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 (which 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 an explicitly bounded probability of error, 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 meaningful conceptualization 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, the successful verification of a zero-knowledge proof exhibit an extreme contrast between being convinced of the validity of a statement and learning anything in addition (while receiving such a convincing proof). It turns out that, 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 locations (randomly) examined in the resulting proof and the confidence in its validity. In 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.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1).

Introduction and Preliminaries

The glory attached to the creativity involved in finding proofs, makes us forget that it is the less glorified process of verification that gives proofs their value. Conceptually speaking, proofs are secondary to the verification process; 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 alleged 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 construction of (meaningful) zero-knowledge proofs, which are of great theoretical and practical interest.
- NP-proofs can be efficiently transformed into a (redundant) form that supports super-fast probabilistic verification via very few random probes into the alleged proof.

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 $\text{poly}(|(x, y)|) = \text{poly}(|x|)$ for $|y| = \text{poly}(|x|)$.

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, in the context of Law, the cross-examination of a witness in court (including its non-verbal components) may be considered a proof (or a refutation) of some claim. Likewise, debates that take place in daily life have an analogous potential of establishing claims and are then perceived as proofs. This perception is quite common in philosophical and political debates, and arise also in scientific debates. Furthermore, some technical "proofs by contradiction" appeal to this daily experience 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. These conflicting attitudes are well-coupled with the difference between written proofs and "interactive" proofs: If one values the proof itself then one may insist on having it archived, whereas if one only cares about the consequence then the way in which it is reached is immaterial.

Interestingly, the set of daily attitudes will be adequate in the current chapter, where *proofs are viewed merely as a vehicle for the verification of the validity of claims*. (This attitude gets to an extreme in the case of zero-knowledge proofs, where we actually require that the proofs themselves be useless beyond being convincing of the validity of the claimed assertion.) In general, we will be interested in modeling various forms of proofs, focusing on proofs that can be verified by automated procedures. These verification procedures are designed to check the validity of potential proofs, and are oblivious of additional features that appeal to humans such as beauty, insightfulness, etc. In the current section we will consider the most general form of proof systems that still allow efficient verification.

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. In such a setting the prover's goal is to convince the verifier, while the verifier should make sure it is not fooled by the prover.

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 phenomenon is given a precise meaning in results such as *Gödel's Incompleteness Theorem* and Turing's theorem regarding the *undecidability of the Halting Problem*. In contrast, recall that $\mathcal{NP}$ was defined as the class of sets having proof systems that support efficient deterministic verification (of "written proofs"). This section is devoted to the study of a more liberal notion of efficient verification procedures (allowing both randomization and interaction).

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. Thus, referring to the aforementioned 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.*

[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.*

- 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 $k = \text{poly}(|x|)$. (Variants on the basic definition are discussed in Section 9.1.3.)

The role of randomness. Randomness is essential to the power of interactive proofs; that is, restricting the verifier to deterministic strategies yields a class of interactive proof systems that has no advantage over the class of 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 sequence of answers to the verifier's sequence of (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 (e.g., to the all-zero string) and noting that both (perfect) completeness and perfect (i.e., errorless) soundness still hold). Since V is deterministic, the prover can predict each message sent by V (because each such message is uniquely determined by the common input and the previous prover messages). 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.* (Note that we do not care about the complexity of determining such a sequence, since no computational bounds are placed on the prover.) Formally, $x \in S$ if and only if there exists a sequence of (prover's) messages that make (the deterministic) V accept x, where the question of whether a specific sequence makes V accept x depends only on the sequence and on the common input x (because V tosses no coins that may affect this decision). 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).[4] 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

[4]Note that knowledge of the verifier's messages may be essential for answering these questions convincingly. In the case that V is deterministic its messages can be determined by the prover, but this may not be possible in the general case (i.e., when V is randomized).

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$\mathcal{NP} \neq \mathcal{NP}$.

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 $\{\phi(v), \phi(u)\} \in E_2$. This ("edge preserving") mapping ϕ, in case it exists, is called an *isomorphism* between the graphs. The following protocol specifies a way of proving that two graphs are not isomorphic, while it is not known whether such a statement can be proven via a non-interactive process (i.e., 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 $G_2 = (V_2, E_2)$.*

- 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.[5] 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 $\mathcal{NP} \neq$ 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: 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: Our presentation focuses on the main ideas, and neglects various implementation details (which can be found in [155, 197]). 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, and replace the logical operations by corresponding arithmetic operations. In particular, OR-clauses are replaced by sums, and the top level conjunction is replaced by a product. Then, we consider the formal summation of the resulting arithmetic expression, where

[5]In case G_1 is not isomorphic to G_2, no graph can be isomorphic to both input graphs (i.e., both to G_1 and to G_2). In this case the graph G' sent in Step (V1) uniquely determines the bit σ. On the other hand, if G_1 and G_2 are isomorphic then, for every G' sent in Step (V1), 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 σ (chosen by the verifier), and so no prover may convince the verifier with probability exceeding 1/2.

summation is taken over all 0-1 assignments to its 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.* Thus, proving that the original Boolean formula is non-satisfiable reduces to proving that the corresponding arithmetic summation evaluates to 0. We highlight two additional observations regarding the resulting arithmetic expression:

1. The arithmetic expression is a low degree polynomial over the integers; specifically, its (total) degree equals the number of clauses in the original Boolean formula.

2. 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. Thus, summing over all 2^n possible 0-1 assignments, where $n \leq vm$ is the number of variables, the result resides in $[0, 2^n v^m]$.

Moving to a Finite Field: Whenever we need to check equality between two integers in $[0, M]$, it suffices to check their 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 an 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. In the rest of this exposition, all arithmetic operations refer to the finite field of q elements, denoted GF(q).

Overview of the actual protocol: stripping summations in iterations. Given a formal expression as in Eq. (9.1), we strip off summations in iterations, stripping a single summation at each iteration, and instantiate the corresponding free variable as follows. At the beginning of each iteration the prover is supposed to supply the univariate polynomial representing the residual expression as a function of the (single) currently stripped variable. (By Observation 1, this is a low degree polynomial and so it has a short description.)[6] The verifier checks that the

[6]We also use Observation 2, which implies that we may use a finite field with elements having a description length that is polynomial in the length of the original Boolean formula (i.e., $\log_2 q = O(vm)$).

polynomial (say, p) is of low degree, and that it corresponds to the current value (say, v) being claimed (i.e., it verifies that $p(0)+p(1) \equiv v$). Next, the verifier randomly instantiates the currently free variable (i.e., it selects uniformly $r \in \mathrm{GF}(q)$), 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 parties proceed to the next iteration (which refers to the residual expression and to the new 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 (detailed): 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

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

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

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.

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 (because $p(r) \not\equiv p'(r) \pmod q$, whereas the new value is set to $p'(r) \bmod q$ and the residual expression evaluates to $p(r)$). 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, a similar proof system (which uses a related arithmetization – see Exercise 9.4) can be used to prove that a given formula has a given number of satisfying assignment; i.e., prove membership in the ("counting") set

$$\{(\phi,k) : |\{\tau : \phi(\tau)=1\}| = k\}. \tag{9.5}$$

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).[7] 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 (integral) 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 fact (to be established in Exercise 9.6) that if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3, \text{poly}(\log M)]$. 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

[7] Actually, the following extension of the foregoing proof system yields a proof system for the set of *unsatisfied* Quantified Boolean Formulae (which is also complete for $\mathcal{PSPACE}$). Alternatively, one may extend the related proof system presented in Exercise 9.4.

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 9.5). 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 shall 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[8] 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 for every $i \in \{1, ..., t\}$ it holds that $\beta_i = V(x, r, \alpha_1, ..., \alpha_i)$.
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 for every $i \in \{t+1, ..., T-1\}$ it holds that $\alpha_{i+1} = f(x, \gamma, m, \beta_{t+1}, ..., \alpha_i, \beta_i)$ and $\beta_i = V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_i)$.

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, the value $f(x,\gamma)$ 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. $\square$

[8]For sake of convenience, when describing the strategy f, we refer to the entire partial transcript of the interaction with V (rather than merely to the sequence of previous messages sent by V).

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 the verifier's 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). Thus, the verifier's past coin tosses are not necessarily revealed by the messages that it sends. In contrast, in public-coin proof systems (a.k.a Arthur-Merlin proof systems), the verifier's messages contain the outcome of any coin that it tosses at the current round. Thus, these messages reveal the randomness used towards generating them (i.e., this randomness becomes public). Actually, without loss of generality, the verifier's messages can be identical to the outcome of the coins tossed at the current round (because any other string that the verifier may compute based on these coin tosses is actually determined by them). 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).[9]

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.*[10]
- *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 Appendix F.2 (or [21] 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 [160]).
- 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.[11] 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:

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

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

[11] 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.

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 [21]): $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f+1)$.

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 $\mathcal{AM}(2)$ 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 9.7).

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 potential 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 (prescribed) 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 prescribed 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, which are briefly surveyed next.

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. Formally:

> 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 foregoing 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.

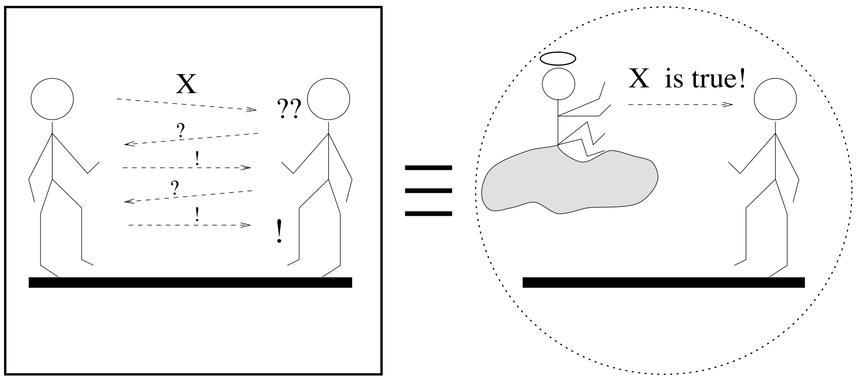


Figure 9.1: Zero-knowledge proofs – an illustration.

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 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.[12] 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 context of zero-knowledge, a benign behavior is any computation that is based (only) on the assertion itself (while assuming that the latter is valid). Thus, a zero-knowledge proof is an interactive proof in which no feasible adversarial verifier strategy can obtain from the interaction 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 refers to two types of probability ensembles, where each ensemble associates a single probability distribution to each relevant input (e.g., a 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 (on some common input), where

[12]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 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 is only given the corresponding input but 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. 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):[13] *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*

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

[13]In the actual definition one relaxes the requirement in one of the following two ways. The first alternative is allowing M^* to run for *expected* (rather than strict) polynomial-time. The second alternative consists of allowing M^* to have no output with probability at most $1/2$ and considering the value of its output conditioned on it having output at all. The latter alternative implies the former, but the converse is not known to hold.

is a negligible function.[14] *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, which is presented in Appendix C.4.2. 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 value of an arbitrary (poly($|x|$)-bit long) auxiliary input). Other definitional issues and related notions are briefly discussed in Appendix C.4.4.

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 (see Exercise 9.9). The same holds for deterministic provers, provided that 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 (see Exercise 9.10). Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For further details, see [87, Sec. 4.5.1].)

Advanced Comment: Knowledge Complexity. Zero-knowledge is the lowest level of a knowledge-complexity hierarchy which 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).[15] 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 [219].) 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, chal-

[14]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 $d(n) < 1/p(n)$). Needless to say, $d(n) \stackrel{\text{def}}{=} 0$ if $S \cap \{0,1\}^n = \emptyset$.

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

lenged 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 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 supplies the isomorphism) may not be zero-knowledge. Furthermore, assuming that Graph Isomorphism is not in $\mathcal{BPP}$, this set has no zero-knowledge NP-proof system, but as we shall shortly see it does have a zero-knowledge interactive proof system.

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*

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

  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 $\pi \circ \phi(v) \overset{\text{def}}{=} \pi(\phi(v))$) 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 (adversarially) 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. Thus, this simulator must depend on the corresponding verifier strategy, and indeed we shall describe the simulator while referring to such a generic verifier strategy. Formally, this means that the simulator's program incorporates the program of the corresponding 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 will later select $\sigma = \tau$), 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).[16] Thus,

[16]Indeed, this observation is identical to the one made in the analysis of the soundness of Construction 9.3.

the value σ selected by the adversarial verifier may depend on G' but not on σ, which implies that $\Pr[\sigma = \tau] = 1/2$. In other words, the simulator's guess (i.e., τ) is correct (i.e., equals σ) with probability 1/2. Now, if the guess is correct then the simulator can produce an output that has the correct distribution, and 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 deterministic polynomial-time algorithm with an auxiliary input).[17]

   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 the messages that they have received). In other words, it suffices to simulate the view 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[18] 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. 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 zero-knowledge proof system presented in Construction 9.9 refers to one specific NP-set that is not known to be in $\mathcal{BPP}$. It turns out that, under reasonable

[17]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).

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

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.

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 $\mathcal{NP} \subseteq \mathcal{BPP}$). In fact, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, graph 3-colorability has no zero-knowledge NP-proof system, but as we shall shortly see it does have a zero-knowledge interactive proof system. This interactive proof system 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, e.g., Theorem 9.11).

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.*

- 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 to 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 verifier 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 with respect to 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 protocol 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 in that case 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 (corresponding 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).[19] Then, we hand all boxes to the verifier, which asks us to open a pair of boxes (corresponding to an edge). If the verifier asks for the pair that we chose (i.e., our guess is successful), then we can complete the simulation by opening these boxes. Otherwise, we try again (with a new random guess and random colors). 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 can derive (from Construction 9.10) zero-knowledge proof sys-

[19] 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.

tems for any NP-set.[20] Furthermore, NP-witnesses can be efficiently transformed into polynomial-size circuits that implement the corresponding (prescribed zero-knowledge) prover strategies.

Theorem 9.11 (The ZK Theorem): *Assuming the existence of* (non-uniformly hard) *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 of Theorem 9.11 (i.e., the existence of one-way functions) 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.)

Theorem 9.12 (The ultimate ZK Theorem): *Assuming the existence of* (non-uniformly hard) *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 for proving 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 (see the proof of Theorem 9.4), probabilistically checkable proofs (see the proof of Theorem 9.16), and the "hardness versus randomness paradigm" (see, e.g., [125]).

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 $\mathcal{IP}(2) \cap \text{co}\mathcal{IP}(2)$, and thus are unlikely to exist for all NP-sets. On the other

[20] 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).

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 [218].

9.2.3 Proofs of Knowledge – a parenthetical subsection

Teaching note: Technically speaking, this topic belongs to Section 9.1, but its more interesting demonstrations refer to zero-knowledge proofs of knowledge – hence its current positioning.

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.

Technically speaking, using a machine as a black-box seems more appealing when the said machine is interactive (i.e., implements an interactive strategy). Indeed, this will be our focus here. Furthermore, conceptually speaking, whatever a machine knows (or does not know) is its own business, whereas 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 extractor 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 convinced her that he knows this coloring.*[21] 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"[22] 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 behavior, 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 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* (non-uniformly hard) *one-way functions, any NP-relation has a zero-knowledge proof of knowledge* (of a corresponding NP-witnesses). *Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given such an NP-witness.*

[21] For simplicity, the reader may consider graphs that have a unique 3-coloring (upto a relabeling). In general, we refer here to instances that have unique solution (cf. Section 6.2.3), which arise naturally in some (cryptographic) applications.

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

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. However, 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 inapproximability 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 Part (1) of the following definition. Yet, the complexity measures introduced in Part (2) are of key importance for the subsequent discussions.

Definition 9.14 (Probabilistically Checkable Proofs – PCP):

1. *A* probabilistically 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 probabilistically 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.*[23] *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 probabilistically checkable proof systems of randomness complexity r and query complexity q. For sets of integer functions, R and Q,

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

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.

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.

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

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 9.11). 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 9.13).

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 9.12). Thus, $\mathcal{PCP}(r,q) \subseteq \text{DTIME}(2^{2^r q+r} \cdot \text{poly})$, 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 for $2^{r(n)}q(n) \leq n$ (or for any other fixed polynomial that bounds $2^r q$) the class $\mathcal{PCP}(r,q)$ is unlikely to contain $\mathcal{NP}$. Actually, $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{NP}$ is not contained in $\mathcal{PCP}(o(\log), o(\log))$ (see Exercise 9.15).

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 observations 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 9.12):

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

$$\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q) \tag{9.6}$$

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 one is willing to tolerate an error probability of ε then it suffices to 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. (9.6). 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 every 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 every 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. We first note that 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 (see Exercise 2.25).[24] For inputs consisting of quadratic equations with n variables, the oracle (of this PCP) is supposed to provide the values of all quadratic expressions in these n variables evaluated at some fixed assignment to these variables. This assignment is supposed to satisfy the system of quadratic equations that is given as input. 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 a quadratic expression obtained by a random linear combination of the quadratic expressions that appear in the quadratic system 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 regarding some satisfying assignment $\tau = \tau_1 \cdots \tau_n$ (also viewed as an n-ary vector over GF(2)). The first part, denoted T_1, is supposed to provide a Hadamard encoding of the said satisfying assignment; that is, for every $\alpha \in \mathrm{GF}(2)^n$ this table is supposed to provide the inner product mod 2 of the n-ary vectors α and τ (i.e., $T_1(\alpha)$ is supposed to equal $\sum_{i=1}^n \alpha_i \tau_i$). The second part, denoted T_2, is supposed to provide all linear combinations of the values of the $\tau_i \tau_j$'s; that is, for every $\beta \in \mathrm{GF}(2)^{n^2}$ (viewed as an n-by-n matrix over GF(2)), the value of $T_2(\beta)$ is supposed to equal $\sum_{i,j} \beta_{i,j} \tau_i \tau_j$. (Indeed T_1 is contained in T_2, because $\sigma^2 = \sigma$ for any $\sigma \in \mathrm{GF}(2)$.) The PCP verifier will use the two tables for checking 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.

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

Testing the Hadamard Code. Note that T_1 is supposed to encode a linear function; that is, there must be some $\tau = \tau_1 \cdots \tau_n \in \mathrm{GF}(2)^n$ such that $T_1(\alpha) = \sum_{i=1}^n \tau_i \alpha_i$ holds for every $\alpha = \alpha_1 \cdots \alpha_n \in \mathrm{GF}(2)^n$. This can be tested by selecting uniformly $\alpha', \alpha'' \in \mathrm{GF}(2)^n$ and checking whether $T_1(\alpha') + T_1(\alpha'') = T_1(\alpha' + \alpha'')$, where $\alpha' + \alpha''$ 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 0.02-far from being linear is rejected with probability at least 0.01 (see Exercise 9.16), 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 0.02-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 0.02-far from being a Hadamard encoding of any $\tau \in \mathrm{GF}(2)^n$, and likewise rejects any T_2 that is 0.02-far from being a Hadamard encoding of any $\tau' \in \mathrm{GF}(2)^{n^2}$. We may thus assume that T_1 (resp., T_2) is 0.02-close to the Hadamard encoding of some τ (resp., τ'). (This does not mean, however, that τ' equals the outer produce of τ with itself.)

In the rest of the analysis, we fix $\tau \in \mathrm{GF}(2)^n$ and $\tau' \in \mathrm{GF}(2)^{n^2}$, and denote the Hadamard encoding of τ (resp., τ') by $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (resp., $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$). Recall that T_1 (resp., T_2) is 0.02-close to f_τ (resp., $f_{\tau'}$).

Self-correction of the Hadamard Code. Suppose that T is ε-close to a linear function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (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.)[25] Thus, assuming that T_1 is 0.02-close to f_τ (resp., T_2 is 0.02-close to $f_{\tau'}$) we may correctly recover (i.e., with error probability 0.02) the value of f_τ (resp., $f_{\tau'}$) at any desired point by making 2 queries to T_1 (resp., T_2).

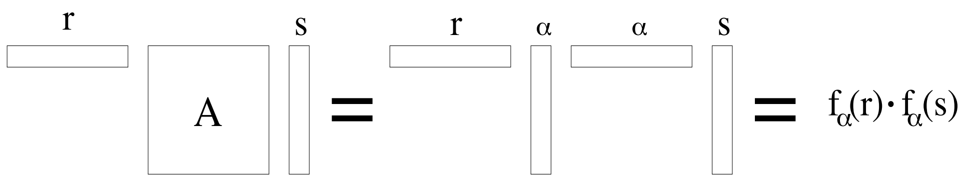


Figure 9.2: Detail for testing consistency of linear and quadratic forms.

[25] 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.

Checking consistency of f_τ and $f_{\tau'}$. Suppose that we are given access to $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$, where $f_\tau(\alpha) = \sum_i \tau_i \alpha_i$ and $f_{\tau'}(\alpha') = \sum_{i,j} \tau'_{i,j}\alpha'_{i,j}$, and that we wish to verify that $\tau'_{i,j} = \tau_i\tau_j$ for every $i,j \in \{1,...,n\}$. In other words, we are given a (somewhat weird) encoding of two matrices, $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$, and we wish to check whether or not these matrices are identical. It can be shown (see Exercise 9.18) 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 uniformly distributed n-ary vectors. Note that, in our case (where $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$), it holds that $r^\top As = \sum_j(\sum_i r_i\tau_i\tau_j)s_j = f_\tau(r)f_\tau(s)$ (see Figure 9.2) and $r^\top A's = \sum_j(\sum_i r_i\tau'_{i,j})s_j = f_{\tau'}(rs^\top)$, where $rs^\top$ is the outer-product of s and r. Thus, (for $(\tau_i\tau_j)_{i,j} \neq (\tau'_{i,j})_{i,j}$) we have $\mathsf{Pr}_{r,s}[f_\tau(r)f_\tau(s) \neq f_{\tau'}(rs^\top)] \geq 1/4$. Using self-correction (to obtain the desired value of $f_{\tau'}$ at $rs^\top$, since $rs^\top$ is not uniformly distributed in $\mathrm{GF}(2)^{n^2}$), we test the consistency of f_τ and $f_{\tau'}$; that is, we select uniformly $r, s \in \mathrm{GF}(2)^n$ and $R \in \mathrm{GF}(2)^{n^2}$ and check that $T_1(r)T_1(s) = T_2(rs^\top + R) - T_2(R)$.

By repeating the consistency test for a constant number of times, we may reject an inconsistent pair of tables with probability at least 0.99. Thus, in the rest of the analysis, we may assume that $(\tau_i\tau_j)_{i,j} = (\tau'_{i,j})_{i,j}$.

Checking that τ satisfies the quadratic system. Suppose that we are given access to f_τ and $f_{\tau'}$ as in the foregoing (where, in particular, $\tau' = \tau\tau^\top$). A key observation is that if τ 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 τ 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 $f_{\tau'}$ at a single point (which corresponds to the quadratic equation). That is, to test whether τ satisfies the quadratic equation $Q(x) = \sigma$, we test whether $f_{\tau'}(Q) = \sigma$. 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 (and linear in the number of equations). If the quadratic system is satisfiable (by some τ), then the verifier accepts the corresponding tables T_1 and T_2 (i.e., $T_1 = f_\tau$ and $T_2 = f_{\tau\tau^\top}$) 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 original proof of the PCP Theorem (Theorem 9.16) consists of three main conceptual steps, *which we briefly sketch first and further discuss later.*

1. Constructing a (non-adaptive) PCP system for $\mathcal{NP}$ having *logarithmic randomness and polylogarithmic query complexity*; that is, this PCP has the desired randomness complexity and a very low (but non-constant) query com-

plexity. Furthermore, this proof system has additional properties that enable proof composition as in the following Step (3).

2. Constructing a PCP system for $\mathcal{NP}$ having *polynomial randomness and constant query complexity*; that is, this PCP has the desired (constant) query complexity but its randomness complexity is prohibitingly high. (Indeed, we showed such a construction in §9.3.2.1.) Furthermore, this proof system too has additional properties enabling proof composition as in Step (3).

3. The proof composition paradigm:[26] In general, this paradigm allows to compose two proof systems such that the "inner" one is used for probabilistically verifying the acceptance criteria of the "outer" verifier. The aim is to conduct this ("composed") 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 PCP system of Step (2), yields a PCP system (for

[26]Our presentation of the composition paradigm follows [33], rather than the original presentation of [15, 14].

$\mathcal{NP}$) 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 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 that is used (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 verifier is given oracle access to two oracles, one representing an input and the other being an alleged proof. Typically, the query complexity of this verifier is lower than the length of the input oracle, and hence this verifier cannot afford reading 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 is only required to 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[27] and the verifier needs to determine whether the assignment appearing in the 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 this value to the self-corrected value 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-

[27]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 involves many technical details. Furthermore, obtaining a robust version of this PCP is beyond the scope of the current text. Thus, the following description should be viewed 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 an arithmetization of CNF formulae, which is different from the one used in §9.1.2.2 (for constructing an interactive proof system for co$\mathcal{NP}$). In the current 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 $\beta_i = (\gamma_i,\sigma_i)$ that is viewed as a variable name augmented by its sign. Thus, for every $\alpha \in \{0,1\}^{\log|\phi|}$ there is a unique $(\beta_1,\beta_2,\beta_3) \in \{0,1\}^{3\log 2n}$ such that $C_\phi(\alpha,\beta_1,\beta_2,\beta_3) = 1$ holds. Next, 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

$$\sum_{\beta_1\beta_2\beta_3 \in \{0,1\}^{3\log 2n}} \Phi(\alpha,\beta_1,\beta_2,\beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \tag{9.7}$$

where $A'(\beta)$ is the value of the β^{th} literal under the (variable) assignment A; that is, for $\beta = (\gamma,\sigma)$, where $\gamma \in \{0,1\}^{\log n}$ is a variable name and $\sigma \in \{0,1\}$ is the literal's type, it holds that $A'(\beta) = \sigma \cdot A(\gamma) + (1-\sigma)\cdot(1 - A(\gamma))$. Thus, Eq. (9.7) holds if and only if the α^{th} clause is satisfied by the assignment induced by A (because $A'(\beta) = 1$ must hold for at least one of the three literals β that appear in this clause).[28] Note that, as in §9.3.2.1, we cannot afford to verify all n instances of Eq. (9.7). Furthermore, unlike in §9.3.2.1, we cannot afford to take a random linear

[28]Note that, for this α there exists a unique triple $(\beta_1,\beta_2,\beta_3) \in \{0,1\}^{3\log 2n}$ such that $\Phi(\alpha,\beta_1,\beta_2,\beta_3) \neq 0$. This triple $(\beta_1,\beta_2,\beta_3)$ encodes the literals appearing in the α^{th} clause, and this clause is satisfied by A if and only if $\exists i \in [3]$ s.t. $A'(\beta_i) = 1$.

combination of these n instances either (because this requires too much randomness). Fortunately, taking a "pseudorandom" linear combination of these equations is good enough. Specifically, using an adequate (efficiently constructible) small-bias probability space (cf. §8.5.2.3) will do. Denoting such a space (of size $\text{poly}(|\phi|\cdot|F|)$ and bias at most $1/6$) by $S \subset \mathrm{F}^{|\phi|}$, we may select uniformly $(s_1,...,s_{|\phi|}) \in S$ and check whether

$$\sum_{\alpha\beta_1\beta_2\beta_3\in\{0,1\}^\ell} s_\alpha \cdot \Phi(\alpha,\beta_1,\beta_2,\beta_3)\cdot\prod_{i=1}^{3}(1-A'(\beta_i)) = 0 \tag{9.8}$$

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. (9.7) then, with probability at least $1/3$ (taken over the choice of $(s_1,...,s_{|\phi|}) \in S$), it is the case that A fails to satisfy Eq. (9.8). 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. Thus, we have reduced the original problem to checking whether, for a random $(s_1,...,s_{|\phi|}) \in S$, Eq. (9.8) holds.

Assuming (for a moment) that A is a low-degree polynomial, we can probabilistically verify Eq. (9.8) by applying a summation test (as in the interactive proof for co$\mathcal{NP}$). Indeed, the verifier obtains the relevant univariate 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. The 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)). Soundness of the summation test follows by setting $|F| \gg O(\ell)$. 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 $\text{poly}(\log(n))$ 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 $\text{poly}(\log(n))$ 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. [14]), 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 evident robustness of a simple 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 has the desired complexities but rejects false assertions with probability inversely proportional to their length, and double the rejection probability in each step while essentially maintaining the initial complexities. That is, in each step, the constant query complexity of the verifier is preserved and its randomness complexity is increased only by a constant term. 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 iterated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labeled) 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* $\mathtt{vlt}(G,\Phi)$ (standing for violation) *the fraction of unsatisfied constraints under the best possible assignment; that is,*

$$\mathtt{vlt}(G,\Phi) = \min_{\alpha:V\to\Sigma}\{|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u),\alpha(v)) = 0\}|/|E|\}. \qquad (9.9)$$

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

Note that $\mathtt{3SAT}$ is reducible to $\mathsf{gapCSP}^{\{1,\ldots,7\}}_{\tau}$ for $\tau(m) = 1/m$; see Exercise 9.19. Our goal is to reduce $\mathtt{3SAT}$ (or rather $\mathsf{gapCSP}^{\{1,\ldots,7\}}_{\tau}$) to $\mathsf{gapCSP}^{\Sigma}_{c}$, for some fixed finite Σ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathsf{gapCSP}^{\Sigma}_{c}$; see Exercise 9.21. (The relationship between constraint satisfaction problems and the PCP Theorem is further discussed in Section 9.3.3.) The desired reduction of $\mathsf{gapCSP}^{\Sigma}_{1/m}$ to $\mathsf{gapCSP}^{\Sigma}_{\Omega(1)}$ 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* gapCSP^{Σ} *to itself such that the following conditions hold with respect to the mapping of any instance* (G,Φ) *to the instance* (G',Φ')*.*

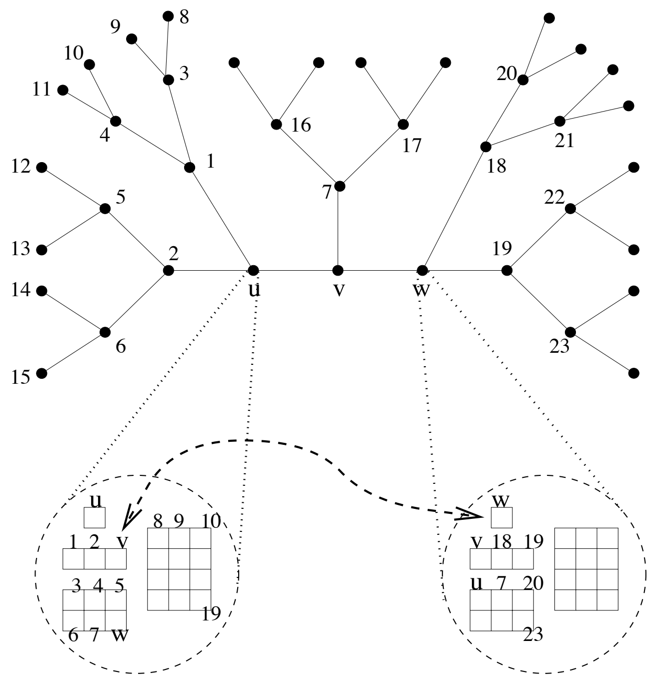
1. *If* $\mathtt{vlt}(G,\Phi) = 0$ *then* $\mathtt{vlt}(G',\Phi') = 0$*.*
2. $\mathtt{vlt}(G',\Phi') \geq \min(2 \cdot \mathtt{vlt}(G,\Phi), c)$*.*
3. $|G'| = O(|G|)$*.*

Proof Outline:[29] The reduction consists of three steps. We first apply a preprocessing step that makes the underlying graph suitable for further analysis. The value of $\mathtt{vlt}$ may decrease during this step by a constant factor. The heart of the reduction is the second step in which we may increase $\mathtt{vlt}$ 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 stress that the aforementioned Σ and c, as well as the auxiliary parameters d and t (to be introduced in the following two paragraphs), are fixed constants that will be determined such that various conditions (which arise in the course of our argument) are satisfied. Specifically, t will be the last parameter to be determined (and it will be made greater than a constant that is determined by all the other parameters).

[29] For details, see [63].

We start with the pre-processing step. Our aim in this step is to reduce the input (G, Φ) of $\texttt{gapCSP}^{\Sigma}$ to an instance (G_1, Φ_1) such that G_1 is a d-regular expander graph.[30] Furthermore, each vertex in G_1 will have at least $d/2$ self-loops, the number of edges is preserved up to a constant factor (i.e., $|G_1| = O(|G|)$), and $\texttt{vlt}(G_1, \Phi_1) = \Theta(\texttt{vlt}(G, \Phi))$. This step is quite simple: see Exercise 9.22. Intuitively, with respect to intersecting a fixed set of edges, a random (t-edge long) walk on the resulting graph G_1 behave like a random sample of (t) edges, while $|G_1| = O(|G|)$ and $\texttt{vlt}(G_1, \Phi_1) = \Omega(\texttt{vlt}(G, \Phi))$.



The alphabet Σ' as a labeling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown (and so the "effective" degree is three). 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.

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

[30] 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 absolute value that is bounded away from d. For further details, see §E.2.1.1.

$\texttt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\texttt{gapCSP}^{\Sigma'}$ 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-labeling of the ("distance $\leq t$") neighborhood of a vertex (see Figure 9.3), and checking that two such labelings 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_2 between vertices u and w, and the constraint associated with it with mandates that the entries corresponding to vertex v in the Σ'-labeling 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'-labeling to the vertices of G_2 must either be consistent with a Σ-labeling 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-constraints (of Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated (by virtue of the latter enforcing many edge-constraints of Φ_1). The point is that *any 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 would have been obvious if G_1 were a complete graph, but it also holds for an expander.)[31]

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 $\texttt{vlt}$). However, so far we obtained an instance of $\texttt{gapCSP}^{\Sigma'}$ rather than an instance of $\texttt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\texttt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\texttt{gapCSP}^{\Sigma'}$ as a (weak) PCP system (analogously to Exercise 9.21), 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 verifier 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 $\Omega(\texttt{vlt}(G_2, \Phi_2))$, which is independent of the choice of the constant t. As in Exercise 9.19, for $\Sigma =$

[31]Indeed, the amplification step may be viewed as a randomness-efficient version of a straightforward repetition procedure, where the sample points are generated by an expander walk (as in Section 8.5.3). In the current context, the saving in randomness translates to a relatively moderate increase in the size of the graph (i.e., the number of edges). We also note that, due to a technical difficulty, it is easier to establish the claimed bound of $\Omega(\sqrt{t} \cdot \texttt{vlt}(G_1, \Phi_1))$ rather than $\Omega(t \cdot \texttt{vlt}(G_1, \Phi_1))$.

$\{0,1\}^{O(1)}$, we can easily obtain an instance of $\texttt{gapCSP}^{\Sigma}$ that has a $\Omega(\texttt{vlt}(G_2,\Phi_2))$ 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. $\square$

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 $\max_{\pi}\{\mathsf{Pr}[V^{\pi}(x)=1]\}$). 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.*

Note that the foregoing approximation problem is parameterized by a PCP verifier V, and its instances are given their value with respect to this verifier (i.e., the instance x has value $\max_{\pi}\{\mathsf{Pr}[V^{\pi}(x)=1]\}$). This per se does not yield a "natural" approximation problem. In order to link PCP systems with natural approximation problems, we take a closer look at the approximation problem associated with $\mathcal{PCP}(r,q)$. For simplicity, we focus on the case of non-adaptive PCP systems (i.e., all the queries are determined beforehand based on the input and the internal coin tosses of the verifier). Fixing an input x for such a system, we consider the $2^{r(|x|)}$ formulae that represent the decision of the verifier on each of the possible outcomes of its coin tosses after inspecting the corresponding bits in the proof oracle. That is, each of these $2^{r(|x|)}$ formulae depends 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 truth assignment that satisfies more than $2^{r(|x|)-1}$ formulae. Furthermore, in the case that $r(n) = O(\log n)$, given x, we can construct the corresponding sequence of formulae in polynomial-time. Hence, the PCP Theorem (i.e., Theorem 9.16) yields *NP-hardness results regarding the approximation of the number of simultaneously satisfiable Boolean 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 $\texttt{gapGSAT}^q_{\varepsilon}$ 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 $\texttt{gapSAT}^q_{\varepsilon}$ 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 $\texttt{gapSAT}^q_\varepsilon$ 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 $\texttt{gapGSAT}^{O(1)}_{1/2}$ is NP-complete, which in turn implies that for some constant $\varepsilon > 0$ the problem $\texttt{gapSAT}^3_\varepsilon$ is NP-complete. The converses hold too. All these claims are stated and proved next.

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* $\texttt{gapGSAT}^q_{1/2}$ *is $\mathcal{NP}$-hard.*
3. *There exists a constant $\varepsilon > 0$ such that* $\texttt{gapSAT}^3_\varepsilon$ *is $\mathcal{NP}$-hard.*

The point of Theorem 9.21 is not its mere validity (which follows from the validity of each of the three items), but rather the fact that its proof is quite simple. Note that Items 2 and 3 make no reference to PCP. Thus, their (easy to establish) equivalence to Item 1 manifests that the hardness of approximating natural optimization problems lies at the heart of the PCP Theorem. In general, probabilistically checkable proof systems for $\mathcal{NP}$ yield strong inapproximability results for various classical optimization problems (cf., Exercise 9.14 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 scan all $2^{O(\log |x|)}$ possible sequence of outcomes of the verifier's coin tosses, and for each such sequence of outcomes we determine the queries made by the verifier as well as the residual decision predicate (where this predicate determines which sequences of answers lead this verifier to accept). That is, for each random-outcome $\omega \in \{0,1\}^{O(\log |x|)}$, we consider the residual predicate, determined by x and ω, that specifies which q-bit long sequence of oracle answers makes the verifier accept x on coins ω. Indeed, this predicate depends only on q variables (which represent the values of the q corresponding oracle answers). Thus, we map x to a sequence of poly($|x|$) formulae, each depending on q variables, obtaining an instance of $\texttt{gapGSAT}^q$. This mapping can be computed in polynomial-time, and indeed $x \in S$ (resp., $x \notin S$) is mapped to a yes-instance (resp., no-instance) of $\texttt{gapGSAT}^q_{1/2}$.

Item 2 implies Item 3 by a standard reduction of GSAT to 3SAT. Specifically, $\texttt{gapGSAT}^q_{1/2}$ reduces to $\texttt{gapSAT}^q_{2^{-(q+1)}}$, which in turn reduces to $\texttt{gapSAT}^3_\varepsilon$ for $\varepsilon = 2^{-(q+1)}/(q-2)$. Note that Item 3 implies Item 2 (e.g., given an instance of $\texttt{gapSAT}^3_\varepsilon$, consider all possible conjunctions of $1/\varepsilon$ disjunctive clauses in the given instance).

We complete the proof by showing that Item 3 implies Item 1. (The same proof shows that Item 2 implies Item 1.) This is done by showing that $\texttt{gapSAT}^3_\varepsilon$ is in $\mathcal{PCP}(O(\varepsilon^{-1}\log), O(1/\varepsilon))$, and using the reduction of $\mathcal{NP}$ to $\texttt{gapSAT}^3_\varepsilon$ to derive a corresponding PCP for each set in $\mathcal{NP}$. In fact, we show that $\texttt{gapGSAT}^q_\varepsilon$ 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. Needless to say, the latter systems have super-logarithmic randomness complexity.

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 exists a logarithmic function ℓ such that, for every integer function k that satisfies $0 \leq k(n) \leq \ell(n)$, it holds that $S \in \mathcal{PCP}(\ell - k, O(2^k)) \subseteq \mathcal{NP}$.*

Proof Sketch: By Theorem 9.16, we have $S \in \mathcal{PCP}(\ell, O(1))$. Consider an emulation of the corresponding verifier in which we try all possibilities for the $k(n)$-bit long prefix of its random-tape. Lastly, recall that $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$. □

Following the establishment of Theorem 9.16, numerous variants of the PCP Characterization of NP were explored. These variants refer to a finer evaluation of various parameters of probabilistically checkable proof systems (for sets in $\mathcal{NP}$). Following is a brief summary of some of these studies.[32]

[32]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. Remarkably, *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 a *relatively modest amount of redundancy* in the proof oracle suffices for supporting probabilistic verification via a constant number of probes.

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 of 1/2 (for NP). It is currently known that this constant is at most *five*, whereas with *three* queries one may get arbitrary close to a soundness error of 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. [115]), 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 9.14 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, 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

Our focus in §9.3.4.1 was 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.[33]

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

Recall that $\mathcal{PCP}(r,q) \subseteq \textsc{Ntime}(t)$, for $t(n) = \mathrm{poly}(n) \cdot 2^{r(n)}$. Thus, the NTIME Hierarchy implies a hierarchy of $\mathcal{PCP}(\cdot, O(1))$ classes, for randomness complexity ranging between logarithmic and polynomial functions.

Chapter Notes

(The following historical notes are quite long and still they fail to properly discuss 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 was 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 was pointed out in [105]. Independently of [105], Babai [17] suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games.* Syntactically, Arthur-Merlin Games are a restricted form

[33]This scaling up is not straightforward, since we wish to maintain polynomial-time verification. The key point is that the CNF formulae that represent the computation of NTIME are highly uniform, and thus the corresponding Boolean functions (and their low degree extensions) can be evaluated in polynomial-time.

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 [21] (improving over [17]).

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. Theorem 9.12 (i.e., $\mathcal{ZK} = \mathcal{IP}$) is due to [30, 126].

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 [155], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [197] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai, Fortnow and Lund [18] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 9.4 follows [197].)

The aforementioned multi-prover proof system of Babai, Fortnow and Lund [18] (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 [19] 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.* [19] 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.* [19] 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 refined (quantitative) version of the notion of probabilistically checkable proof systems (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 [18] can be reinterpreted as establishing $\mathcal{NEXP} = \mathcal{PCP}(\texttt{poly}, \texttt{poly})$, the work of Feige *et. al.* [70] establishes $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, where $f(n) = O(\log n \cdot \log\log n)$. (In retrospect, we note that the work of Babai *et. al.* [19] implies that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, \texttt{polylog})$, but the latter terminology was not available at the time.)

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 $\mathcal{PCP}(\cdot, \cdot)$. The obvious challenge was showing that $\mathcal{NP}$ equals $\mathcal{PCP}(\texttt{log}, \texttt{log})$. This challenge was met by Arora and Safra [15]. Actually, they showed that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, q)$, where $q(n) = o(\log n)$.

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 [14], and is captured by the PCP Characterization Theorem, which asserts that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$.

Indeed the PCP Characterization Theorem is a culmination of a sequence of impressive works [155, 18, 19, 70, 15, 14]. 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).

Our overview of the original proof of the PCP Theorem (in §9.3.2.1–9.3.2.2) is based on [14, 15].[34] 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 even stronger variants of the PCP Theorem (which are surveyed in §9.3.4.1). These include the Parallel Repetition Theorem [178], the use of the Long-Code [27], and the application of Fourier analysis in this setting [112, 113].

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 later, Kilian [139] 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

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

communication complexities are poly-logarithmic.[35] Interestingly, these argument systems rely on the fact that $\mathcal{NP} \subseteq \mathcal{PCP}(f,f)$, for $f(n) = \text{poly}(\log n)$. We mention that Micali [158] 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 [115] (for a study of their 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 behaving 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.

Guideline: Modify the (first part of the) proof of Theorem 9.4, by stripping $O(\log n)$ summations in each round.

Exercise 9.4 (an interactive proof system for $\#\mathcal{P}$) Using the main part of the proof of Theorem 9.4, present a proof system for the counting set of Eq. (9.5).

Guideline: Use a slightly different arithmetization of CNF formulae. Specifically, instead of replacing the clause $x \vee \neg y \vee z$ by the term $(x + (1-y) + z)$, replace it by the term $(1 - ((1-x) \cdot y \cdot (1-z)))$.

Exercise 9.5 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.

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

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

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

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).[36]

Exercise 9.6 Prove that if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3,L]$, where $L = \mathrm{poly}(\log M)]$. Prove the same for the interval $[L,2L]$.

Guideline: Let $a \neq b \in [0,M]$ and let $P_1,...,P_t$ be an enumeration of the primes in the interval $[3,\mathrm{poly}(\log M)]$ such that for every $i = 1,...,t$ it holds that $a \equiv b \pmod{P_i}$. Using the Chinese Reminder Theorem, prove that $Q \stackrel{\text{def}}{=} \prod_{i=1}^{t} P_i \leq M$ (because otherwise $a = b$ follows by combining $a \equiv b \pmod{Q}$ with the hypothesis $a,b \in [0,M]$). It follows that $t < \log_2 M$. Using a lower-bound on the density of prime numbers, the claim follows.

Exercise 9.7 (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. Similarly, let $\mathcal{AM}'$ denote the public-coin version of $\mathcal{IP}'$.

1. Establish $\mathcal{IP}'(f) \subseteq \mathcal{AM}'(f+3)$ by noting that the proof of Theorem F.2, which establishes $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$, extends to the two-sided error setting.

2. Prove that $\mathcal{AM}'(f) \subseteq \mathcal{AM}'(f+1)$ by extending the ideas underlying the proof of Theorem 6.9, which actually establishes that $\mathcal{BPP} \subseteq \mathcal{AM}(1)$ (where $\mathcal{BPP} = \mathcal{AM}'(0)$).

[36]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.

Using the Round Speed-up Theorem (i.e., Theorem F.3), conclude that, for every function $f : \mathbb{N} \to \mathbb{N} \setminus \{1\}$, it holds that $\mathcal{IP}'(f) = \mathcal{AM}(f) = \mathcal{IP}(f)$.

Guideline: We focus on establishing $\mathcal{AM}'(f) \subseteq \mathcal{AM}(f+1)$ 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 $\mathcal{MA} = \mathcal{AM}(1)$, we obtain the desired result. For further details, see [78].

Exercise 9.8 In continuation to Exercise 9.7, show that $\mathcal{IP}'(f) = \mathcal{IP}(f)$ for every function $f : \mathbb{N} \to \mathbb{N}$ (including $f \equiv 1$).

Guideline: Focus on establishing $\mathcal{IP}'(1) = \mathcal{IP}(1)$, which is identical to Part 2 of Exercise 6.12. Note that the relevant classes defined in Exercise 6.12 coincide with $\mathcal{IP}(1)$ and $\mathcal{IP}'(1)$; that is, $\mathcal{MA} = \mathcal{IP}(1)$ and $\mathcal{MA}^{(2)} = \mathcal{IP}'(1)$.

Exercise 9.9 (on the role of soundness error in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with perfect soundness (i.e., soundness error equals zero) then $S \in \mathcal{BPP}$.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier. Consider the algorithm that on input x, accepts x if and only if $M(x)$ represents a valid view of the verifier in an accepting interaction (i.e., an interaction that leads the verifier to accept the common input x). Use the simulation condition to analyze the case $x \in S$, and the perfect soundness hypothesis to analyze the case $x \notin S$.

Exercise 9.10 (on the role of interaction in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with a uni-directional communication then $S \in \mathcal{BPP}$.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier, and let $M'(x)$ denote the part of this view that consists of the prover message. Consider the algorithm that on input x, obtains $m \leftarrow M'(x)$, and emulates the verifier's decision on input x and message m. Note that this algorithm ignores the part of $M(x)$ that represents the verifier's internal coin tosses, and uses fresh verifier's coins when deciding on (x, m).

Exercise 9.11 (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 in the proof oracle that are examined by V 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.)

Exercise 9.12 (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[37] steps and halts within a total number of

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

$2^{r(n)} \cdot \mathrm{poly}(n)$ steps. Thus, $S \in \mathrm{NTIME}(2^r \cdot \mathrm{poly}) \cap \mathrm{DTIME}(2^{2^r q + r} \cdot \mathrm{poly})$.

Guideline: 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 9.13 (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 is given direct access to a non-uniform $\mathrm{poly}(\ell)$-bit long advice.

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 of the said size, 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 9.14 (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). Note that V_x contains only accepting "views" of the verifier. 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 $f \leq q$ 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 $2^{r(|x|)}$.

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.*

Guideline: 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 (having a standard PCP). Larger factors can be obtained by considering a PCP that results from repeating the original PCP for a constant number of times. The result for `Clique` follows by considering the complement graph.

Exercise 9.15 Using the ideas of Exercise 9.14, 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 reduces PCP to instances of the `Clique` problem (and ignore the fact that we actually get a stronger reduction 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. This yields a reduction of `Clique` to a finite set, and $\mathcal{NP} = \mathcal{P}$ follows (by the $\mathcal{NP}$-completeness of `Clique`).

Exercise 9.16 (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., $\delta(f) = \Pr_{x\in G}[f(x)\neq h(x)]$). 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* cases (regarding $f(x) \overset{?}{=} h(x)$, $f(y) \overset{?}{=} h(y)$, and $f(x+y) \overset{?}{=} h(x+y)$) that are possible when $f(x)+f(y)\neq f(x+y)$, where these three cases refer to the disagreement of h and f on exactly one out of the three relevant points.)

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 9.17.

Exercise 9.17 (a better analysis of the BLR Linearity Test (cf. [38])) In continuation to Exercise 9.16, 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 $\phi_y(x) \overset{\text{def}}{=} f(x+y) - f(y)$, and define $\phi(x)$ as the corresponding plurality vote (i.e., $\phi(x) \overset{\text{def}}{=} \text{argmax}_{v \in H}\{|\{y \in G : \phi_y(x) = v\}|\}$).

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

   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 fictitious expression $\Pr_{r \in G}[\phi(x) + \phi(y) \neq \phi(x + y)]$, and showing that it is strictly smaller than 1 (and hence $\phi(x) + \phi(y) \neq \phi(x + y)$ is false). Upper-bound the probabilistic expression by

   $$\Pr_r[\phi(x) \neq f(x + r) - f(r) \vee \phi(y) \neq f(r) - f(r - y) \vee \phi(x + y) \neq f(x + r) - f(r - y)].$$

   Use the union bound (and Item 1), and note that $\Pr_r[\phi(x) \neq f(x + r) - f(r)] < 2\varepsilon(f) < 1/3$, whereas $\Pr_r[\phi(y) \neq f(r) - f(r - y)] = \Pr_{r'}[\phi(y) \neq f(y + r') - f(r')]$ and $\Pr_r[\phi(x + y) \neq f(x + r) - f(r - y)] = \Pr_{r'}[\phi(x + y) \neq f(x + y + r') - f(r')]$ (by substituting $r' = r - y$).

3. Prove that f is $2\varepsilon(f)$-close to ϕ.

   Extra guideline: Denoting $B = \{x \in G : \Pr_{y \in G}[f(x) \neq \phi_y(x)] \geq 1/2\}$, prove that $\varepsilon(f) \geq (1/2) \cdot (|B|/|G|)$. Note that if $x \in G \setminus B$ then $f(x) = \phi(x)$.

We comment that better bounds on the behavior of $\varepsilon(f)$ as a function of $\delta(f)$ are known.

Exercise 9.18 (checking matrix identity) Let M be a non-zero m-by-n matrix over GF(p). Prove that $\Pr_{r,s}[r^\top M s \neq 0] \geq (1 - p^{-1})^2$, where r (resp., s) is a random m-ary (resp., n-ary) vector.

Guideline: Prove that if $v \neq 0^m$ then $\Pr_s[v^\top s = 0] = p^{-1}$, and that if M has rank ρ then $\Pr_r[r^\top M = 0^n] = p^{-\rho}$.

Exercise 9.19 (3SAT and CSP with two variables) Show that 3SAT is reducible to $\texttt{gapCSP}_{\tau}^{\{1,\dots,7\}}$ for $\tau(m) = 1/m$, 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 9.14.

Exercise 9.20 (CSP with two Boolean variables) In contrast to Exercise 9.19, prove that for every positive function $\tau : \mathbb{N} \to (0,1]$ the problem $\texttt{gapCSP}_{\tau}^{\{0,1\}}$ is solvable in polynomial-time.

Guideline: Reduce $\texttt{gapCSP}_{\tau}^{\{0,1\}}$ to 2SAT.

Exercise 9.21 Show that, for any fixed finite Σ and constant $c > 0$, the problem $\texttt{gapCSP}_{c}^{\Sigma}$ is in $\mathcal{PCP}(\texttt{log}, O(1))$.

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. This verifier makes $\log_2 |\Sigma|$ queries and reject each no-instance with probability at least c.

Exercise 9.22 For any constant Σ and $d \geq 14$, show that $\texttt{gapCSP}^{\Sigma}$ 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 $\texttt{vlt}(G_1, \Phi_1) = \Theta(\texttt{vlt}(G, \Phi))$. 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, augment the resulting N_1-vertex graph by the edges of a 3-regular expander of size N_1 (while associating with these edges the trivially satisfied constraint; i.e., $\phi(i,j) = 1$ 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 9.23 (free bit complexity zero) Note that only sets in $\mathcal{BPP}$ have PCPs of *query* complexity zero. Furthermore, Exercise 9.12 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 9.24 (free bit complexity one) In continuation to Exercise 9.23, 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 poly(2^r), and see Exercise 2.22.