previous iterations. The i^{th} iteration consists of two steps (messages): a prover step followed by a verifier step. The prover is supposed to provide the verifier with the univariate polynomial p_i that satisfies

$$p_i(z)\stackrel{\text{def}}{=}\sum_{x_{i+1}=0,1}\cdots\sum_{x_n=0,1}\phi(r_1,...,r_{i-1},z,x_{i+1},...,x_n)\bmod q\,. \tag{9.3}$$

Denote by p_i' the actual polynomial sent by the prover (i.e., the honest prover sets $p_i'=p_i$). Then, the verifier first checks if $p_i'(0)+p_i'(1)\equiv v_{i-1}\pmod{q}$, and next uniformly selects $r_i\in\mathrm{GF}(q)$ and sends it to the prover. Needless to say, the verifier will reject if the first check is violated. The claim to be proven in the next iteration is

$$\sum_{x_{i+1}=0,1}\cdots\sum_{x_n=0,1}\phi(r_1,...,r_{i-1},r_i,x_{i+1},...,x_n)\equiv v_i\pmod{q}, \tag{9.4}$$

where $v_i\stackrel{\text{def}}{=}p_i'(r_i)\bmod q$.

Completeness of the protocol: When the initial claim (i.e., Eq. (9.1)) holds, the prover can supply the correct polynomials (as determined in Eq. (9.3)), and this will lead the verifier to always accept.

[5]Here and elsewhere, we denote by $\mathrm{GF}(q)$ the finite field having q elements.

Soundness of the protocol: It suffices to upper-bound the probability that, for a particular iteration, the entry claim (i.e., Eq. (9.2)) is false while the ending claim (i.e., Eq. (9.4)) is valid. Both claims refer to the current summation expression being equal to the current value, where 'current' means either at the beginning of the iteration or at its end. Let $p(\cdot)$ be the actual polynomial representing the expression when stripping the current variable, and let $p'(\cdot)$ be any potential answer by the prover. We may assume that $p'(0)+p'(1) \equiv v \pmod q$ and that p' is of low-degree (as otherwise the verifier will reject). Using our hypothesis (that the entry claim of Eq. (9.2) is false), we know that $p(0)+p(1) \not\equiv v \pmod q$. Thus, p' and p are different low-degree polynomials, and so they may agree on very few points (if at all). In case the verifier instantiation (i.e., its choice of random r) does not happen to be one of these few points, the ending claim (i.e., Eq. (9.4)) is false too. Details are left as an exercise (see Exercise 9.2).

This establishes that the set of non-satisfiable CNF formulae has an interactive proof system. Actually, the same proof system can be used to prove that a given formula has a given number of satisfying assignment; i.e., prove membership in the ("counting") set

$$\{(\phi,k) : |\{\tau : \phi(\tau)=1\}| = k\}.$$

Using adequate reductions, it follows that every problem in $\#\mathcal{P}$ has an interactive proof system (i.e., for every $R \in \mathcal{PC}$, the set $\{(x,k) : |\{y : (x,y)\in R\}| = k\}$ is in $\mathcal{IP}$). Proving that $\mathcal{PSPACE} \subseteq \mathcal{IP}$ requires a little more work.

Interactive Proofs for PSPACE (basic idea). We present an interactive proof for the set of satisfied Quantified Boolean Formulae (QBF), which is complete for $\mathcal{PSPACE}$ (see Theorem 5.15). Recall that the number of quantifiers in such formulae is unbounded (e.g., it may be polynomially related to the length of the input), that there are both existential and universal quantifiers, and furthermore these quantifiers may alternate. In the arithmetization of these formulae, we replace existential quantifiers by summations and universal quantifiers by products. Two difficulties arise when considering the application of the forgoing protocol to the resulting arithmetic expression. Firstly, the value of the expression (which may involve a big number of nested formal products) is only upper-bounded by a double exponential function (in the length of the input). Secondly, when stripping a summation (or a product), the expression may be a polynomial of high degree (due to nested formal products that may appear in the remaining expression). For example, both phenomena occur in the following expression

$$\sum_{x=0,1} \prod_{y_1=0,1} \cdots \prod_{y_n=0,1} (x+y_n),$$

which equals $\sum_{x=0,1} x^{2^{n-1}} \cdot (1+x)^{2^{n-1}}$. The first difficulty is easy to resolve by using the Chinese Reminder Theorem (i.e., if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3, \mathrm{poly}(\log M)]$). Thus, we let the verifier selects a random prime q of length that is linear in the length of the original formula, and the two parties consider the arithmetic expression reduced modulo this q. The second difficulty is resolved by noting that $\mathcal{PSPACE}$ is actually reducible to a special form of QBF in which no variable appears both to the left and to the right of more than one universal quantifier (see the proof of Theorem 5.15 or alternatively Exercise 9.4). It follows that when arithmetizing and stripping summations (or products) from the resulting arithmetic expression, the corresponding univariate polynomial is of low degree (i.e., at most twice the length of the original formula, where the factor of two is due to the single universal quantifier that has this variable quantified on its left and appearing on its right).

IP is contained in PSPACE: We show that, for every interactive proof system, there exists an *optimal prover strategy* that can be implemented in polynomial-space, where an optimal prover strategy is one that maximizes the probability that the prescribed verifier accepts the common input. It follows that $\mathcal{IP} \subseteq \mathcal{PSPACE}$, because (for every $S \in \mathcal{IP}$) we can emulate the interaction of the prescribed verifier with an optimal prover strategy in polynomial space.

Proposition 9.5 *Let V be a probabilistic polynomial-time interactive machine. Then, there exists a polynomial-space computable prover strategy f that, for every x maximizes the probability that V accepts x. That is, for every P^* and every x it holds that the probability that V accepts x after interacting with P^* is upper-bounded by the probability that V accepts x after interacting with f.*

Proof Sketch: For every common input x and any possible partial transcript γ of the interaction so far, the strategy f determines an optimal next message for the prover by considering all possible coin tosses of the verifier that are consistent with (x, γ). Specifically, f is determined recursively such that $f(x, \gamma) = m$ if m maximizes the number of verifier coins that are consistent with (x, γ) and lead the verifier to accept when subsequent prover moves are determined by f (which is where recursion is used). That is, coins r support the setting $f(x, \gamma) = m$, where $\gamma = (\alpha_1, \beta_1, ..., \alpha_t, \beta_t)$, if the following two conditions hold:

1. r is consistent with (x, γ), which means that $\beta_i = V(x, r, \alpha_1, ..., \alpha_i)$ for $i = 1, ..., t$.
2. r leads V to accept (when subsequent prover moves are determined by f), which means that $V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_T) = 1$, where $\alpha_{i+1} = f(x, \gamma, m, \beta_{t+1}, ..., \alpha_i \beta_i)$ and $\beta_i = V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_i)$ for $i = t+1, ..., T-1$.

That is, $f(x, \gamma) = m$ if m maximizes the value of $\mathsf{E}[f(x, \gamma, m, V(x, R_\gamma, m))]$, where R_γ is selected uniformly among the r's that are consistent with (x, γ). Thus, $f(x, \gamma)$ can be computed in polynomial-space when given oracle access to $f(x, \gamma, \cdot, \cdot)$, and the proposition follows by standard composition of space-bounded computations. □

9.1.3 Variants and finer structure: an overview

In this subsection we consider several variants on the basic definition of interactive proofs as well as finer complexity measures. This is an advanced subsection, which only provides an overview of the various notions and results (as well as pointers to proofs of the latter).

9.1.3.1 Arthur-Merlin games a.k.a public-coin proof systems

The verifier's messages in a general interactive proof system are determined arbitrarily (but efficiently) based on its view of the interaction so far (which includes its internal coin tosses, which without loss of generality can take place at the onset of the interaction). In public-coin proof systems (a.k.a Arthur-Merlin proof systems), the verifier's messages are the outcome of any coin it tosses at the current round. Assuming, without loss of generality, that the verifier tosses the same number ℓ of coins in each round, we may assume that the verifier's message in each round consists of the outcome of ℓ coin tosses (because any other string that the verifier may compute is determined by the coins tossed so far). Note that the proof systems presented in the proof of Theorem 9.4 are of the public-coin type, whereas this is not the case for the Graph Non-Isomorphism proof system (of Construction 9.3). Thus, although not all natural proof systems are of the public-coin type, every set having an interactive proof system also has a public-coin interactive proof system. This means that, *in the context of interactive proof systems, asking random questions is as powerful as asking clever questions.*

Indeed, public-coin proof systems are a syntactically restricted type of interactive proof systems. This restriction may make the design of such systems more complex, but potentially facilitates their analysis (and especially the analysis of a generic system). Another advantage of public-coin proof systems is that the verifier's actions (except for its final decision) are oblivious of the prover's messages. This property is used in the proof of Theorem 9.12.

9.1.3.2 Interactive proof systems with two-sided error

In Definition 9.1 error probability is allowed in the soundness condition but not in the completeness condition. In such a case, we say that the proof system has perfect completeness (or one-sided error probability). A more general definition allows an error probability (upper-bounded by, say, 1/3) in both the completeness and soundness conditions. Note that sets having such generalized (two-sided error) interactive proofs are also in $\mathcal{PSPACE}$, and thus allowing two-sided error does not increase the power of interactive proofs. See further discussion at the end of §9.1.3.3.

9.1.3.3 A hierarchy of interactive proof systems

Definition 9.1 only refers to the total computation time of the verifier, and thus allows an arbitrary (polynomial) number of messages to be exchanged. A finer definition refers to the number of messages being exchanged (also called the number of rounds).[6]

[6] An even finer structure emerges when considering also the total length of the messages sent by the prover (see [102]).

Definition 9.6 (The round-complexity of interactive proof):

- *For an integer function m, the complexity class $\mathcal{IP}(m)$ consists of sets having an interactive proof system in which, on common input x, at most $m(|x|)$ messages are exchanged between the parties.*[7]

- *For a set of integer functions, M, we let $\mathcal{IP}(M) \stackrel{\text{def}}{=} \bigcup_{m \in M} \mathcal{IP}(m)$. Thus, $\mathcal{IP} = \mathcal{IP}(\texttt{poly})$.*

For example, interactive proof systems in which the verifier sends a single message that is answered by a single message of the prover corresponds to $\mathcal{IP}(2)$. Clearly, $\mathcal{NP} \subseteq \mathcal{IP}(1)$, yet the inclusion may be strict because in $\mathcal{IP}(1)$ the verifier may toss coins after receiving the prover's single message. (Also note that $\mathcal{IP}(0) = \text{co}\mathcal{RP}$.) Concerning the finer structure of the IP-hierarchy, the following is known:

- A linear speed-up (see Section F.2 (or [20] and [107])): For every integer function, f, such that $f(n) \geq 2$ for all n, the class $\mathcal{IP}(O(f(\cdot)))$ collapses to the class $\mathcal{IP}(f(\cdot))$. In particular, $\mathcal{IP}(O(1))$ collapses to $\mathcal{IP}(2)$.

- The class $\mathcal{IP}(2)$ contains sets not known to be in $\mathcal{NP}$; e.g., Graph Non-Isomorphism (see Construction 9.3). However, under plausible intractability assumptions, $\mathcal{IP}(2) = \mathcal{NP}$ (see [155]).

- If $\text{co}\mathcal{NP} \subseteq \mathcal{IP}(2)$ then the Polynomial-Time Hierarchy collapses (see [42]).

It is conjectured that $\text{co}\mathcal{NP}$ is *not* contained in $\mathcal{IP}(2)$, and consequently that interactive proofs with an unbounded number of message exchanges are more powerful than interactive proofs in which only a bounded (i.e., constant) number of messages are exchanged.[8] The class $\mathcal{IP}(1)$ (also denoted $\mathcal{MA}$) seems to be *the* "real" randomized (and yet non-interactive) version of $\mathcal{NP}$: Here the prover supplies a candidate (polynomial-size) "proof", and the verifier assesses its validity probabilistically (rather than deterministically).

The IP-hierarchy (i.e., $\mathcal{IP}(\cdot)$) equals an analogous hierarchy, denoted $\mathcal{AM}(\cdot)$, that refers to public-coin (a.k.a Arthur-Merlin) interactive proofs. That is, for every integer function f, it holds that $\mathcal{AM}(f) = \mathcal{IP}(f)$. For $f \geq 2$, it is also the case that $\mathcal{AM}(f) = \mathcal{AM}(O(f))$; actually, the aforementioned linear speed-up for $\mathcal{IP}(\cdot)$ is established by combining the following two results:

1. Emulating $\mathcal{IP}(\cdot)$ by $\mathcal{AM}(\cdot)$ (see §F.2.1 or [107]): $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$.

2. Linear speed-up for $\mathcal{AM}(\cdot)$ (see §F.2.2 or [20]): $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$.

In particular, $\mathcal{IP}(O(1)) = \mathcal{AM}(2)$, even if $\mathcal{AM}(2)$ is restricted such that the verifier tosses no coins after receiving the prover's message. (Note that $\mathcal{IP}(1) = \mathcal{AM}(1)$ and $\mathcal{IP}(0) = \mathcal{AM}(0)$ are trivial.) We comment that it is common to denote $\mathcal{AM}(2)$

[7] We count the total number of messages exchanged regardless of the direction of communication.

[8] Note that the linear speed-up cannot be applied for an unbounded number of times, because each application may increase (e.g., square) the time-complexity of verification.

by $\mathcal{AM}$, which is indeed inconsistent with the convention of using $\mathcal{IP}$ to denote $\mathcal{IP}(\texttt{poly})$.

The fact that $\mathcal{IP}(O(f)) = \mathcal{IP}(f)$ is proved by establishing an analogous result for $\mathcal{AM}(\cdot)$ demonstrates the advantage of the public-coin setting for the study of interactive proofs. A similar phenomenon occurs when establishing that the IP-hierarchy equals an analogous two-sided error hierarchy (see Exercise 9.5).

9.1.3.4 Something completely different

We stress that although we have relaxed the requirements from the verification procedure, by allowing it to interact with the prover, toss coins, and risk some (bounded) error probability, we did not restrict the validity of its assertions by assumptions concerning the potential prover. This should be contrasted with other notions of proof systems, such as computationally-sound ones (see §9.1.4.2), in which the validity of the verifier's assertions depends on assumptions concerning the prover(s).

9.1.4 On computationally bounded provers: an overview

Recall that our definition of interactive proofs (i.e., Definition 9.1) makes no reference to the computational abilities of the potential prover. This fact has two conflicting consequences:

1. The completeness condition does not provide any upper bound on the complexity of the corresponding proving strategy (which convinces the verifier to accept valid assertions).

2. The soundness condition guarantees that, regardless of the computational effort spend by a cheating prover, the verifier cannot be fooled to accept invalid assertions (with probability exceeding the soundness error).

Note that providing an upper-bound on the complexity of the (presecribed) prover strategy P of a specific interactive proof system (P, V) only strengthens the claim that (P, V) is a proof system for the corresponding set (of valid assertions). We stress that the presecribed prover strategy is referred to only in the completeness condition (and is irrelevant to the soundness condition). On the other hand, relaxing the definition of interactive proofs such that soundness holds only for a specific class of cheating prover strategies (rather than for all cheating prover strategies) weakens the corresponding claim. In this advanced section we consider both possibilities.

Teaching note: Indeed, this is an advanced subsection, which is best left for independent reading. It merely provides an overview of the various notions, and the reader is directed to the chapter's notes for further detail (i.e., pointers to the relevant literature).

9.1.4.1 How powerful should the prover be?

Assume that a set S is in $\mathcal{IP}$. This means that there is a verifier V that can be convinced to accept any input in S but cannot be fooled to accept any input not in S (except with small probability). One may ask how powerful should a prover be such that it can convince the verifier V to accept any input in S. Note that Proposition 9.5 asserts that an optimal prover strategy can be implemented in polynomial-space (and that we cannot expect better for a generic set in $\mathcal{PSPACE} = \mathcal{IP}$), but we will seek better upper-bounds on the complexity of the prover that convinces a specific verifier (which in turn corresponds to a specific set S). More interestingly, considering all possible verifiers that give rise to interactive proof systems for S, we ask what is the minimum power required from a prover that satisfies the completeness requirement with respect to one of these verifiers?

We stress that, unlike the case of computationally-sound proof systems (see §9.1.4.2), we do not restrict the power of the prover in the soundness condition, but rather consider the minimum complexity of provers meeting the completeness condition. Specifically, we are interested in *relatively efficient* provers that meet the completeness condition. The term "relatively efficient prover" has been given three different interpretations.

1. A prover is considered *relatively efficient* if, when given an auxiliary input (in addition to the common input in S), it works in (probabilistic) polynomial-time. Specifically, in case $S \in \mathcal{NP}$, the auxiliary input maybe an NP-proof that the common input is in the set. Still, even in this case the interactive proof need not consist of the prover sending the auxiliary input to the verifier; for example, an alternative procedure may allow the prover to be zero-knowledge (see Construction 9.10).

   This interpretation is adequate and in fact crucial for applications in which such an auxiliary input is available to the otherwise polynomial-time parties. Typically, such auxiliary input is available in cryptographic applications in which parties wish to prove in (zero-knowledge) that they have correctly conducted some computation. In these cases the NP-proof is just the transcript of the computation by which the claimed result has been generated, and thus the auxiliary input is available to the proving party.

2. A prover is considered *relatively efficient* if it can be implemented by a probabilistic polynomial-time oracle machine with oracle access to the set S itself. (Note that the prover in Construction 9.3 has this property.)

   This interpretation generalizes the notion of self-reducibility of NP-sets. (Recall that by self-reducibility of an NP-set we mean that the search problem of finding an NP-witness is polynomial-time reducible to deciding membership in the set (cf. Definition 2.13).)

3. A prover is considered *relatively efficient* if it can be implemented by a probabilistic machine that runs in time that is polynomial in the deterministic complexity of the set. This interpretation relates the difficulty of convincing a "lazy verifier" to the complexity of finding the truth alone.

Hence, in contrast to the first interpretation, which is adequate in settings where assertions are generated along with their NP-proofs, the current interpretation is adequate in settings in which the prover is given only the assertion and has to find a proof to it by itself (before trying to convince a lazy verifier of its validity).

9.1.4.2 Computational-soundness

Relaxing the soundness condition such that it only refers to relatively-efficient ways of trying to fool the verifier (rather than to all possible ways) yields a fundamentally different notion of a proof system. Assertions proven in such a system are not necessarily correct; they are correct only if the potential cheating prover does not exceed the presumed complexity limits. As in §9.1.4.1, the notion of "relative efficiency" can be given different interpretations, the most popular one being that the cheating prover strategy can be described by a (non-uniform) family of polynomial-size circuits. The latter interpretation coincides with the first interpretation used in §9.1.4.1 (i.e., a probabilistic polynomial-time strategy that is given an auxiliary input (of polynomial length)). Specifically, the soundness condition is replaced by the following computational soundness condition that asserts that it is infeasible to fool the verifier into accepting false statements:

> For every prover strategy that is implementable by a family of polynomial-size circuits $\{C_n\}$, and every sufficiently long $x \in \{0,1\}^* \setminus S$, the probability that V accepts x when interacting with $C_{|x|}$ is less than $1/2$.

As in case of standard soundness, the computational-soundness error can be reduced by repetitions. We warn, however, that unlike in the case of standard soundness (where both sequential and parallel repetitions will do), the computational-soundness error cannot always be reduced by parallel repetitions.

It is common and natural to consider proof systems in which the prover strategies considered both in the completeness and soundness conditions satisfy the same notion of relative efficiency. Protocols that satisfy these conditions with respect to the aforementioned interpretation are called arguments. We mention that argument systems may be more efficient (e.g., in terms of their communication complexity) than interactive proof systems.

9.2 Zero-Knowledge Proof Systems

Zero-Knowledge proofs are fascinating and extremely useful constructs. Their fascinating nature is due to their seemingly contradictory definition: zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proven. Their applicability in the domain of cryptography is vast; they are typically used to force malicious parties to behave according to a predetermined protocol. In addition to their direct applicability in Cryptography, zero-knowledge proofs serve as a good bench-mark for the study of various problems regarding cryptographic protocols. In this section we focus on the conceptual contents of zero-knowledge, and relegate their cryptographic applications to Appendix C.

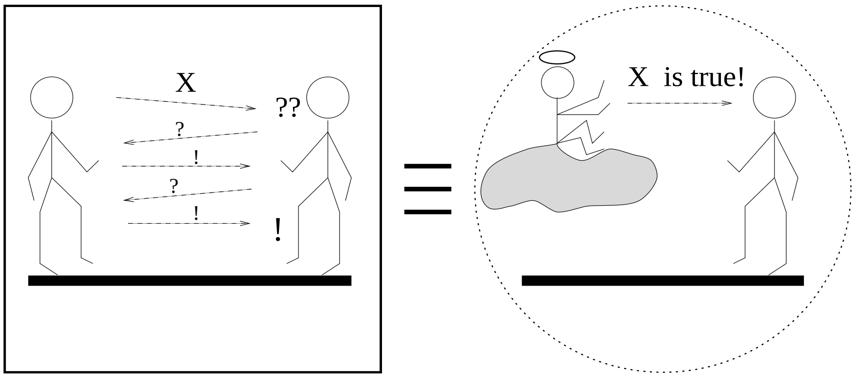


Figure 9.1: Zero-knowledge proofs – an illustration.

Turning back to the conceptual angle, we highlight the fact that standard proofs are believed to yield knowledge and not merely establish the validity of the assertion being proven. Indeed, it is commonly believed that (good) proofs provide a deeper understanding of the theorem being proved. At the technical level, an NP-proof of membership in some set $S \in \mathcal{NP} \setminus \mathcal{P}$ yields something (i.e., the NP-proof itself) that is typically hard to compute (even when assuming that the input is in S). For example, a 3-coloring of a graph is an NP-proof that the graph is 3-colorable, but it yields information (i.e., the coloring) that is infeasible to compute (when given an arbitrary 3-colorable graph). In contrast to such NP-proofs, which seem to yield a lot of knowledge, zero-knowledge proofs yield no knowledge at all; that is, the latter exhibit an extreme contrast between being convincing (of the validity of a statement) and teaching anything on top of the validity of the statement.

Teaching note: We believe that the treatment of zero-knowledge proofs provided in this section suffices for the purpose of a course in complexity theory. For an extensive treatment of (computational) zero-knowledge proofs, the interested reader is referred to [87, Chap. 4].

9.2.1 Definitional Issues

Loosely speaking, zero-knowledge proofs are proofs that yield nothing beyond the validity of the assertion; that is, a verifier obtaining such a proof only gains conviction in the validity of the assertion. This is formulated by saying that anything that can be feasibly obtained from a zero-knowledge proof is also feasibly computable from the (valid) assertion itself. The latter formulation follows the simulation paradigm, which is discussed next.

9.2.1.1 A wider perspective: the simulation paradigm

In defining zero-knowledge proofs, we view the verifier as a potential adversary that tries to gain knowledge from the (prescribed) prover.[9] We wish to state that no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). Let us consider the desired formulation from a wide perspective.

A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can be obtained within essentially the same computational effort by a benign behavior. The definition of the "benign behavior" captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. For example, in the previous paragraph, we said that a proof is zero-knowledge if it yields nothing beyond the validity of the assertion (i.e., the benign behavior is any computation that is based (only) on the assertion itself, while assuming that the latter is valid). Thus, in a zero-knowledge proof no feasible adversarial strategy for the verifier can obtain more than a "benign verifier" (which believes the assertion) can obtain from the assertion itself. We comment that the simulation paradigm is pivotal to many definitions in cryptography (e.g., it underlies the definition of security of encryption schemes and cryptographic protocols); for further details see Appendix C.

9.2.1.2 The basic definitions

Zero-knowledge is a property of some prover strategies. More generally, zero-knowledge is a property of some interactive machines. Fixing an interactive machine (e.g., a prescribed prover), we consider what can be gained (i.e., computed) by an *arbitrary feasible adversary* (e.g., a verifier) *that interacts with the aforementioned fixed machine* on a common input taken from a predetermined set (in our case the set of valid assertions). This gain is compared against what can be computed by an *arbitrary feasible algorithm* (called a simulator) that is only given the input itself. The fixed machine is zero-knowledge if the "computational power" of these two (fundamentally different settings) is essentially equivalent. Details follow.

The formulation of the zero-knowledge condition considers two ensembles of probability distributions, each ensemble associates a probability distribution to each valid assertion. Specifically, in the case of interactive proofs, the first ensemble represents the output distribution of the verifier after interacting with the specified prover strategy P, where the verifier is employing an arbitrary efficient strategy (not necessarily the specified one). The second ensemble represents the output distribution of some probabilistic polynomial-time algorithm (which does not interact with anyone). The basic paradigm of zero-knowledge asserts that for every ensemble of the first type there exist a "similar" ensemble of the second type.

[9] Recall that when defining a proof system (e.g., an interactive proof system), we view the prover as a potential adversary that tries to fool the (prescribed) verifier (into accepting invalid assertions).

The specific variants differ by the interpretation given to the notion of *similarity*. The most strict interpretation, leading to perfect zero-knowledge, is that similarity means equality.

Definition 9.7 (perfect zero-knowledge, over-simplified):[10] *A prover strategy, P, is said to be* perfect zero-knowledge over a set S *if for every probabilistic polynomial-time verifier strategy, V^*, there exists a probabilistic polynomial-time algorithm, M^*, such that*

$$(P,V^*)(x) \equiv M^*(x)\,, \qquad \textit{for every } x \in S$$

where $(P,V^*)(x)$ is a random variable representing the output of verifier V^* after interacting with the prover P on common input x, and $M^*(x)$ is a random variable representing the output of machine M^* on input x.

We comment that any set in co$\mathcal{RP}$ has a perfect zero-knowledge proof system in which the prover keeps silence and the verifier decides by itself. The same holds for $\mathcal{BPP}$ provided that we relax the definition of interactive proof system to allow two-sided error. Needless to say, our focus is on non-trivial proof systems; that is, proof systems for sets outside of $\mathcal{BPP}$.

A somewhat more relaxed interpretation (of the notion of similarity), leading to almost-perfect zero-knowledge (a.k.a statistical zero-knowledge), is that similarity means statistical closeness (i.e., negligible difference between the ensembles). The most liberal interpretation, leading to the standard usage of the term zero-knowledge (and sometimes referred to as computational zero-knowledge), is that similarity means computational indistinguishability (i.e., failure of any efficient procedure to tell the two ensembles apart). Combining the foregoing discussion with the relevant definition of computational indistinguishability (i.e., Definition C.5), we obtain the following definition.

Definition 9.8 (zero-knowledge, somewhat simplified): *A prover strategy, P, is said to be* zero-knowledge over a set S *if for every probabilistic polynomial-time verifier strategy, V^*, there exists a probabilistic polynomial-time* simulator, *M^*, such that for every probabilistic polynomial-time* distinguisher*, D, it holds that*

$$d(n) \stackrel{\text{def}}{=} \max_{x\in S\cap\{0,1\}^n} \{|\mathsf{Pr}[D(x,(P,V^*)(x))=1] - \mathsf{Pr}[D(x,M^*(x))=1]|\}$$

is a negligible function.[11] *We denote by $\mathcal{ZK}$ the class of sets having zero-knowledge interactive proof systems.*

Definition 9.8 is a simplified version of the actual definition. Specifically, in order to guarantee that zero-knowledge is preserved under sequential composition it is necessary to slightly augment the definition (by providing V^* and M^* with the same arbitrary auxiliary input of poly($|x|$) length). For details see Section C.4.2. Other definitional issues and related notions are briefly discussed in Section C.4.4.

[10] In the actual definition one either allows M^* to run for *expected* polynomial-time or allows M^* to have no output with probability at most 1/2. The latter alternative implies the former, but the converse is not known to hold.

[11] That is, d vanishes faster that the reciprocal of any positive polynomial (i.e., for every positive polynomial p and for sufficiently large n, it holds that $d(n) < 1/p(n)$).

On the role of randomness and interaction. It can be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which the verifier is deterministic. The same holds for deterministic provers, provided we consider "auxiliary-input" zero-knowledge (as in Definition C.9). It can also be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which a single message is sent. Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For details, see [87, Sec. 4.5.1].)

Advanced Comment: Knowledge Complexity. Zero-knowledge is the lowest level of a knowledge-complexity hierarchy that quantifies the "knowledge revealed in an interaction." Specifically, the knowledge complexity of an interactive proof system may be defined as the minimum number of oracle-queries required in order to efficiently simulate an interaction with the prover. (See [86, Sec. 2.3.1] for references.)

9.2.2 The Power of Zero-Knowledge

When faced with a definition as complex (and seemingly self-contradictory) as the definition of zero-knowledge, one should indeed wonder whether the definition can be met (in a non-trivial manner).[12] It turns out that the existence of non-trivial zero-knowledge proofs is related to the existence of intractable problems in $\mathcal{NP}$. In particular, we will show that if one-way functions exist then every NP-set has a zero-knowledge proof system. (For the converse, see [87, Sec. 4.5.2] or [212].) We first demonstrate the scope of zero-knowledge by a presenting a simple (perfect) zero-knowledge proof system for a specific NP-set that is not known to be in $\mathcal{BPP}$. In this case we make no intractability assumptions, but the result is significant only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$.

9.2.2.1 A simple example

> *A story not found in the Odyssey refers to the not so famous Labyrinth of the Island of Aeaea. The Sorceress Circe, daughter of Helius, challenged godlike Odysseus to traverse the Labyrinth from its North Gate to its South Gate. Canny Odysseus doubted whether such a path existed at all and asked beautiful Circe for a proof, to which she replied that if she showed him a path this would trivialize for him the challenge of traversing the Labyrinth. "Not necessarily," clever Odysseus replied, "you can use your magic to transport me to a random place in the labyrinth, and then guide me by a random walk to a gate of my choice. If we repeat this enough times then I'll be convinced that there is a labyrinth-path between the two gates, while you will not reveal to me such a path." "Indeed," wise Circe thought to herself, "showing this mortal a random path from a random location in the labyrinth to*

[12] Note that any set in $\mathcal{BPP}$ has a trivial zero-knowledge (two-sided) proof system in which the verifier just determines membership by itself.

the gate he chooses will not teach him more than his taking a random walk from that gate."

The foregoing story illustrates the main idea underlying the zero-knowledge proof for Graph Isomorphism presented next. Recall that the set of pairs of isomorphic graphs is not known to be in $\mathcal{BPP}$, and thus the straightforward NP-proof system (in which the prover just suplies the isomorphism) may not be zero-knowledge. Furthermore, assuming that Graph Isomorphism is not in $\mathcal{BPP}$, this set has no zero-knowledge NP-proofs of membership, but as we shall see it has zero-knowledge interactive proofs.

Construction 9.9 (zero-knowledge proof for Graph Isomorphism):

- Common Input: *A pair of graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$*. Let* ϕ *be an isomorphism between the input graphs; namely,* ϕ *is a 1-1 and onto mapping of the vertex set* V_1 *to the vertex set* V_2 *such that* $\{u, v\} \in E_1$ *if and only if* $\{\phi(v), \phi(u)\} \in E_2$.

- Prover's first Step (P1): *The prover selects a random isomorphic copy of* G_2*, and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation* π *from the set of permutations over the vertex set* V_2*, and constructs a graph with vertex set* V_2 *and edge set*

  $$E \stackrel{\text{def}}{=} \{\{\pi(u), \pi(v)\} : \{u, v\} \in E_2\} .$$

  The prover sends (V_2, E) *to the verifier.*

- Motivating Remark: *If the input graphs are isomorphic, as the prover claims, then the graph sent in Step P1 is isomorphic to both input graphs. However, if the input graphs are* not *isomorphic then no graph can be isomorphic to both of them.*

- Verifier's first Step (V1): *Upon receiving a graph,* $G' = (V', E')$*, from the prover, the verifier asks the prover to show an isomorphism between* G' *and one of the input graphs, chosen at random by the verifier. Namely, the verifier uniformly selects* $\sigma \in \{1, 2\}$*, and sends it to the prover (who is supposed to answer with an isomorphism between* G_σ *and* G'*).*

- Prover's second Step (P2): *If the message,* σ*, received from the verifier equals 2 then the prover sends* π *to the verifier. Otherwise (i.e.,* $\sigma \neq 2$*), the prover sends* $\pi \circ \phi$ *(i.e., the composition of* π *on* ϕ*, defined as* $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$*) to the verifier.*

  (Indeed, the prover treats any $\sigma \neq 2$ *as* $\sigma = 1$*. In the analysis we shall assume, without loss of generality, that* $\sigma \in \{1, 2\}$ *always holds.)*

- Verifier's second Step (V2): *If the message, denoted* ψ*, received from the prover is an isomorphism between* G_σ *and* G' *then the verifier outputs 1, otherwise it outputs 0.*

The verifier strategy in Construction 9.9 is easily implemented in probabilistic polynomial-time. In case the prover is given an isomorphism between the input graphs as auxiliary input, also the prover's program can be implemented in probabilistic polynomial-time. The motivating remark justifies the claim that Construction 9.9 constitutes an interactive proof system for the set of pairs of isomorphic graphs. As for the zero-knowledge property, consider first the special case in which the verifier actually follows the prescribed strategy (and selects σ at random, and in particular obliviously of the graph G' it receives). The view of this verifier can be easily simulated by selecting σ and ψ at random, constructing G' as a random isomorphic copy of G_σ (via the isomorphism ψ), and outputting the triplet (G', σ, ψ). Indeed (even in this case), the simulator behaves differently from the prescribed prover (which selects G' as a random isomorphic copy of G_2, via the isomorphism π), but its output distribution is identical to the verifier's view in the real interaction. However, the forgoing description assumes that the verifier follows the prescribed strategy, while in general the verifier may select σ depending on the graph G'. Thus, a slightly more complicated simulation (described next) is required.

A general clarification may be in place. Recall that we wish to simulate the interaction of an arbitrary verifier strategy with the prescribed prover, and so the simulator must depend on the verifier strategy. Indeed, we shall describe the simulator while referring to such a generic verifier strategy. Formally this means that the simulator's program incorporates the aforementioned verifier strategy. (Actually, the following simulator uses the generic verifier strategy as a subroutine.)

Turning back to the specific protocol of Construction 9.9, the basic idea is that simulator tries to guess σ and can complete a simulation if its guess turns out to be correct. Specifically, the simulator selects $\tau \in \{1, 2\}$ uniformly (hoping that the verifier selects $\sigma = \tau$), and constructs G' by randomly permuting G_τ (and thus being able to present an isomorphism between G_τ and G'). Recall that the simulator is analyzed only on yes-instances (i.e., the input graphs G_1 and G_2 are isomorphic). The point is that if G_1 and G_2 are isomorphic, then the graph G' does not yield any information regarding the simulator's guess (i.e., τ), and thus this guess is correct with probability $1/2$. (Indeed, this is analogous to the analysis of the soundness of Construction 9.3.) If the guess is correct then the simulator can produce an output that has the correct distribution, otherwise the entire process is repeated.

Useful conventions. We wish to highlight three conventions that were either used (implicitly) in the foregoing analysis or can be used to simplify the description of (this and/or) other zero-knowledge simulators.

1. Without loss of generality, we may assume that the cheating verifier strategy is implemented by a *deterministic* polynomial-size circuit (or, equivalently, by a polynomial-time algorithm with an auxiliary input).[13]

[13] This observation is not crucial, but it does simplify the analysis (by eliminating the need to specify a sequence of coin tosses in each invocation of the verifier's strategy).

This is justified by fixing any outcome of the verifier's coins, and observing that our (uniform) simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.

2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e., the common input, their internal coin tosses, and messages that they received). In other words, it suffices to simulate the view of that cheating verifiers have of the real interaction.

   This is justified by noting that the final output of any verifier can be obtained from its view of the interaction, where the complexity of the transformation is upper-bounded by the complexity of the verifier's strategy.

3. Without loss of generality, it suffices to construct a "weak simulator" that produces output with some noticeable[14] probability such that whenever an output is produced it is distributed "correctly" (i.e., similarly to the distribution occuring in real interactions with the prescribed prover).

   This is justified by repeatedly invoking such a weak simulator (polynomially) many times and using the first output produced by any of these invocations. Note that by using an adequate number of invocations, we fail to produce an output with negligible probability. (Alternatively, we always produce an output after an expected polynomial number of invocations.) Furthermore, note that a simulator that fails to produce output with negligible probability can be converted to a simulator that always produces an output, while incurring a negligible statistic deviation in the output distribution.

9.2.2.2 The full power of zero-knowledge proofs

The forgoing zero-knowledge proof system refers to one NP-set that is not known to be in $\mathcal{BPP}$. It turns out that, under reasonable assumptions, zero-knowledge can be used to prove membership in *any* NP-set. Intuitively, it suffices to establish this fact for a single NP-complete set, and thus we focus on presenting a zero-knowledge proof system for the set of 3-colorable graphs. Note that it is easy to prove that a given graph G is 3-colorable by just presenting a 3-coloring of G (and the same holds for membership in any set in $\mathcal{NP}$), but this NP-proof is not a zero-knowledge proof (unless $\mathcal{P} = \mathcal{NP}$). In fact, assuming $\mathcal{P} \neq \mathcal{NP}$, graph 3-colorability has no zero-knowledge NP-proofs, but as we shall see it has zero-knowledg interactive proofs. This interactive proof will be described while referring to "boxes" in which information can be hidden and later revealed. Such boxes can be implemented using one-way functions (see below).

Construction 9.10 (Zero-knowledge proof of 3-colorability, abstract description): *The description refers to abstract non-transparent boxes that can be perfectly locked and unlocked such that these boxes perfectly hide their contents while being locked.*

[14] Recall that a probability is called noticeable if it is greater than the reciprocal of some positive polynomial (in the relevant parameter).

- Common Input: *A simple graph* $G=(V,E)$.
- Prover's first step: *Let* ψ *be a 3-coloring of* G. *The prover selects a random permutation,* π, *over* $\{1,2,3\}$, *and sets* $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, *for each* $v \in V$. *Hence, the prover forms a random relabeling of the 3-coloring* ψ. *The prover sends the verifier a sequence of* $|V|$ *locked and non-transparent boxes such that the* v^{th} *box contains the value* $\phi(v)$.
- Verifier's first step: *The verifier uniformly selects an edge* $\{u,v\} \in E$, *and sends it to the prover.*
- Motivating Remark: *The boxes are supposed to contain a 3-coloring of the graph, and the verifier asks to inspect the colors of vertices* u *and* v. *Indeed, for the zero-knowledge condition, it is crucial that the prover only responds to pairs that correspond to edges of the graph.*
- Prover's second step: *Upon receiving an edge* $\{u,v\} \in E$, *the prover sends to the verifier the keys to boxes* u *and* v.

  For simplicity of the analysis, if the verifier sends $\{u,v\} \notin E$ *then the prover behaves as if it has received a fixed* (or random) *edge in* E, *rather than suspending the interaction, which would have been the natural thing to do.*
- Verifier's second step: *The verifier unlocks and opens boxes* u *and* v, *and accepts if and only if they contain two different elements in* $\{1,2,3\}$.

The verifier strategy in Construction 9.10 is easily implemented in probabilistic polynomial-time. The same holds with respect to the prover's strategy, provided that it is given a 3-coloring of G as auxiliary input. Clearly, if the input graph is 3-colorable then the verfier accepts with probability 1 when interacting with the prescribed prover. On the other hand, if the input graph is not 3-colorable, then any contents put in the boxes must be invalid on at least one edge, and consequently the verifier will reject with probability at least $\frac{1}{|E|}$. Hence, the foregoing protocol exhibits a non-negligible gap in the accepting probabilities between the case of 3-colorable graphs and the case of non-3-colorable graphs. To increase the gap, the game may be repeated sufficiently many times (of course, using independent coin tosses in each repetition).

In the abstract setting of Construction 9.10, the zero-knowledge property follows easily, because one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. This indeed demonstrates that the verifier learns nothing from the interaction, because it expects to see a random pair of different colors (and indeed this is what it sees). Note that the aforementioned expectation relies on the fact that the boxes correspond to vertices that are connected by an edge.

This simple demonstration of the zero-knowledge property is not possible in the digital implementation (discussed next), because the boxes are not totally unaffected by their contents (but are rather effected, yet in an indistinguishable manner). Instead, we simulate the interaction as follows. We first guess (at random) which pair of boxes (rorresponding to an edge) the verifier would ask to open, and place a random pair of distinct colors in these boxes (and garbage in the rest).[15] We hand all boxes to the verifier. In case the verifier asks for the chosen pair (i.e., the one we guessed), we can complete the simulation. Otherwise, we try again (with a new random guess). Thus, it suffices to use boxes that hide their contents almost perfectly (rather than being perfectly opaque). Such boxes can be implemented digitally.

Teaching note: Indeed, we recommend presenting and analyzing in class only the foregoing abstract protocol. It suffices to briefly comment about the digital implementation, rather than presenting a formal proof of Theorem 9.11 (which can be found in [96] (or [87, Sec. 4.4])).

Digital implementation. We implement the abstract boxes (referred to in Construction 9.10) by using (adequately defined) commitment schemes. Loosely speaking, such a scheme is a two-phase game between a sender and a receiver such that after the first phase the sender is "committed" to a value and yet, at this stage, it is infeasible for the receiver to find out the committed value (i.e., the commitment is "hiding"). The committed value will be revealed to the receiver in the second phase and it is guaranteed that the sender cannot reveal a value other than the one committed (i.e., the commitment is "binding"). Such commitment schemes can be implemented assuming the existence of one-way functions (as in Definition 7.3).

Zero-knowledge proofs for other NP-sets. Using the fact that 3-colorability is NP-complete, one gets zero-knowledge proofs for any NP-set.[16] Furthermore, NP-witnesses can be efficiently transformed into polynomial-size circuits that implement the corresponding (presecribed zero-knowledge) prover strategies.

Theorem 9.11 (The ZK Theorem): *Assuming the existence of one-way functions, any NP-proof can be efficiently transformed into a* (computational) *zero-knowledge interactive proof. In particular,* $\mathcal{NP} \subseteq \mathcal{ZK}$.

The hypothesis (regarding the existence of one-way functions) in Theorem 9.11 seems unavoidable, because the existence of zero-knowledge proofs for "hard on the average" problems implies the existence of one-way functions (and, likewise, the existence of zero-knowledge proofs for sets outside $\mathcal{BPP}$ implies the existence of "auxiliary-input one-way functions"). Theorem 9.11 has a dramatic effect on the design of cryptographic protocols (see Appendix C). In a different vein we mention that, under the same assumption, any interactive proof can be transformed into a zero-knowledge one. (This transformation, however, is not efficient.)

[15] An alternative (and more efficient) simulation consists of putting random independent colors in the various boxes, hoping that the verifier asks for an edge that is properly colored. The latter event occurs with probability (approximately) 2/3, provided that the boxes hide their contents (almost) perfectly.

[16] Actually, we should either rely on the fact that the standard Karp-reductions are invertible in polynomial time or on the fact that the 3-colorability protocol is actually zero-knowledge with respect to auxiliary inputs (as in Definition C.9).

Theorem 9.12 (The ultimate ZK Theorem): *Assuming the existence of one-way functions, $\mathcal{IP} = \mathcal{ZK}$.*

Loosely speaking, Theorem 9.12 can be proved by recalling that $\mathcal{IP} = \mathcal{AM}(\texttt{poly})$ and modifying any public-coin protocol as follows: the modified prover sends commitments to its messages rather than the messages themselves, and once the original interaction is completed it proves (in zero-knowledge) that the corresponding transcript would have been accepted by the original verifier. Indeed, the latter assertion is of the "NP type", and thus the zero-knowledge proof system guaranteed in Theorem 9.11 can be invoked to prove it.

Reflection. The proof of Theorem 9.11 uses the fact that 3-colorability is NP-complete in order to obtain a zero-knowledge proofs for any set in $\mathcal{NP}$ by using such a protocol for 3-colorability (i.e., Construction 9.10). Thus, an NP-completeness result is used here in a "positive" way; that is, in order to construct something rather than in order to derive a hardness result. This was probably the first positive application of NP-completeness. Subsequent positive uses of completeness results have appeared in the context of interactive proofs [150, 191], probabilistically checkable proofs [18, 70, 14, 13], and "hardness versus randomness trade-offs" [19].

Perfect and Statistical Zero-Knowledge. The foregoing results may be contrasted with the results regarding the complexity of statistical zero-knowledge proof systems: Statistical zero-knowledge proof systems exist only for sets in $\mathcal{IP}(2) \cap \mathrm{co}\mathcal{IP}(2)$, and thus are unlikely to exist for all NP-sets. On the other hand, the class Statistical Zero-Knowledge is known to contain some hard problems, and turns out to have interesting complexity theoretic properties (e.g., being closed under complementation, and having very natural complete problems). The interested reader is referred to [211].

9.2.3 Proofs of Knowledge – a parenthetical subsection

Teaching note: Technically speaking, this topic belongs to Section 9.1, but the more interesting examples of proof of knowledge are ones that are zero-knowledge.

Loosely speaking, "proofs of knowledge" are interactive proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph), and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn is equivalent to the assertion that the graph is 3-colorable).

What do we mean by saying that a *machine* knows something? Any standard dictionary suggests several meanings for the verb `to know`, but these are typically phrased with reference to the notion of *awareness*, a notion which is certainly inapplicable in the context of machines. Instead, we should look for a *behavioristic* interpretation of the verb `to know`. Indeed, it is reasonable to link knowledge with the ability to do something (e.g., the ability to write down whatever one knows). Hence, we will say that a machine knows a string α if it *can* output the string α. But this seems as total non-sense too: a machine has a well defined output –

either the output equals α or it does not. So what can be meant by saying that *a machine can do something?* Loosely speaking, it may mean that the machine can be *easily modified* so that it does whatever is claimed. More precisely, it may mean that there exists an *efficient* machine that, using the original machine as a black-box (or given its code as an input), outputs whatever is claimed.

So much for defining the *knowledge of machines*. Yet, whatever a machine knows or does not know is its own business. What can be of interest and reference *to the outside* is whatever can be deduced about the knowledge of a machine by interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge).

For sake of simplicity let us consider a concrete question: *how can a machine prove that it knows a 3-coloring of a graph?* An obvious way is just sending the 3-coloring to the verifier. Yet, we claim that applying the protocol in Construction 9.10 (i.e., the zero-knowledge proof system for 3-Colorability) is an alternative way of proving knowledge of a 3-coloring of the graph.

The definition of a *verifier of knowledge of 3-coloring* refers to any possible prover strategy and links the ability to "extract" a 3-coloring (of a given graph) from such a prover to the probability that this prover convinces the verifier. That is, the definition postulates the existence of an efficient universal way of "extracting" a 3-coloring of a given graph by using any prover strategy that convinces this verifier to accept this graph with probability 1 (or, more generally, with some noticeable probability). On the other hand, we should no expect this extrator to obtain much from prover strategies that fail to convince the verifier (or, more generally, convince it with negligible probability). A robust definition should allow a smooth transition between these two extremes (and in particular between provers that convince the verifier with noticeable probability and those that convince it with negligible probability). Such a definition should also support the intuition by which the following strategy of Alice is zero-knowledge: *Alice sends Bob a 3-coloring of a given graph provided that Bob has successfully proved to her that he knows such a coloring.* We stress that the zero-knowledge property of Alice's strategy should hold regardless of the proof-of-knowledge system used for proving Bob's knowledge of a 3-coloring.

Loosely speaking, we say that an interactive machine, V, constitutes a verifier for knowledge of 3-coloring if, for any prover strategy P, the complexity of extracting a 3-coloring of G when using machine P as a "black box"[17] is inversely proportional to the probability that V is convinced by P (to accept the graph G). Namely, the extraction of the 3-coloring is done by an oracle machine, called an extractor, that is given access to a function specifying the behavior P (i.e., the messages it sends in response to particular messages it may receive). We require that the (*expected*) *running time of the extractor, on input G and access to an oracle specifying P's messages, be inversely related* (by a factor polynomial in $|G|$) *to the probability that P convinces V to accept G*. In particular, if P always convinces V to accept G, then the extractor runs in expected polynomial-time. The same holds in case P convinces V to accept with noticeable probability. On the other hand, if P never

[17]Indeed, one may consider also non-black-box extractors.

convinces V to accept, then nothing is required of the extractor. We stress that the latter special cases do not suffice for a satisfactory definition; see discussion in [87, Sec. 4.7.1].

Proofs of knowledge, and in particular zero-knowledge proofs of knowledge, have many applications to the design of cryptographic schemes and cryptographic protocols. These are enabled by the following general result.

Theorem 9.13 (Theorem 9.11, revisited): *Assuming the existence of one-way functions, any NP-relation has a zero-knowledge proof of knowledge* (of corresponding NP-witnesses). *Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given such an NP-witness.*

9.3 Probabilistically Checkable Proof Systems

> **Teaching note:** Probabilistically checkable proof (PCP) systems may be viewed as a restricted type of interactive proof systems in which the prover is memoryless and responds to each verifier message as if it were the first such message. This perspective creates a tighter link with previous sections, but is somewhat contrived. Indeed, such a memoryless prover may be viewed as a static object that the verifier may query at locations of its choice. But then it is more appealing to present the model using the (more traditional) terminology of oracle machines rather than using (and degenerating) the terminology of interactive machines.

Probabilistically checkable proof systems can be viewed as standard (deterministic) proof systems that are augmented with a probabilistic procedure capable of evaluating the validity of the assertion by examining few locations in the alleged proof. In fact, we focus on the latter probabilistic procedure, which is given direct access to the individual bits of the alleged proof (and need not scan it bit-by-bit). Thus, the alleged proof is a string, as in the case of a traditional proof system, but we are interested in probabilistic verification procedures that access only few locations in the proof, and yet are able to make a meaningful probabilistic verdict regarding the validity of the alleged proof. Specifically, the verification procedure should accept any valid proof (with probability 1), but rejects with probability at least 1/2 any alleged proof for a false assertion.

The main complexity measure associated with probabilistically checkable proof systems is indeed their query complexity. Another complexity measure of natural concern is the length of the proofs being employed, which in turn is related to the randomness complexity of the system. The randomness complexity of PCPs plays a key role in numerous applications (e.g., in composing PCP systems as well as when applying PCP systems to derive inapproximability results), and thus we specify this parameter rather than the proof length.

> **Teaching note:** Indeed, PCP systems are most famous for their role in deriving numerous inapproximation results (see Section 9.3.3), but our view is that the latter is merely one extremely important application of the fundamental notion of a PCP system. Our presentation is organized accordingly.

9.3.1 Definition

Loosely speaking, a probabilistically checkable proof system consists of a probabilistic polynomial-time verifier having access to an oracle that represents an alleged proof (in redundant form). Typically, the verifier accesses only few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. As in the case of interactive proof systems, it is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adequate oracle); whereas, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter which oracle is used. The basic definition of the PCP setting is given in Item (1) below. Yet, the complexity measures introduced in Item (2) are of key importance for the subsequent discussions.

Definition 9.14 (Probabilistic Checkable Proofs – PCP):

1. *A* probabilistic checkable proof system (PCP) for a set *S is a probabilistic polynomial-time oracle machine, called* verifier *and denoted V, that satisfies the following two conditions:*

   - Completeness: *For every $x \in S$ there exists an oracle π_x such that, on input x and access to oracle π_x, machine V always accepts x.*
   - Soundness: *For every $x \notin S$ and every oracle π, on input x and access to oracle π, machine V rejects x with probability at least $\frac{1}{2}$.*

2. *We say that a probabilistic checkable proof system has* query complexity $q : \mathbb{N} \to \mathbb{N}$ *if, on any input of length n, the verifier makes at most $q(n)$ oracle queries.*[18] *Similarly, the* randomness complexity $r : \mathbb{N} \to \mathbb{N}$ *upper-bounds the number of coin tosses performed by the verifier on a generic n-bit long input.*

   For integer functions r and q, we denote by $\mathcal{PCP}(r, q)$ the class of sets having probabilistic checkable proof systems of randomness complexity r and query complexity q. For sets of integer functions, R and Q,

$$\mathcal{PCP}(R, Q) \stackrel{\text{def}}{=} \bigcup_{r \in R\,,\, q \in Q} \mathcal{PCP}(r, q)\,.$$

We note that the oracle π_x referred to in the completeness condition a PCP system constitutes a proof in the standard mathematical sense (with respect to a verification procedure that examines all possible outcomes of V's internal coin tosses). Furthermore, the oracles in PCP systems of logarithmic randomness complexity constitute NP-proofs. However, these oracles have the extra remarkable property of enabling a lazy verifier to toss coins, take its chances and "assess" the validity of the proof without reading all of it (but rather by reading a tiny portion of it). Potentially, this allows the verifier to utilize very long proofs (i.e., of super-polynomial length) or alternatively examine very few bits of an NP-proof.

[18] As usual in complexity theory, the oracle answers are always binary (i.e., either 0 or 1).

We note that the error probability (in the soundness condition) of PCP systems can be reduced by successive applications of the proof system. In particular, repeating the process for k times, reduces the probability that the verifier is fooled by a false assertion to 2^{-k}, whereas all complexities increase by at most a factor of k. Thus, PCP systems provide a trade-off between the number of locations examined in the proof and the confidence in the validity of the assertion.

Adaptive versus non-adaptive verifiers. Definition 9.14 allows the verifier to be adaptive; that is, the verifier may determine its queries based on the answers it has received to previous queries (in addition to their dependence on the input and the verifier's internal coin tosses). In contrast, non-adaptive verifiers determine all their queries based solely on their input and internal coin tosses. We comment that most constructions of PCP systems use non-adaptive verifiers, and in fact in many sources PCP systems are defined as non-adaptive.

Randomness versus proof length. Note that the "effective" length of proofs for any PCP system is related to its query and randomness complexities, where the effective length means the number of locations in a generic proof-oracle that may be examined on a fixed input and any possible sequence of internal coin tosses. Specifically, if the PCP system has query complexity q and randomness complexity r then its effective proof length is upper-bounded by 2^{q+r}, whereas a bound of $2^r \cdot q$ holds for non-adaptive systems (see Exercise 9.7). On the other hand, in some sense, the randomness complexity of a PCP can be upper-bounded by the logarithm of the length of the proofs employed (provided we allow non-uniform verifiers; see Exercise 9.9).

On the role of randomness. The PCP Theorem (i.e., $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$) exhibits a trade-off between the number of bits examined in the alleged proof and the confidence in the validity of the assertion. We note that such a trade-off is impossible if one requires the verifier to be deterministic. This is due to the fact that every set in $\mathcal{PCP}(r,q)$ has an NP-proof system that employs proofs of length $2^r q$ (see Exercise 9.8). Thus, $\mathcal{PCP}(r,q) \subseteq \mathrm{DTIME}(2^{2^r q} \cdot \mathrm{poly})$, and, in particular, $\mathcal{PCP}(0, \texttt{log}) = \mathcal{P}$. Furthermore, since it is unlikely that all NP-sets have NP-proof systems that employs proofs of (say) linear length, it follows that $\mathcal{PCP}(r,q)$ is unlikely to contain $\mathcal{NP}$ for $2^{r(n)} q(n) \leq n$ (or for any other fixed polynomial that bounds $2^r q$). Actually, $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{NP}$ is not contained in $\mathcal{PCP}(o(\log), o(\log))$ (see Exercise 9.11).

9.3.2 The Power of Probabilistically Checkable Proofs

The celebrated PCP Theorem asserts that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$, and this result is indeed the focus of the current section. But before getting to it we make several simple observation regarding the PCP Hierarchy.

We first note that $\mathcal{PCP}(\texttt{poly}, 0)$ equals co$\mathcal{RP}$, whereas $\mathcal{PCP}(0, \texttt{poly})$ equals $\mathcal{NP}$. It is easy to prove an upper bound on the non-deterministic time complexity of sets in the PCP hierarchy (see Exercise 9.8):

Proposition 9.15 (upper-bounds on the power of PCPs): *For every polynomially bounded integer function r, it holds that $\mathcal{PCP}(r, \texttt{poly}) \subseteq \text{NTIME}(2^r \cdot \texttt{poly})$. In particular, $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$.*

The focus on PCP systems of logarithmic randomness complexity reflects an interest in PCP systems that utilize proof oracles of polynomial length (see discussion in Section 9.3.1). We stress that such PCP systems (i.e., $\mathcal{PCP}(\texttt{log}, q)$) are NP-proof systems with a (potentially amazing) extra property: the validity of the assertion can be "probabilistically evaluated" by examining a (small) portion (i.e., $q(n)$ bits) of the proof. Thus, for any fixed polynomially bounded function q, a result of the form

$$\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q) \tag{9.5}$$

is interesting (because it applies also to NP-sets having witnesses of length exceeding q), and the smaller q – the better. The PCP Theorem asserts the amazing fact by which q can be made a constant.

Theorem 9.16 (The PCP Theorem): *$\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$.*

Thus, probabilistically checkable proofs in which the verifier tosses only logarithmically many coins and makes only a constant number of queries exist for every set in $\mathcal{NP}$. Furthermore, the proof of Theorem 9.16 is constructive in the sense that it allows to efficiently transform any NP-witness (for an instance of a set in $\mathcal{NP}$) into an oracle that makes the PCP verifier accept (with probability 1). Thus, NP-proofs can be transformed into NP-proofs that offer a trade-off between the portion of the proof being read and the confidence it offers. Specifically, for every $\varepsilon > 0$, if the verifier is willing to tolerate an error probability of ε then it for it examine $O(\log(1/\varepsilon))$ bits of the (transformed) NP-proof. Indeed (as discussed in Section 9.3.1), these bit locations need to be selected at random.

A new characterization of NP: Combining Theorem 9.16 with Proposition 9.15 we obtain the following characterization of $\mathcal{NP}$.

Corollary 9.17 (The PCP characterization of NP): *$\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$.*

The proof of the PCP Theorem: Theorem 9.16 is a culmination of a sequence of remarkable works, each establishing meaningful and increasingly stronger versions of Eq. (9.5). A presentation of the full proof of Theorem 9.16 is beyond the scope of the current work (and is, in our opinion, unsuitable for a basic course in complexity theory). Instead, we present an overview of the original proof (see §9.3.2.2) as well as of an alternative proof (see §9.3.2.3) that was found more than a decade later. We will start, however, by presenting a weaker result that is used in both proofs of Theorem 9.16 and is also of independent interest. This weaker result (see §9.3.2.1) asserts that any NP-set has a PCP system with constant query complexity (albeit with polynomial randomness complexity); that is, $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$.

Teaching note: In our opinion, presenting in class any part of the proof of the PCP Theorem should be given low priority. In particular, presenting the connections between PCP and the complexity of approximation should be given a higher priority. As for relative priorities among the following three subsections, we recommend giving §9.3.2.1 the highest priority, because it offers a direct demonstration of the power of PCPs. As for the two alternative proofs of the PCP Theorem itself, our recommendation depends on the intended goal. On one hand, for the purpose of merely giving a taste of the ideas involved in the proof, we prefer an overview of the original proof (provided in §9.3.2.2). On the other hand, for the purpose of actually providing a full proof, we definitely prefer the new proof (which is only outlined in §9.3.2.3).

9.3.2.1 Proving that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$

The fact that any NP-set has a PCP system with constant query complexity (regardless of its randomness complexity) already testifies to the power of PCP systems. It asserts that *probabilistic verification of proofs is possible by inspecting very few locations in a* (potentially huge) *proof.* Indeed, the PCP systems presented next utilize exponentially long proofs, but they do so while inspecting these proofs at a constant number of (randomly selected) locations.

We start with a brief overview of the construction. It suffices to construct a PCP for proving the satisfiability of a given system of quadratic equations over GF(2), because this problem is NP-complete.[19] The oracle (of this PCP) is supposed to provide the values of all quadratic expressions evaluated at some assignment to the (say n) variables of the system of quadratic equations (given as input). This assignment is supposed to satisfy the latter system. We distinguish two tables in the oracle: The first table corresponding to the (2^n) linear expressions and the second table to the (2^{n^2}) quadratic expressions. Each table is tested for self-consistency (via a "linearity test"), and the two tables are tested to be consistent with each other (via a "matrix-equality" test, which utilizes "self-correction"). Each of these tests utilizes a constant number of Boolean queries, and randomness that is logarithmic in the size of the corresponding table (and is thus $O(n^2)$). Finally, we test (again via self-correction) the value assigned by these tables to an expression obtained by a random linear combination of the system of quadratic equations that is given as input. Details follow.

The starting point. We construct a PCP system for the set of satisfiable quadratic equations over GF(2). The input is a sequence of such equations over the variables $x_1, ..., x_n$, and the proof oracle consist of two parts (or tables), which are supposed to provide information reagrding some satisfying assignment $\alpha = \alpha_1 \cdots \alpha_n$ (also viewed as an n-ary vector over GF(2)). The first part, T_1, is supposed to provide a Hadamard encoding of the said satisfying assignment; that is, entry $\beta \in \mathrm{GF}(2)^n$ in this table is supposed to provide the inner product mod 2 of the n-ary vectors α and β (i.e., $T_1(\beta)$ is supposed to equal $\sum_{i=1}^{n} \beta_i \alpha_i$). The second part, T_2, is supposed to provide all linear combinations of the values of the $\alpha_i \alpha_j$'s;

[19]Here and elsewhere, we denote by GF(2) the 2-element field.

that is, for $\gamma \in \mathrm{GF}(2)^{n^2}$, the value of $T_2(\gamma)$ is supposed to equal $\sum_{i,j} \gamma_{i,j}\alpha_i\alpha_j$. (Indeed T_1 is contained in T_2, because $\sigma^2 = \sigma$ for any $\sigma \in \mathrm{GF}(2)$.) The PCP verifier will uses the two tables to check that the input (i.e., a sequence of quadratic equations) is satisfied by the assignment that is encoded in the two tables. Needless to say, these tables may not be a valid encoding of any n-ary vector (let alone one that satisfies the input), and so the verifier also needs to check that the encoding is (close to being) valid. We will focus on this task first.

Testing the Hadamard Code. Note that T_1 is supposed to encode a linear function; that is, there must be some $\alpha = \alpha_1 \cdots \alpha_n \in \mathrm{GF}(2)^n$ such that $T_1(\beta) = \sum_{i=1}^n \alpha_i\beta_i$ for every $\beta = \beta_1 \cdots \beta_n \in \mathrm{GF}(2)^n$. This can be tested by selecting uniformly $\beta', \beta'' \in \mathrm{GF}(2)^n$ and checking whether $T_1(\beta') + T_1(\beta'') = T_1(\beta' + \beta'')$, where $\beta' + \beta''$ denotes addition of vectors over GF(2). The analysis of this natural tester turns out to be quite complex. Nevertheless, it is indeed the case that any table that is 0.01-far from being linear is rejected with probability at least 0.02 (see Exercise 9.12), where T is ε-far from being linear if T disagrees with any linear function f on more than an ε fraction of the domain (i.e., $\mathsf{Pr}_r[T(r) = f(r)] > \varepsilon$).

By repeating the linearity test for a constant number of times, we may reject each table that is 0.01-far from being a codeword of the Hadamard Code with probability at least 0.99. Thus, using a constant number of queries, the verifier rejects any T_1 that is 0.01-far from being a Hadamard encoding of any $\alpha \in \mathrm{GF}(2)^n$, and likewise rejects any T_2 that is 0.01-far from being a Hadamard encoding of any $\alpha' \in \mathrm{GF}(2)^{n^2}$. We may thus assume that T_1 (resp., T_2) is 0.01-close to the Hadamard encoding of some α (resp., α'). (This does not mean, however, that α' equals the outer produce of α with itself.)

In the rest of the analysis, we fix $\alpha \in \mathrm{GF}(2)^n$ and $\alpha' \in \mathrm{GF}(2)^{n^2}$, and denote the Hadamard encoding of α (resp., α') by $f_\alpha : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (resp., $f_{\alpha'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$). Recall that T_1 (resp., T_2) is 0.01-close to f_α (resp., $f_{\alpha'}$).

Self-correction of the Hadamard Code. Suppose that T is ε-close to a linear function f (i.e., $\mathsf{Pr}_r[T(r) = f(r)] \leq \varepsilon$). Then, we can recover the value of f at any desired point x, by making two (random) queries to T. Specifically, for a uniformly selected $r \in \mathrm{GF}(2)^n$, we use the value $T(x+r) - T(r)$. Note that the probability that we recover the correct value is at least $1 - 2\varepsilon$, because $\mathsf{Pr}_r[T(x+r) - T(r) = f(x+r) - f(r)] \geq 1 - 2\varepsilon$ and $f(x+r) - f(r) = f(x)$ by linearity of f. (Needless to say, for $\varepsilon < 1/4$, the function T cannot be ε-close to two different linear functions.)[20] Thus, assuming that T_1 is 0.01-close to f_α (resp., T_2 is 0.01-close to $f_{\alpha'}$) we may correctly recover (i.e., with error probability 0.02) the value of f_α (resp., $f_{\alpha'}$) at any desired point by making 2 queries to T_1 (resp., T_2).

Checking consistency of f_α and $f_{\alpha'}$. Suppose that we are given access to $f_\alpha : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and $f_{\alpha'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$, where $f_\alpha(\beta) = \sum_i \alpha_i\beta_i$ and

[20]Indeed, this fact follows from the self-correction argument, but a simpler proof merely refers to the fact that the Hadamard code has relative distance 1/2.

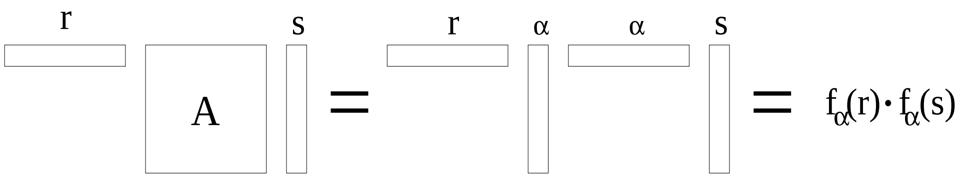


Figure 9.2: Detail for testing consistency of linear and quadratic forms.

$f_{\alpha'}(\beta') = \sum_{i,j} \alpha'_{i,j}\beta'_{i,j}$, and that we wish to verify that $\alpha'_{i,j} = \alpha_i\alpha_j$ for every $i,j \in \{1,...,n\}$. In other words, we are given a (somewhat weird) encoding of two matrices, $A = (\alpha_i\alpha_j)_{i,j}$ and $A' = (\alpha'_{i,j})_{i,j}$, and we wish to check whether or not these matrices are identical. It can be shown (see Exercise 9.14) that if $A \neq A'$ then $\mathsf{Pr}_{r,s}[r^\top As \neq r^\top A's] \geq 1/4$, where r and s are random n-ary vectors. Note that, in our case (see Figure 9.2), $r^\top As = \sum_j(\sum_i r_i\alpha_i\alpha_j)s_j = f_\alpha(r)f_\alpha(s)$ and $r^\top A's = \sum_j(\sum_i r_i\alpha'_{i,j})s_j = f_{\alpha'}(rs^\top)$, where $rs^\top$ is the outer-product of s and r, and so (for $A \neq A'$) we have $\mathsf{Pr}_{r,s}[f_\alpha(r)f_\alpha(s) \neq f_{\alpha'}(rs^\top)] \geq 1/4$. Using self-correction (to obtain the desired value of $f_{\alpha'}$ on $rs^\top$, which is not uniformly distributed in $\mathrm{GF}(2)^{n^2}$), we test the consistency of f_α and $f_{\alpha'}$; that is, we select uniformly $r,s \in \mathrm{GF}(2)^n$ and $R \in \mathrm{GF}(2)^{n^2}$ and check whether or not $T_1(r)T_1(s) = T_2(rs^\top + R) - T_2(R)$.

Checking that α satisfies the quadratic system. Suppose that we are given access to f_α and $f_{\alpha'}$ as above (i.e., in particular, $\alpha' = \alpha\alpha^\top$ is the outer product of α with itself). A key observation is that if α does not satisfy a system of quadratic equations then, with probability 1/2, it does not satisfy a random linear combination of these equations. Thus, in order to check whether α satisfies the quadratic system, we create a single quadratic equation (by taking such a random linear combination) and compare the value of the resulting quadratic expression to the corresponding value, by recovering the value of $f_{\alpha'}$ at a single point (which corresponds to the quadratic equation). That is, to test whether α satisfies the quadratic equation $Q(x) = \sigma$, we test whether $f_{\alpha'}(Q) = \sigma$. The actual checking is implemented by the verifier using self-correction (of the table T_2).

To summarize, the verifier performs a constant number of queries and uses randomness that is quadratic in the number of variables. If the quadratic system is satisfiable (by some α), then the verifier accepts the corresponding tables T_1 and T_2 (i.e., $T_1 = f_\alpha$ and $T_2 = f_{\alpha\alpha^\top}$) with probability 1. On the other hand, if the quadratic system is unsatisfiable, then any pair of tables (T_1, T_2) will be rejected with constant probability (by one of the foregoing tests). It follows that $\mathcal{NP} \subseteq \mathcal{PCP}(r, O(1))$, where $r(n) = O(n^2)$.

9.3.2.2 Overview of the first proof of the PCP Theorem

The proof of the PCP Theorem (Theorem 9.16) is one of the most complicated proofs in the Theory of Computation. The original proof consists of three main conceptual steps, which we roughly sketch first and further discuss later.

1. Constructing a (non-adaptive) PCP system for $\mathcal{NP}$ having *logarithmic randomness and polylogarithmic query complexity*. Furthermore, this proof system has additional properties that enable proof composition as in Step (3) below.

2. Constructing a PCP system for $\mathcal{NP}$ having *polynomial randomness and constant query complexity* (indeed as in §9.3.2.1). This proof system too has additional properties enabling proof composition as in Step (3).

3. The proof composition paradigm:[21] In general, this paradigm allows to compose two proof systems such that the "inner" one is used to probabilistically verify the acceptance criteria of the "outer" verifier. The aim is to conduct the latter verification using much fewer queries than the query complexity of the "outer" proof system. In particular, the inner verifier cannot afford to read its input, which makes composition more subtle than the term suggests.

   Loosely speaking, the *outer* verifier should be robust in the sense that its soundness condition guarantee that with high probability the oracle answers are "far" from satisfying the residual decision predicate (rather than merely not satisfy it). (Furthermore, the latter predicate, which is well-defined by the non-adaptive nature of the outer verifier, must have a circuit of size bounded by a polynomial in the number of queries.) The *inner* verifier is given oracle access to its input and is charged for each query made to it, but is only required to reject with high probability inputs that are far from being valid (and, as usual, accept inputs that are valid). That is, the inner verifier is actually a verifier of proximity.

   Composing two such PCPs yields a new PCP for $\mathcal{NP}$, where the new proof oracle consists of the proof oracle of the "outer" system and a sequence of proof oracles for the "inner" system (one "inner" proof per each possible random-tape of the "outer" verifier). Thus, composing an outer verifier of randomness complexity r' and query complexity q' with an inner verifier of randomness complexity r'' and query complexity q'' yields a PCP of randomness complexity $r(n) = r'(n) + r''(q'(n))$ and query complexity $q(n) = q''(q'(n))$, because $q'(n)$ represents the length of the input (oracle) that is accessed by the inner verifier. Recall that the outer verifier is non-adaptive, and thus if the inner verifier is non-adaptive (resp., robust) then so is the verifier resulting from the composition, which is important in case we wish to compose the latter verifier with another inner verifier.

In particular, the proof system of Step (1) is composed with itself [using $r'(n) = r''(n) = O(\log n)$ and $q'(n) = q''(n) = \text{poly}(\log n)$] yielding a PCP system (for $\mathcal{NP}$) of randomness complexity $r(n) = r'(n)+r''(q'(n)) = O(\log n)$ and query complexity $q(n) = q''(q'(n)) = \text{poly}(\log\log n)$. Composing the latter system (used as an "outer" system) with the the PCP system of Step (2), yields a PCP system (for

[21] Our presentation of the composition paradigm follows [33], rather than the original presentation of [14, 13].

$\mathcal{NP}$) of randomness complexity $r(n)+\text{poly}(q(n)) = O(\log n)$ and query complexity $O(1)$, thus establishing the PCP Theorem.

A more detailed overview – the plan. The foregoing description of Step 3 uses two (non-trivial) PCP systems and refers to additional properties such as robustness and verification of proximity. A PCP system of polynomial randomness complexity and constant query complexity (as postulated in Step 2) is outlined in §9.3.2.1. We thus start by discussing the notions of verifying proximity and being robust, while demonstrating their applicability to the said PCP. Finally, we outline the other PCP system (i.e., the one postulated in Step 1).

PCPs of Proximity. Recall that a standard PCP verifier gets an explicit input and is given oracle access to an alleged proof (for membership of the input in a predetermined set). In contrast, a PCP of proximity is given oracle access to two oracles, one representing an input and the other being an alleged proof. Typically, the query complexity of the corresponding verifier is lower than the length of the input oracle, and hence the verifier cannot afford to read the entire input and cannot be expected to make absolute statements about it. Indeed, instead of deciding whether or not the input is in a predetermined set, the verifier needs only distinguish the case that the input is in the set from the case that the input is *far* from the set (where far means being at *relative* Hamming distance at least 0.01 (or any other constant)).

For example, consider a variant of the system of §9.3.2.1 in which the quadratic system is fixed[22] and the verifier needs to determine whether an input oracle satisfies the said system or is far from any assignment that satisfies it. The proof oracle is as in §9.3.2.1, and a PCP of proximity may proceed as in §9.3.2.1 and in addition perform a proximity test to verify that the input oracle is close to the assignment encoded in the proof oracle. Specifically, the verifier may read a uniformly selected bit of the input oracle and compare the value to the self-correction obtained from the proof oracle (i.e., for a uniformly selected $i \in \{1, ..., n\}$, we compare the i^{th} bit of the input oracle to the self-correction of the value $T_1(0^{i-1}10^{n-i})$, obtained from the proof oracle).

Robust PCPs. Composing an "outer" PCP verifier with an "inner" PCP verifier of proximity makes sense provided that the *outer* verifier rejects in a "robust" manner. That is, the soundness condition of a robust verifier requires that (with probability at least 1/2) the oracle answers are *far* from any sequence that is acceptable by the residual predicate (rather than merely that the answers are rejected by this predicate). Indeed, if the outer verifier is (non-adaptive and) robust, then it suffices that the inner verifier distinguish (with the help of an adequate proof) answers that are valid from answers that are far from being valid.

For example, if robustness is defined as referring to *relative constant distance* (which is indeed the case), then the PCP of §9.3.2.1 (as well as any PCP of con-

[22] Indeed, in our applications the quadratic system will be "known" to the ("inner") verifier, because it is determined by the ("outer") verifier.

stant query complexity) is trivially robust. However, we will not care about the robustness of this PCP, because we only use this PCP as an inner verifier in proof composition. In contrast, we will care about the robustness of PCPs that are used as outer verifiers (e.g., the PCP presented next).

Teaching note: Unfortunately, the construction of a PCP of logarithmic randomness and polylogarithmic query complexity for NP is quite complicated and involves many technical details. Furthermore, obtaining a robust version of this PCP is beyond the scope of the current text. Thus, we view the following description as merely providing a flavor of the underlying ideas.

PCP of logarithmic randomness and polylogarithmic query complexity for NP. We start by showing that $NP \subseteq \mathcal{PCP}(f, f)$, for $f(n) = \text{poly}(\log n)$. The proof system is based on a different arithmetization of CNF formulae (i.e., different than the one used in §9.1.2.2 for constructing an interactive proof system for co$\mathcal{NP}$). In this arithmetization, the names of the variables (resp., clauses) of the input formula ϕ are represented by binary strings of logarithmic (in $|\phi|$) length, and a *generic* variable (resp., clause) of ϕ is represented by a logarithmic number of *new variables* (which are assigned values in a finite field $\mathrm{F} \supset \{0,1\}$). The (structure of the) input 3CNF formula $\phi(x_1, ..., x_n)$ is represented by a Boolean function $C_\phi : \{0,1\}^{O(\log n)} \to \{0,1\}$ such that $C_\phi(\alpha, \beta_1, \beta_2, \beta_3) = 1$ if and only if, for $i = 1, 2, 3$, the i^{th} literal in the α^{th} clause has index $\beta_i = (\gamma_i, \sigma_i)$ (which is viewed as a variable name augmented by its sign). We consider a multi-linear extension of C_ϕ over F, denoted Φ; that is, Φ is the (unique) multi-linear polynomial that agrees with C_ϕ on $\{0,1\}^{O(\log n)} \subset \mathrm{F}^{O(\log n)}$. Thus, on input ϕ, the verifier first constructs C_ϕ and Φ. Part of the proof oracle of this verifier is viewed as function $A : \mathrm{F}^{\log n} \to \mathrm{F}$, which is supposed to be a multi-linear extension of a truth assignment that satisfies ϕ (i.e., for every $\gamma \in \{0,1\}^{\log n} \equiv [n]$, the value $A(\gamma)$ is supposed to be the value of the γ^{th} variable in such an assignment). Thus, we wish to check whether, for every $\alpha \in \{0,1\}^{\log |\phi|}$, it holds that

$$\sum_{\beta_1\beta_2\beta_3 \in \{0,1\}^{3\log 2n}} \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \tag{9.6}$$

where $A'(\beta)$ is the value assigned by A to the β^{th} literal (i.e., if $\beta = (\gamma, \sigma)$, where $\gamma \in \{0,1\}^{\log n}$ is a variable name and $\sigma \in \{0,1\}$ is the literal's type, then $A'(\beta) = \sigma \cdot A(\gamma) + (1-\sigma) \cdot (1 - A(\gamma))$). Thus, Eq. (9.6) holds if and only if the α^{th} clause is satisfied by the assignment induced by A (since $A'(\beta) = 1$ must hold for at least one of the three literals β that appear in this clause). As in §9.3.2.1, we cannot afford to verify all instances of Eq. (9.6), and unlike in §9.3.2.1 we cannot afford to take a random linear combination of them either (as this requires too much randomness). Fortuntaely, taking a "pseudorandom" linear combination of these equations is good enough. Specifically, using an adequate small-bias probability space[23] $S \subset$

[23]See Section 8.6.2.

$\mathrm{F}^{|\phi|}$ (of size poly$(|\phi| \cdot |F|)$), the verifier may select uniformly $(s_1, ..., s_{|\phi|}) \in S$ and verify

$$\sum_{\alpha\beta_1\beta_2\beta_3 \in \{0,1\}^\ell} s_\alpha \cdot \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \tag{9.7}$$

where $\ell \stackrel{\text{def}}{=} \log|\phi| + 3\log 2n$. The small-bias property guarantees that if A fails to satisfy any of the equations of type Eq. (9.6) then, with high probability (over the choice of $(s_1, ..., s_{|\phi|}) \in S$), it is the case that A fails to satisfy Eq. (9.7). (Since $|S| = \text{poly}(|\phi| \cdot |F|)$ rather that $|S| = 2^{|\phi|}$, we can select a sample in S using $O(\log|\phi|)$ coin tosses.)

Now, assuming that A is a low-degree polynomial, we can probabilistically verify Eq. (9.7) by applying a summation test (as in the interactive proof for co$\mathcal{NP}$). Indeed, the verifier obtains the relevant univariant polynomials by making adequate queries (which specify the entire sequence of choices made so far in the summation test). Note that after stripping the ℓ summations, the verifier end-ups up with an expression that contains three unknown values of A', which it may obtain by making corresponding queries to A. This summation test involves tossing $\ell \cdot \log|\mathrm{F}|$ coins and making $(\ell+3) \cdot O(\log|\mathrm{F}|)$ Boolean queries (which correspond to ℓ queries that are each answered by a univariate polynomial of constant degree (over F), and three queries to A (each answered by an element of F)). Needless to say, we must also check that A is indeed a multi-variate polynomial of low degree (or rather that it is close to such a polynomial). A low-degree test of complexities similar to those of the summation text does exist. Using a finite field F of poly$(\log(n))$ elements, this yields $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$ for $f(n) \stackrel{\text{def}}{=} O(\log(n) \cdot \log\log(n))$.

To obtain the desired PCP system of logarithmic randomness complexity, we represent the names of the original variables and clauses by $\frac{O(\log n)}{\log\log n}$-long sequences over $\{1, ..., \log n\}$, rather than by logarithmically-long binary sequences. This requires using low degree polynomial extensions (i.e., polynomial of degree $(\log n) - 1$), rather than multi-linear extensions. We can still use a finite field of poly$(\log(n))$ elements, and so we need only $\frac{O(\log n)}{\log\log n} \cdot O(\log\log n)$ random bits for the summation and low-degree tests. However, the number of queries (needed for obtaining the answers in these tests) grows, because now the polynomials involved have individual degree $(\log n) - 1$ rather than constant individual degree. This merely means that the query complexity increases by a factor of $O(\log n / \log\log n)$. Thus, we obtain $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q)$ for $q(n) \stackrel{\text{def}}{=} O(\log^2 n)$.

Recall that, in order to use the latter PCP system in composition, we need to guarantee that it (or a version of it) is robust as well as to present a version that is a PCP of proximity. The latter version is relatively easy to obtain (using ideas as applied to the PCP of §9.3.2.1), whereas obtaining robustness is too complex to be described here. We comment that one way of obtaining a robust PCP system is by a generic application of a (randomness-efficient) "parallelization" of PCP systems (cf. [13]), which in turn depends heavily on highly efficient low-degree tests. A alternative approach (cf. [33]) capitalizes of the specific structure of the summation test (as well as on the actual robustness of the low-degree test).

Digest. Assuming that $\mathcal{P} \neq \mathcal{NP}$, the PCP Theorem asserts a PCP system that obtains simultaneously the minimal possible randomness and query complexity (up to a multiplicative factor). The forgoing construction obtains this remarkable result by combining two different PCPs: the first PCP obtains logarithmic randomness but uses polylogarithmically many queries, whereas the second PCP uses a constant number of queries but has polynomial randomness complexity. We stress that each of the two PCP systems is highly non-trivial and very interesting by itself. We highlight the fact that these PCPs can be composed using a very simple composition method that refers to auxiliary properties such as robustness and proximity testing. (Composition of PCP systems that lack these extra properties is possible, but is far more cumbersome and complex.)

9.3.2.3 Overview of the second proof of the PCP Theorem

The original proof of the PCP Theorem focuses on the construction of two PCP systems that are highly non-trivial and interesting by themselves, and combines them in a natural manner. Loosely speaking, this combination (via proof composition) *preserves* the good features of each of the two systems; that is, it yields a PCP system that inherits the (logarithmic) randomness complexity of one system and the (constant) query complexity of the other. In contrast, the following alternative proof is focused at the "amplification" of PCP systems, via a gradual process of logarithmically many steps. We start with a trivial "PCP" system that rejects false assertions with probability inversely proportional to their length, and double the rejection probability in each step. Furthermore, in each step, the constant query complexity is preserved and the length of the PCP oracle is increased only by a constant factor. Thus, the process gradually transforms an extremely weak PCP system into a remarkably strong PCP system as postulated in the PCP Theorem.

In order to describe the aforementioned process we need to redefine PCP systems so to allow arbitrary soundness error. In fact, for technical reasons it is more convenient to describe the process as an interated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labelled) graphs such that the vertices correspond to (non-Boolean) variables and the edges are associated with constraints.

Definition 9.18 (CSP with 2-variable constraints): *For a fixed finite set Σ, an instance of* CSP *consists of a graph $G = (V, E)$* (which may have parallel edges and self-loops) *and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e \in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0,1\}$. The value of an assignment $\alpha : V \to \Sigma$ is the number of constraints satisfied by α; that is, the value of α is $|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 1\}|$. We denote by $\mathtt{vlt}(G,\Phi)$ the fraction of unsatisfied constraints under the best possible assignment; that is,*

$$\mathtt{vlt}(G,\Phi) = \min_{\alpha : V \to \Sigma} \{|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 0\}|/|E|\} \qquad (9.8)$$

For various functions $t : \mathbb{N} \to [0,1]$, we will consider the promise problem $\mathtt{gapCSP}_t^{\Sigma}$, having instances as above, such that the yes*-instances are fully satisfiable instances* (i.e., $\mathtt{vlt} = 0$) *and the* no-*instances are pairs* (G, Φ) *satisfying* $\mathtt{vlt}(G, \Phi) \geq t(|G|)$, *where* $|G|$ *denotes the number of edges in* G.

Note that 3SAT is reducible to $\mathtt{gapCSP}_t^{\{1,\ldots,7\}}$ for $t(m) = 1/m$; see Exercise 9.15. Our goal is to reduce 3SAT (or rather $\mathtt{gapCSP}_t^{\{1,\ldots,7\}}$) to $\mathtt{gapCSP}_c^{\Sigma}$, for some fixed finite Σ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}_c^{\Sigma}$; see Exercise 9.16. The desired reduction is obtained by iteratively applying the following reduction logarithmically many times.

Lemma 9.19 (amplifying reduction of gapCSP to itself): *For some finite* Σ *and constant* $c > 0$*, there exists a polynomial-time reduction of* $\mathtt{gapCSP}^{\Sigma}$ *to itself such that the following conditions hold with respect to the mapping of any instance* (G, Φ) *to the instance* (G', Φ').

1. *If* $\mathtt{vlt}(G, \Phi) = 0$ *then* $\mathtt{vlt}(G', \Phi') = 0$.
2. $\mathtt{vlt}(G', \Phi') \geq \min(2 \cdot \mathtt{vlt}(G, \Phi), c)$.
3. $|G'| = O(|G|)$.

Proof Outline:[24] The reduction consists of three steps. We first apply a pre-processing step that makes the underlying graph suitable for further analysis. The value of vlt may decrease during this step by a constant factor. The heart of the reduction is the second step in which we may increase vlt by any desired constant factor. The latter step also increases the alphabet Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems; e.g., the one presented in §9.3.2.1). Details follow.

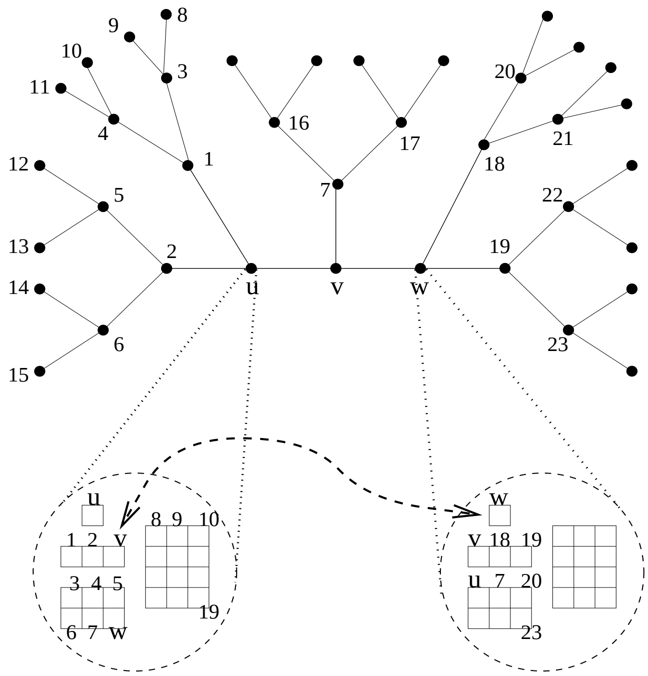
We first note that the aforementioned Σ and c, as well as the auxiliary parameters d and t, are fixed constants that will be determined so to satisfy various conditions that arise in the course of our argument.

We start with the pre-processing step. Our aim in this step is to reduce the input (G, Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1, Φ_1) such that G_1 is a d-regular expander graph.[25] Furthermore, each vertex in G_1 will have at least $d/2$ self-loops, $|G_1| = O(|G|)$, and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. This step is quite simple: see Exercise 9.17.

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. This is done by reducing the instance (G_1, Φ_1) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\mathtt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$. Specifically, the vertex set of G_2 is identical to the vertex set of G_1, and each t-edge long path in G_1 is replaced by a corresponding edge in G_2, which is thus a d^t-regular graph. The constraints in Φ_2 are the natural ones, viewing each element of Σ' as a Σ-labelling of the ("distance $\leq t$") neigborhood of a vertex (see Figure 9.3), and checking that

[24] For details, see [63].

[25] A d-regular graph is a graph in which each vertex is incident to exactly d edges. Loosely speaking, an expander graph has the property that each moderately balanced cut (i.e., partition of its vertex set) has relatively many edges crossing it. An equivalent definition, also used in the actual analysis, is that the second eigenvalue of the corresponding adjacency matrix has asolute value that is bounded away from d.



The alphabet Σ' as a labelling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown. The two-sided arrow indicates one of the edges in G_1 that will contribute to the edge constraint between u and w in (G_2, Φ_2).

Figure 9.3: The amplifying reduction in the second proof of the PCP Theorem.

two such labellings are consistent as well as satisfy Φ_1. That is, suppose that there is a path of length at most t in G_1 going from vertex u to vertex w and passing through vertex v. Then, there is an edge in G_1 between vertices u and w, and the constraint associated with it with mandates that the entries correspondings to vertex v in the Σ'-labelling of vertices u and w are identical. In addition, if the G_1-edge (v, v') is on a path of length at most t starting at u then the corresponding edge in G_2 is associated a constraint that enforces the constraint that is associated to (v, v') in Φ_1.

Clearly, if $\texttt{vlt}(G_1, \Phi_1) = 0$ then $\texttt{vlt}(G_2, \Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\texttt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \texttt{vlt}(G_1, \Phi_1)), c)$. Here we merely provide a rough intuition and refer the interested reader to [63]. The intuition is that any Σ'-labelling to the vertices of G_2 must either be consistent with a Σ-labelling of G_1 or violate the "equality constraints" of many edges in G_2. Focusing on the first case and relying on the hypothesis that G_1 is an expander, it follows that the set of violated edge-constaints (of Φ_1) with respect to the aforementioned Σ-labelling causes many more edge-constaints of Φ_2 to be violated (by virtue of the latter enforcing many edge-constaints of Φ_1). The point is that a set F of edges of G_1 is likely to appear on a $\min(\Omega(t) \cdot |F|/|G_1|, \Omega(1))$ fraction of the edges of G_2 (i.e., t-paths of G_1). (Note that the claim is obvious if G_1 were a complete graph, but it also holds for an expander.)[26]

The factor of $\Omega(\sqrt{t})$ gained in the second step makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step). Furthermore, for a suitable choice of the constant t, the aforementioned gain yields an overall constant factor amplification (of $\mathtt{vlt}$). However, so far we obtained an instance of $\mathtt{gapCSP}^{\Sigma'}$ rather than an instance of $\mathtt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\mathtt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\mathtt{gapCSP}^{\Sigma'}$ as a (weak) PCP system and composing it with an inner-verifier, using the proof composition paradigm outlined in §9.3.2.2. We stress that the inner-verifier used here needs only handle instances of constant size (i.e., having description length $O(d^t \log|\Sigma|)$), and so the one presented in §9.3.2.1 will do. The resulting PCP-system uses randomness $r \stackrel{\text{def}}{=} \log_2|G_2| + O(d^t \log|\Sigma|)^2$ and a constant number of binary queries, and has rejection probability $\Omega(\mathtt{vlt}(G_2, \Phi_2))$, which is indepedent of the choice of the constant t. As in Exercise 9.15, for $\Sigma = \{0,1\}^{O(1)}$, we can easily obtain an instance of $\mathtt{gapCSP}^{\Sigma}$ that has a $\Omega(\mathtt{vlt}(G_2, \Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance is $O(2^r) = O(|G_2|)$, because d and t are constants. This completes the last step as well as the (outline of the) proof of the entire lemma. □

9.3.3 PCP and Approximation

The characterization of $\mathcal{NP}$ in terms of probabilistically checkable proofs plays a central role in the study of the complexity of approximation problems (cf., Section 10.1.1). To demonstrate this relationship, we first note that a PCP system V gives rise to a natural approximation problem; that is, on input x, the task is approximating the probability that V accepts x when given oracle access to the best possible π (i.e., we wish to approximate $\max_\pi\{\Pr[V^\pi(x) = 1]\}$). Thus, if $S \in \mathcal{PCP}(r, q)$ then deciding membership in S is reducible to approximating the maximum among $\exp(2^{r+q})$ quantities (corresponding to all effective oracles), where each quantity can be evaluated in time $2^r \cdot \text{poly}$. Note that an approximation up to a constant factor (of 2) will do.

Let us take a closer look at the approximation problem associated with $\mathcal{PCP}(r, q)$. We focus, for simplicity, on the case of non-adaptive PCP systems (i.e., all the queries are determined beforehand by the input and the internal coin tosses of the verifier). On input x, we consider $2^{r(|x|)}$ formulae, each depending on $q(|x|)$ Boolean variables that represent the values of the corresponding bits in the proof oracle. Thus, if x is a yes-instance then there exists a truth assignment (to these variables) that satisfies all $2^{r(|x|)}$ formulae, whereas if x is a no-instance then there exists no

[26] We also note that due to a technical difficulty it is easier to establish the claimed bound of $\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1))$ rather than $\Omega(t \cdot \mathtt{vlt}(G_1, \Phi_1))$.

truth assignment that satisfies more than $2^{r(|x|)-1}$ formulae. When focusing on the case that q is constant, this motivates the following definition.

Definition 9.20 (gap problems for SAT and generalized-SAT): *For constants $q \in \mathbb{N}$ and $\varepsilon > 0$, the promise problem* $\texttt{gapGSAT}_\varepsilon^q$ *consists of instances that are each a sequence of q-variable Boolean formulae. The* yes*-instances are sequences that are simultaneously satisfiable, whereas the* no*-instances are sequences for which no Boolean assignment satisfies more than a $1 - \varepsilon$ fraction of the formulae in the sequence. The promise problem* $\texttt{gapSAT}_\varepsilon^q$ *is defined analogously, except that in this case each instance is a sequence of formulae that are each a single disjunctive clause.*

Indeed, each instance of $\texttt{gapSAT}_\varepsilon^q$ is naturally viewed as q-CNF formulae, and we consider an assignment that satisfies as many clauses (of the input CNF) as possible. As hinted, $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$ implies that $\texttt{gapGSAT}_{1/2}^{O(1)}$ is $\mathcal{NP}$-complete. The converse holds too, and (for some constant $\varepsilon > 0$) the same holds for $\texttt{gapSAT}_\varepsilon^3$.

Theorem 9.21 (equivalent formulations of the PCP Theorem). *The following three conditions are equivalent:*

1. The PCP Theorem: *there exists a constant q such that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q)$.*
2. *There exists a constant q such that* $\texttt{gapGSAT}_{1/2}^q$ *is $\mathcal{NP}$-hard.*
3. *There exists a constant $\varepsilon > 0$ such that* $\texttt{gapSAT}_\varepsilon^3$ *is $\mathcal{NP}$-hard.*

Note that Items 2 and 3 make no reference to PCP. Instead, they manifest that the hardness of approximation lies at the heart of the PCP Theorem. In general, probabilistic checkable proof systems for $\mathcal{NP}$ yield strong inapproximability results for various classical optimization problems (cf., Exercise 9.10 and Section 10.1.1).

Proof: We first show that the PCP Theorem implies the NP-hardness of $\texttt{gapGSAT}$. We may assume, without loss of generality, that, for some constant q and every $S \in \mathcal{NP}$, it holds that $S \in \mathcal{PCP}(O(\log), q)$ via a non-adaptive verifier (because q adaptive queries can be emulated by 2^q non-adaptive queries). We reduce S to $\texttt{gapGSAT}$ as follows. On input x, we consider all $2^{O(\log|x|)}$ possible coin tosses of the verifier, and for each sequence of outcomes we determine the queries made as well as the residual decision predicate (of the PCP verifier). That is, for each random-outcome $\omega \in \{0,1\}^{O(\log|x|)}$, we consider the residual predicate determined by x and ω, and note that this predicate depends only on q variables (which represent the values of the q corresponding oracle answers). Thus, we obtain an instance of $\texttt{gapGSAT}^q$, and indeed $x \in S$ (resp., $x \notin S$) is mapped to a yes-instance (resp., no-instance) of $\texttt{gapGSAT}_{1/2}^q$.

Item 2 implies Item 3 by a standard reduction of GSAT to 3SAT. Specifically, $\texttt{gapGSAT}_{1/2}^q$ reduces to $\texttt{gapSAT}_{2^{-(q+1)}}^q$, which in turn reduces to $\texttt{gapSAT}_\varepsilon^3$ for $\varepsilon = 2^{-(q+1)}/(q-2)$. (Clearly, Item 3 implies Item 2.)

We complete the proof by showing that Item 3 implies Item 1. In fact, we show that $\texttt{gapGSAT}_\varepsilon^q$ is in $\mathcal{PCP}(O(\varepsilon^{-1}\log), O(q/\varepsilon))$, and do so by presenting a very natural PCP system. In this PCP system the proof oracle is supposed to be an satisfying assignment, and the verifier selects at random one of the (q-variable) formulae in the input sequence, and checks whether it is satisfied by the (assignment given by the) oracle. This amounts to tossing logarithmically many coins and making q queries. This verifier always accepts yes-instances (when given access to an adequate oracle), whereas each no-instances is rejected with probability at least ε (no matter which oracle is used). To amplify the rejection probability (to the desired threshold of $1/2$), we invoke the foregoing verifier $O(\varepsilon^{-1})$ times. ■

Gap amplifying reductions – a reflection. Items 2 and 3 of Theorem 9.21 assert the existence of "gap amplifying" reductions of problems like 3SAT to themselves. These reductions map yes-instances to yes-instances (as usual), while mapping no-instances to no-instances of a special type such that a "gap" is created between the yes-instances and no-instances at the image of the reduction. For example, in the case of 3SAT, unsatisfiable formulae are mapped to formulae that are not merely unsatisfiable but rather have no assignment that satisfies more than a $1-\varepsilon$ fraction of the clauses. Thus, PCP constructions are essentially "gap amplifying" reductions.

9.3.4 More on PCP itself: an overview

We start by discussing variants of the PCP characterization of NP, and next turn to PCPs having expressing power beyond NP.

9.3.4.1 More on the PCP characterization of NP

Interestingly, the two complexity measures in the PCP-characterization of $\mathcal{NP}$ can be traded off such that at the extremes we get $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$ and $\mathcal{NP} = \mathcal{PCP}(0, \texttt{poly})$, respectively.

Proposition 9.22 *For every $S \in \mathcal{NP}$, there exist a constant α such that for every integer function ℓ that satisfies $0 \leq \ell(n) \leq \alpha \log_2 n$ it holds that $S \in \mathcal{PCP}(r, q) \subseteq \mathcal{NP}$, where $r(n) = \alpha \cdot \log_2 n - \ell(n)$ and $q(n) = O(2^{\ell(n)})$.*

Proof Sketch: Starting with Theorem 9.16, consider all possibilities for the $\ell(n)$-long prefix of the random tape of the verifier. □

Following the establishment of Theorem 9.16, numerous variants of the PCP Characterization of NP were explored. Following is a brief summary of some of these studies.[27]

[27] With the exception of works that appeared after [86], we provide no references for the results quoted here. We refer the interested reader to [86, Sec. 2.4.4].

The length of PCPs. Recall that the effective length of the oracle in any $\mathcal{PCP}(\texttt{log}, \texttt{log})$ system is polynomial (in the length of the input). Furthermore, in the PCP systems underlying the proof of Theorem 9.16 the queries refer only to a polynomially long prefix of the oracle, and so the actual length of these PCPs for $\mathcal{NP}$ is polynomial. Remarkablly, the length of PCPs for $\mathcal{NP}$ can be made nearly-linear (in the combined length of the input and the standard NP-witness), while maintaining constant query complexity, where by nearly-linear we mean linear up to a poly-logarithmic factor. (For details see [34, 63].) This means that *the redundency required for supporting probabilistic verification of proofs is relatively modest.*

The number of queries in PCPs. Theorem 9.16 asserts that a constant number of queries suffice for PCPs with logarithmic randomness and soundness error 1/2 (for NP). It is currently known that this constant is at most 5, whereas with 3 queries one may get arbitrary close to error 1/2. The obvious trade-off between the number of queries and the soundness error gives rise to the robust notion of amortized query complexity, defined as the ratio between the number of queries and (minus) the logarithm (to based 2) of the soundness error. *For every $\varepsilon > 0$, any set in $\mathcal{NP}$ has a PCP system with logarithmic randomness and amortized query complexity* $1+\varepsilon$ (cf. [114]), whereas only sets in $\mathcal{P}$ have PCPs of logarithmic randomness and amortized query complexity 1 (or less).

The free-bit complexity. The motivation to the notion of free bits came from the PCP–to–MaxClique connection (see Exercise 9.10 and [27, Sec. 8]), but we believe that this notion is of independent interest. Intuitively, this notion distinguishes between queries for which the acceptable answer is determined by previously obtained answers (i.e., the verifier compares the answer to a value determined by the previous answers) and queries for which the verifier only records the answer for future usage. The latter queries are called free (because any answer to them is "acceptable"). For example, in the linearity test (see §9.3.2.1) the first two queries are free and the third is not (i.e., the test accepts if and only if $f(x) + f(y) = f(x + y)$). The amortized free-bit complexity is define analogously to the amortized query complexity. Interestingly, *$\mathcal{NP}$ has PCPs with logarithmic randomness and amortized free-bit complexity less than any positive constant.*

Adaptive versus non-adaptive verifiers. Recall that a PCP verifier is called non-adaptive if its queries are determined solely based on its input and the outcome of its coin tosses. (A general verifier, called adaptive, may determine its queries also based on previously received oracle answers.) Recall that the PCP Characterization of NP (i.e., Theorem 9.16) is established using a non-adaptive verifier; however, it turns out that *adaptive verifiers are more powerful than non-adaptive ones in terms of quantitative results*: Specifically, for PCP verifiers making three queries and having logarithmic randomness complexity, adaptive queries provide for soundness error at most 0.51 (actually $0.5 + \varepsilon$ for any $\varepsilon > 0$) for any set in $\mathcal{NP}$, whereas *non-adaptive* queries provide soundness error 5/8 (or less) only for sets in $\mathcal{P}$.

Non-binary queries. Our definition of PCP allows only binary queries. Certainly, non-binary queries can always be coded as binary ones, but the converse is not necessarily valid, in particular in adversarial settings. Note that the soundness condition constitutes an implicit adversarial setting, where a bad proof may be thought of as being selected by an adversary. Thus, when several binary queries are packed into one non-binary query, the adversary need not respect the packing (i.e., it may answer inconsistently on the same binary query depending on the other queries packed with it). For this reason, "parallel repetition" is highly non-trivial in the PCP setting. Still, a Parallel Repetition Theorem that refers to independent invocations of the same PCP is known [172], but it is not applicable for obtaining soundness error smaller than a constant (while preserving logarithmic randomness). Nevertheless, using adequate "consistency tests" one may construct PCP systems for $\mathcal{NP}$ using logarithmic randomness, a constant number of (non-binary) queries and *soundness error exponential in the length of the answers.* (Currently, this is known only for sub-logarithmic answer lengths.)

9.3.4.2 PCP with super-logarithmic randomness

The foregoing text has focused on the important case where the verifier tosses logarithmically many coins, and hence the "effective proof length" is polynomial. Here we mention that the PCP Theorem scales up.

Theorem 9.23 (Theorem 9.16 – Generalized): *Let $t(\cdot)$ be an integer function such that $n<t(n)<2^{\text{poly}(n)}$. Then,* $\text{NTIME}(t) \subseteq \mathcal{PCP}(O(\log t), O(1))$.

Recall that $\mathcal{PCP}(r, q) \subseteq \text{NTIME}(t)$, for $t(n) = \text{poly}(n) \cdot 2^{r(n)}$.

Chapter Notes

(The following historical notes are quite long and still they fail to mention several important technical contributions that played an important role in the development of the area. For further details, the reader is referred to [86, Sec. 2.6.2].)

Motivated by the desire to formulate the most general type of "proofs" that may be used within cryptographic protocols, Goldwasser, Micali and Rackoff [105] introduced the notion of an *interactive proof system.* Although the main thrust of their work is the introduction of a special type of interactive proofs (i.e., ones that are *zero-knowledge*), the possibility that interactive proof systems may be more powerful from NP-proof systems has been pointed out in [105]. Independently of [105], Babai [16] suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games* (and conjectured to be "very close" to $\mathcal{NP}$). Syntactically, Arthur-Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones (cf., [107]). The speed-up result (i.e., $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$) is due to [20] (improving over [16]).

The first evidence of the power of interactive proofs was given by Goldreich, Micali, and Wigderson [96], who presented an interactive proof system for Graph

Non-Isomorphism (Construction 9.3). More importantly, they demonstrated the *generality and wide applicability of zero-knowledge proofs*: Assuming the existence of one-way function, they showed how to construct zero-knowledge interactive proofs for any set in $\mathcal{NP}$ (Theorem 9.11). This result has had a dramatic impact on the design of cryptographic protocols (cf., [97]). For further discussion of zero-knowledge and its applications to cryptography see Appendix C (and, in particular, Section C.4). Theorem 9.12 (i.e., $\mathcal{ZK} = \mathcal{IP}$) is due to [30, 122].

Probabilistically checkable proof (PCP) systems are related to *multi-prover interactive proof systems*, a generalization of interactive proofs that was suggested by Ben-Or, Goldwasser, Kilian and Wigderson [31]. Again, the main motivation came from the zero-knowledge perspective; specifically, introducing multi-prover zero-knowledge proofs for $\mathcal{NP}$ without relying on intractability assumptions. Yet, the complexity theoretic prospects of the new class, denoted $\mathcal{MIP}$, have not been ignored.

The amazing power of interactive proof systems has been demonstrated by using algebraic methods. The basic technique has been introduced by Lund, Fortnow, Karloff and Nisan [150], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [191] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai, Fortnow and Lund [17] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 9.4 follows [191].)

The aforementioned multi-prover proof system of Babai, Fortnow and Lund [17] (hereafter referred to as the BFL proof system) has been the starting point for fundamental developments regarding $\mathcal{NP}$. The first development was the discovery that the BFL proof system can be "scaled-down" from $\mathcal{NEXP}$ to $\mathcal{NP}$. This important discovery was made independently by two sets of authors: Babai, Fortnow, Levin, and Szegedy [18] and Feige, Goldwasser, Lovász, and Safra [69]. However, the manner in which the BFL proof is scaled-down is different in the two papers, and so are the consequences of the scaling-down.

Babai *et. al.* [18] start by considering (only) inputs encoded using a special error-correcting code. The encoding of strings, relative to this error-correcting code, can be computed in polynomial time. They presented an almost-linear time algorithm that transforms NP-witnesses (to inputs in a set $S \in \mathcal{NP}$) into *transparent proofs* that can be verified (as vouching for the correctness of the encoded assertion) in (probabilistic) *poly-logarithmic time* (by a Random Access Machine). Babai *et. al.* [18] stress the practical aspects of transparent proofs; specifically, for rapidly checking transcripts of long computations. In contrast, in the proof system of Feige *et. al.* [69, 70] the verifier stays polynomial-time and only two more refined complexity measures (i.e., the randomness and query complexities) are reduced to poly-logarithmic. This eliminates the need to assume that the input is in a special error-correcting form, and yields a refinement of the complexity class introduced in [76], where the refinement is obtained by specifying the randomness and query complexities (see Definition 9.14). Hence, whereas the BFL proof system [17] can be reinterpreted as establishing

$$\mathcal{NEXP} = \mathcal{PCP}(\texttt{poly}, \texttt{poly}), \tag{9.9}$$

the work of Feige *et. al.* [70] establishes

$$\mathcal{NP} \subseteq \mathcal{PCP}(f, f), \quad \text{where } f(n) = O(\log n \cdot \log\log n). \tag{9.10}$$

We note that the work of Babai *et. al.* [18] implies

$$\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, \mathtt{polylog})\,. \tag{9.11}$$

Interest in the new complexity class became immense since Feige *et. al.* [69, 70] demonstrated its relevance to proving the intractability of approximating some combinatorial problems (specifically, for MaxClique). When using the PCP–to–MaxClique connection established by Feige *et. al.*, the randomness and query complexities of the verifier (in a PCP system for an NP-complete set) relate to the strength of the negative results obtained for approximation problems. This fact provided a very strong motivation for trying to reduce these complexities and obtain a tight characterization of $\mathcal{NP}$ in terms of $\mathcal{PCP}(\cdot,\cdot)$. Once the work of Feige *et. al.* [70] had been presented, the challenge was clear: showing that $\mathcal{NP}$ equals $\mathcal{PCP}(\mathtt{log}, \mathtt{log})$. This challenge was met by Arora and Safra [14]. Actually, they showed that

$$\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, f(\cdot))\,, \qquad \text{where } f(n) = o(\log n). \tag{9.12}$$

Hence, a new challenge arose; namely, further reducing the query complexity – in particular to a constant – while maintaining the logarithmic randomness complexity. Again, additional motivation for this challenge came from the relevance of such a result to the study of approximation problems. The new challenge was met by Arora, Lund, Motwani, Sudan and Szegedy [13], and is captured by the equation

$$\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1)). \tag{9.13}$$

Indeed the PCP Characterization Theorem (captured in Eq. (9.13)) is a culmination of a sequence of great works [150, 17, 18, 70, 14, 13]. These works are rich in innovative ideas (e.g., various arithmetizations of SAT as well as various forms of proof composition) and employ numerous techniques (e.g., low-degree tests, self-correction, and pseudorandomness). For further detail, the interested reader is referred to [86, Sec. 2.6.2].

Our overview of the original proof of the PCP Theorem (in §9.3.2.1–9.3.2.2) is based on [13, 14].[28] The alternative proof outlined in §9.3.2.3 is due to Dinur [63]. We also mention some of the ideas and techniques involved in deriving variants of the PCP Theorem. These include the Parallel Repetition Theorem [172], the use of the Long-Code [27], and the application of Fourier analysis in this setting [111, 112].

Computationally-Sound Proof Systems. Argument systems were defined by Brassard, Chaum and Crépeau [46], with the motivation of providing *perfect* zero-knowledge arguments (rather than zero-knowledge *proofs*) for $\mathcal{NP}$. A few years

[28] Our presentation also benefits from the notions of PCPs of proximity and robustness, put forward in [33, 64].

later, Kilian [135] demonstrated their significance beyond the domain of zero-knowledge by showing that, under some reasonable intractability assumptions, every set in $\mathcal{NP}$ has a computationally-sound proof in which the randomness and communication complexities are poly-logarithmic.[29] Interestingly, these argument systems rely on $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, for $f(n) = \text{poly}(\log n)$. We mention that Micali [153] suggested a different type of computationally-sound proof systems (which he called CS-proofs).

Final comment: The current chapter is a revision of [86, Chap. 2]. In particular, more details are provided here for the main topics, whereas numerous secondary topics discussed in [86, Chap. 2] are not mentioned here (or are only briefly mentioned here). In addition, a couple of the research directions that were mentioned in [86, Sec. 2.4.4] received considerable attention in the period that elapsed, and improved results are currently known. In particular, the interested reader is referred to [33, 34, 63] (for a study of the length of PCPs) and to [114] (for a study of the amortized query complexity).

Exercises

Exercise 9.1 (parallel error-reduction for interactive proof systems) Prove that the error probability (in the soundness condition) can be reduced by parallel repetitions of the proof system.

Guideline: As a warm-up consider first the case of public-coin interactive proof systems. Next, note that the analysis generalizes to arbitrary interactive proof systems. (Extra hint: As a mental experiment, consider a "powerful verifier" that emulates the original verifier while behaining as in the public-coin model.) A proof appears in [86, Apdx. C.1].

Exercise 9.2 Complete the details of the proof that co$\mathcal{NP} \subseteq \mathcal{IP}$ (i.e., the first part of the proof of Theorem 9.4). In particular, regarding the proof of non-satisfiability of a CNF with n variables and m clauses, what is the length of the messages sent by the two parties? What is the soundness error?

Exercise 9.3 Present a $n/O(\log n)$-round interactive proof for the non-satisfiability of a CNF having n variables.
(Hint: Modify the (first part of the) proof of Theorem 9.4, by stripping $O(\log n)$ summations in each round.)

Exercise 9.4 Show that `QBF` can be reduced to a special form of `QBF` in which no variable appears both to the left and the right of more than one universal quantifier.

Guideline: Consider a process (which proceeds from left to right) of "refreshing" variables after each universal quantifier. Let $\phi(x_1, ..., x_s, y, x_{s+1}, ..., x_{s+t})$ be a quantifier-free boolean formula and let $Q_{s+1}, ..., Q_{s+t}$ be an arbitrary sequence of quantifiers. Then, we replace the quantified (sub-)formula

$$\forall y Q_{s+1} x_{s+1} \cdots Q_{s+t} x_{s+t}\, \phi(x_1, ..., x_s, y, x_{s+1}, ..., x_{s+t})$$

[29] We comment that interactive proofs are unlikely to have such low complexities; see [102].

by the (sub-)formula

$$\forall y \exists x_1' \cdots \exists x_s' [(\wedge_{i=1}^s (x_i' = x_i)) \ \wedge \ Q_{s+1} x_{s+1} \ \cdots \ Q_{s+t} x_{s+t} \, \phi(x_1', ..., x_s', y, x_{s+1}, ..., x_{s+t})] \, .$$

Note that the variables $x_1, ..., x_s$ do not appear to the right of the quantifier Q_{s+1} in the replaced formula, and that the length of the replaced formula grows by an additive term of $O(s)$. This process of refreshing variables is applied from left to right on the entire sequence of universal quantifiers (except the inner one, for which this refreshing is useless).[30]

Exercise 9.5 (on interactive proofs with two-sided error (following [78])) Let $\mathcal{IP}'(f)$ denote the class of sets having a two-sided error interactive proof system in which a total of $f(|x|)$ messages are exchanged on common input x. Extending the ideas underlying the proof of Theorem 6.7, show that $\mathcal{IP}'(f) \subseteq \mathcal{IP}(f+3)$. Note that $\mathcal{IP}'(f) \subseteq \mathcal{AM}(f+1)$ follows.

Guideline: Consider the public-coin version of $\mathcal{IP}'$, denoted $\mathcal{AM}'$, and note that the proof of $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+2)$ (cf. [107]) extends to $\mathcal{IP}'(f) \subseteq \mathcal{AM}'(f+2)$. Observe that the proof of Theorem 6.7 actually establishes that $\mathcal{BPP} \subseteq \mathcal{MA} = \mathcal{AM}(1)$, and that $\mathcal{BPP} = \mathcal{AM}'(0)$. Our aim is to establish $\mathcal{AM}'(f) \subseteq \mathcal{AM}(f+1)$ for arbitrary f (rather than for $f \equiv 0$). Consider an optimal prover strategy and the set of verifier coins that make the verifier accept any fixed yes-instance. Applying the ideas underlying the transformation of $\mathcal{BPP}$ to $\mathcal{MA}$, we obtain the desired result. For further details, see [78].

Exercise 9.6 In continuation to Exercise 9.5, show that $\mathcal{IP}'(f) = \mathcal{IP}(f)$ for every function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f \geq 1$.
(Hint: If $f \geq 2$ then the claim follows by combining Exercise 9.5 with $\mathcal{IP}(O(f)) = \mathcal{IP}(f)$. For the case of $f = 1$, note that $\mathcal{IP}(1) = \mathcal{AM}(1)$, and that the single prover message added in Exercise 9.5 can be incorporated in the single message sent in $\mathcal{IP}(1)$.)

Exercise 9.7 (on the effective length of PCP oracles) Suppose that V is a PCP verifier of query complexity q and randomness complexity r. Show that for every fixed x, the number of possible locations examined on input x (when considering all possible internal coin tosses of V and all possible answers it may receive) is upper-bounded by $2^{q(|x|)+r(|x|)}$. Show that if V is non-adaptive then the upper-bound can be improved to $2^{r(|x|)} \cdot q(|x|)$.
(Hint: In the adaptive case, the i^{th} query is determined by V's internal coin tosses and the previous $i-1$ answers. In the non-adaptive case, all q queries are determined by V's internal coin tosses.)

---

[30] For example,

$$\exists z_1 \forall z_2 \exists z_3 \forall z_4 \exists z_5 \forall z_6 \, \phi(z_1, z_2, z_3, z_4, z_5, z_6)$$

is first replaced by

$$\exists z_1 \forall z_2 \exists z_1' \, [(z_1' = z_1) \ \wedge \ \exists z_3 \forall z_4 \exists z_5 \forall z_6 \, \phi(z_1', z_2, z_3, z_4, z_5, z_6)]$$

and next (written as $\exists z_1 \forall z_2' \exists z_1' \, [(z_1' = z_1) \ \wedge \ \exists z_3' \forall z_4' \exists z_5' \forall z_6' \, \phi(z_1', z_2', z_3', z_4', z_5', z_6')])$ is replaced by

$$\exists z_1 \forall z_2' \exists z_1' \, [(z_1' = z_1) \ \wedge \ \exists z_3' \forall z_4' \exists z_1'' \exists z_2'' \exists z_3'' \\ [(\wedge_{i=1}^3 (z_i'' = z_i')) \ \wedge \ \exists z_5' \forall z_6' \phi(z_1'', z_2'', z_3'', z_4', z_5', z_6')]].$$

Thus, in the resulting formula, no variable appears both to the left and to the right of more than a single universal quantifier.

Exercise 9.8 (upper-bounds on the complexity of PCPs) Suppose that a set S has a PCP of query complexity q and randomness complexity r. Show that S can be decided by a non-deterministic machine that, on input of length n, makes at most $2^{r(n)} \cdot q(n)$ non-deterministic[31] steps and halts within a total number of $2^{r(n)} \cdot \text{poly}(n)$ steps. Thus, $S \in \text{NTIME}(2^r \cdot \text{poly}) \cap \text{DTIME}(2^{2^r q} \cdot \text{poly})$.

(Hint: For each input $x \in S$ and each possible value $\omega \in \{0,1\}^{r(|x|)}$ of the random-tape, we consider a sequence of $q(|x|)$ bit values that represent a sequence of oracle answers that make the verifier accept. Indeed, for fixed x and $\omega \in \{0,1\}^{r(|x|)}$, each setting of the $q(|x|)$ oracle answers determine the computation of the corresponding verifier (including the queries it makes).)

Exercise 9.9 (on the effective randomness of PCPs) Suppose that a set S has a PCP of query complexity q that utilizes proof oracles of length ℓ. Show that, for every constant $\varepsilon > 0$, the set S has a "non-uniform" PCP of query complexity q, soundness error $0.5 + \varepsilon$ and randomness complexity r such that $r(n) = O(1) + \log_2(\ell(n) + n)$. By a "non-uniform PCP" we mean one in which the verifier is a probabilistic polynomial-time oracle machine that gets a non-uniform advice of length polynomial in ℓ.

Guideline: Consider a PCP verifier V as in the hypothesis, and denote its randomness complexity by r_V. We construct a non-uniform verifier V' that, on input of length n, obtains as advice a set $R_n \subseteq \{0,1\}^{r_V(n)}$ of cardinality $O((\ell(n)+n)/\varepsilon^2)$, and emulates V on a uniformly selected element of R_n. Show that for a random R_n, the verifier V' satisfies the claims of the exercise.

(Extra hint: Fixing any input $x \notin S$ and any oracle $\pi \in \{0,1\}^{\ell(|x|)}$, upper-bound the probability that a random set R_n causes V' to accept x with probability $0.5 + \varepsilon$ when using the oracle π.)

Exercise 9.10 (The FGLSS-reduction [70]) For any $S \in \mathcal{PCP}(r,q)$, consider the following mapping of instances for S to instances of the `Independent Set` problem. The instance x is mapped to a graph $G_x = (V_x, E_x)$, where $V_x \subseteq \{0,1\}^{r(|x|)+q(|x|)}$ consists of pairs (ω, α) such that the PCP verifier *accepts* the input x, when using coins $\omega \in \{0,1\}^{r(|x|)}$ and receiving the answers $\alpha = \alpha_1 \cdots \alpha_{q(|x|)}$ (to the oracle queries determined by x, r and the previous answers). The set E_x consists of edges that connect vertices that represents inconsistent view of the said verifier; that is, the vertex $v = (\omega, \alpha_1 \cdots \alpha_{q(|x|)})$ is connected to the vertex $v' = (\omega', \alpha'_1 \cdots \alpha'_{q(|x|)})$ if there exists i and i' such that $\alpha_i \neq \alpha'_i$ and $\mathsf{q}_i^x(v) = \mathsf{q}_{i'}^x(v')$, where $\mathsf{q}_i^x(v)$ (resp., $\mathsf{q}_{i'}^x(v')$) denotes the i-th (resp., i'-th) query of the verifier on input x, when using coins ω (resp., ω') and receiving the answers $\alpha_1 \cdots \alpha_{i-1}$ (resp., $\alpha'_1 \cdots \alpha'_{i'-1}$). In particular, for every $\omega \in \{0,1\}^{r(|x|)}$ and $\alpha \neq \alpha'$, if $(\omega, \alpha), (\omega, \alpha') \in V_x$, then $((\omega, \alpha), (\omega, \alpha')) \in E_x$.

1. Prove that the mapping $x \mapsto G_x$ can be computed in time that is polynomial in $2^{r(|x|)+q(|x|)} \cdot |x|$.

   (Note that the number of vertices in G_x equals $2^{r(|x|)+f(|x|)}$, where f is the free-bit complexity of the PCP verifier.)

2. Prove that, for every x, the size of the maximum independent set in G_x is at most $2^{r(|x|)}$.

[31] See §4.2.1.3 for definition of non-deterministic machines.

3. Prove that if $x \in S$ then G_x has an independent set of size $2^{r(|x|)}$.

4. Prove that if $x \notin S$ then the size of the maximum independent set in G_x is at most $2^{r(|x|)-1}$.

In general, denoting the PCP verifier by V, prove that the size of the maximum independent set in G_x is exactly $2^{r(|x|)} \cdot \max_\pi\{\Pr[V^\pi(x) = 1]\}$. (Note the similarity to the proof of Proposition 2.25.) Show that the PCP Theorem implies that the size of the maximum independent set (resp., clique) in a graph is NP-hard to approximate to within any constant factor.

(Hint: Note that an independent set in G_x corresponds to a set of coins R and a partial oracle π' such that V accepts x when using coins in R and accessing any oracle that is consistent with π'. The FGLSS reduction creates a gap of a factor of 2 between yes and no-instances of S. Larger factors can be obtained by considering a PCP that results from repeating the given PCP for a constant number of times. The result for `Clique` follows by considering the complement graph.)

Exercise 9.11 In continuation to Exercise 9.10, prove that, for any $t(n) = o(\log n)$, it holds that $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$ implies that $\mathcal{P} = \mathcal{NP}$.

Guideline: We only use the fact that the said reduction actually reduces PCP to instances of the `Clique` problem (and ignore the fact that we actually get a stronger result that refers to a "gapClique" problem). Furthermore, when applies to problems in $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$, this reduction runs in polynomial-time. The key observation is that this reduction maps instances of the `Clique` problem (which is in $\mathcal{NP} \subseteq \mathcal{PCP}(o(\log), o(\log))$) to shorter instances of the same problem (because $2^{o(\log n)} \ll n$). Thus, iteratively applying the reduction, we can reduce instances of `Clique` to instances of constant size. Using the $\mathcal{NP}$-completeness of `Clique`, we reduce $\mathcal{NP}$ to a finite set, and $\mathcal{NP} = \mathcal{P}$ follows.

Exercise 9.12 (a simple but partial analysis of the BLR Linearity Test) For Abelian groups G and H, consider functions from G to H. For such a (generic) function f, consider the linearity (or rather homomorphism) test that selects uniformly $r, s \in G$ and checks that $f(r)+f(s) = f(r+s)$. Let $\delta(f)$ denote the distance of f from the set of homomorphisms (of G to H); that is, $\delta(f)$ is the minimum taken over all homomorphisms $h : G \to H$ of $\Pr_{x \in G}[f(x) \neq h(x)]$. Using the following guidelines, prove that the probability that the test rejects f, denoted $\varepsilon(f)$, is at least $3\delta(f) - 6\delta(f)^2$.

1. Suppose that h is the homomorphism closest to f (i.e., $\delta(f) = \Pr[f(x) \neq h(x)]$). Prove that $\varepsilon(f) = \Pr_{x,y \in G}[f(x) + f(y) \neq f(x,y)]$ is lower-bounded by $3 \cdot \Pr_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x,y) = h(x+y)]$.

   (Hint: consider three out of four disjoint events that support the case $f(x)+f(y) \neq f(x,y)$.)

2. Prove that $\Pr_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x,y) = h(x+y)] \geq \delta(f) - 2\delta(f)^2$.

   (Hint: lower-bound the said probability by $\Pr_{x,y}[f(x) \neq h(x)] - (\Pr_{x,y}[f(x) \neq h(x) \wedge f(y) \neq h(y)] + \Pr_{x,y}[f(x) \neq h(x) \wedge f(x+y) \neq h(x+y)])$.)

Note that the lower-bound $\varepsilon(f) \geq 3\delta(f) - 6\delta(f)^2$ increases with $\delta(f)$ only in the case that $\delta(f) \leq 1/4$. Furthermore, the lower-bound is useless in the case that

$\delta(f) \geq 1/2$. Thus an alternative lower-bound is needed in case $\delta(f)$ approaches $1/2$ (or is larger than it); see Exercise 9.13.

Exercise 9.13 (a better analysis of the BLR Linearity Test (cf. [38])) In continuation to Exercise 9.12, use the following guidelines in order to prove that $\varepsilon(f) \geq \min(1/7, \delta(f)/2)$. Specifically, focusing on the case that $\varepsilon(f) < 1/7$, show that f is $2\varepsilon(f)$-close to some homomorphism (and thus $\varepsilon(f) \geq \delta(f)/2$).

1. Define the vote of y regarding the value of f at x as $\phi_y(x) \stackrel{\rm def}{=} f(x+y)-f(y)$, and define $\phi(x)$ as the corresponding plurality vote (i.e., $\phi(x) \stackrel{\rm def}{=} \mathrm{argmax}_{v\in H}\{|\{y \in G : \phi_y(x)=v\}|\}$).

   Prove that, for every $x \in G$, it holds that $\mathsf{Pr}_y[\phi_y(x) = \phi(x)] \geq 1 - 2\varepsilon(f)$.

   Extra guideline: Fixing x, call a pair (y_1, y_2) good if $f(y_1) + f(y_2 - y_1) = f(y_2)$ and $f(x+y_1)+f(y_2-y_1) = f(x+y_2)$. Prove that, for any x, a random pair (y_1, y_2) is good with probability at least $1 - 2\varepsilon(f)$. On the other hand, for a good (y_1, y_2), it holds that $\phi_{y_1}(x) = \phi_{y_2}(x)$. Show that the graph in which edges correspond to good pairs must have a connected component of size at least $(1 - 2\varepsilon(f)) \cdot |G|$. Note that $\phi_y(x)$ is identical for all vertices y in this connected component, which in turn contains a majority of all y's in G.

2. Prove that ϕ is a homomorphism; that is, prove that, for every $x, y \in G$, it holds that $\phi(x) + \phi(y) = \phi(x + y)$.

   Extra guideline: Prove that $\phi(x) + \phi(y) = \phi(x + y)$ holds by considering the somewhat ficticious expression $\mathsf{Pr}_{r\in G}[\phi(x) + \phi(y) \neq \phi(x + y)]$, and showing that it is structly smaller than 1 (and hence $\phi(x)+\phi(y) \neq \phi(x+y)$ is false). Upper-bound the probabilistic expression by

   $$\mathsf{Pr}_r[\phi(x)\neq f(x + r) - f(r) \vee \phi(y)\neq f(r) - f(r - y) \vee \phi(x + y)\neq f(x + r) - f(r - y)].$$

   Use the union bound (and Item 1), and note that $\mathsf{Pr}_r[\phi(x) \neq f(x + r) - f(r)] < 2\varepsilon(f) < 1/3$, whereas $\mathsf{Pr}_r[\phi(y) \neq f(r) - f(r - y)] = \mathsf{Pr}_{r'}[\phi(y) \neq f(y + r') - f(r')]$ and $\mathsf{Pr}_r[\phi(x + y) \neq f(x + r) - f(r - y)] = \mathsf{Pr}_{r'}[\phi(x + y) \neq f(x + y + r') - f(r')]$ (by substituting $r' = r - y$).

3. Prove that f is $2\varepsilon(f)$-close to ϕ.

   Extra guideline: By Item 1, for every x, it holds that $\mathsf{Pr}_y[f(x+y)-f(y) = \phi(x)] \geq 1-2\varepsilon(f)$. Thus, there exists $y \in G$ such that $\mathsf{Pr}_x[f(x+y)-f(y) = \phi(x)] \geq 1-2\varepsilon(f)$, and for $z = x + y$ we have $\mathsf{Pr}_z[f(z) - f(y) = \phi(z - y)] \geq 1 - 2\varepsilon(f)$. Using Item 2 and letting $\eta \stackrel{\rm def}{=} f(y) - \phi(y)$, we have $\mathsf{Pr}_z[f(z) = \phi(z) + \eta] \geq 1 - 2\varepsilon(f)$. Prove that $\mathsf{Pr}_{x,y}[(\phi(x) + \eta) + (\phi(y) + \eta) = \phi(x + y) + \eta] \geq 1 - 7\varepsilon(f) > 0$, using $\varepsilon(f) < 1/7$, $\mathsf{Pr}_{x,y}[f(x)+f(y) = f(x+y)] = 1-\varepsilon(f)$, and $\mathsf{Pr}_x[f(x) = \phi(x)+\eta] \geq 1-2\varepsilon(f)$ (and the analogous statements for y and $x + y$). Using Item 2, $\eta = 0$ follows.

We comment that better bounds on the behavior of $\varepsilon(f)$ as a function of $\delta(f)$ are known.

Exercise 9.14 Let M be a non-zero m-by-n matrix over $\mathrm{GF}(p)$. Prove that $\mathsf{Pr}_{r,s}[r^\top M s \neq 0] \geq (1-p^{-1})^2$, where r (resp., s) is a random m-ary (resp., n-ary) vector.

Guideline: Prove that if $v \neq 0^m$ then $\mathsf{Pr}_s[v^\top s = 0] = p^{-1}$, and that if M has rank ρ then $\mathsf{Pr}_r[r^\top M = 0^n] = p^{-\rho}$.

Exercise 9.15 Show that 3SAT is reducible to $\mathtt{gapCSP}_t^{\{1,\ldots,7\}}$ for $t(m) = 1/m$, where gapCSP is as in Definition 9.18. Furthermore, show that the size of the resulting gapCSP instance is linear in the length of the input formula.

Guideline: Given an instance ψ of 3SAT, consider the graph in which vertices correspond to clauses of ψ, edges correspond to pairs of clauses that share a variable, and the constraints represent the natural consistency condition regarding partial assignments that satisfy the clauses. See a similar construction in Exercise 9.10.

Exercise 9.16 Show that, for any fixed finite Σ and constant $c > 0$, the problem $\mathtt{gapCSP}_c^\Sigma$ is in $\mathcal{PCP}(\mathtt{log}, O(1))$.

Guideline: Consider an oracle that, for some satisfying assignment for the CSP-instance (G, Φ), provides a trivial encoding of the assignment; that is, for a satisfying assignment $\alpha : V \to \Sigma$, the oracle responds to the query (v, i) with the i^{th} bit in the binary representation of $\alpha(v)$. Consider a verifier that uniformly selects an edge (u, v) of G and checks the constraint $\phi_{(u,v)}$ when applied to the values $\alpha(u)$ and $\alpha(v)$ obtained from the oracle.

Exercise 9.17 For any constant Σ and $d \geq 14$, show that $\mathtt{gapCSP}^\Sigma$ can be reduced to itself such that the instance at the target of the reduction is a d-regular expander, and the fraction of violated constraints is preserved up to a constant factor. That is, the instance (G, Φ) is reduced to (G_1, Φ_1) such that G_1 is a d-regular expander graph and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. Furthermore, make sure that $|G_1| = O(|G|)$ and that each vertex in G_1 has at least $d/2$ self-loops.

Guideline: First, replace each vertex of degree $d' > 3$ by a 3-regular expander of size d', and connect each of the original d' edges to a different vertex of this expander, thus obtaining a graph of maximum degree 4. Maintain the constraints associated with the original edges, and associate the equality constraint (i.e., $\phi(i,j) = 1$ if and only if $i = j$) to each new edge (residing in any of the added expanders). Next, denoting the number of vertices in the resulting graph by N_1, add to this graph a 3-regular expander of size N_1 (while associating with these edges the trivially satisfied constraint $\phi(i,j) = 1$ for all $i, j \in \Sigma$). Finally, add at least $d/2$ self-loops to each vertex (using again trivially satisfied constraints), so to obtain a d-regular graph. Prove that this sequence of modifications may only decrease the fraction of violated constraints, and that the decrease is only by a constant factor. The latter assertion relies on the equality constraints associated with the small expanders used in the first step.

Exercise 9.18 (free bit complexity zero) Note that only sets in $\mathcal{BPP}$ have PCPs of query complexity zero. Furthermore, Exercise 9.8 implies that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and query complexity zero.

1. Show that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and free-bit complexity zero.

   (Hint: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity zero.)

2. In contrast, show that Graph Non-Isomorphism has a PCP system of free-bit complexity zero (and linear randomness complexity).

Exercise 9.19 (free bit complexity one) In continuation to Exercise 9.18, prove that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and free-bit complexity one.

Guideline: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity one and randomness complexity r. Note that the question of whether the resulting graph has an independent set of size 2^r can be expressed as a 2CNF formula of size $\text{poly}(2^r)$, and see Exercise 2.21.