# Computational Complexity:

## A Conceptual Perspective

Oded Goldreich

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

May 25, 2008

# List of Figures

# Chapter 9

# Probabilistic Proof Systems

*A proof is whatever convinces me.*

Shimon Even (1935–2004)

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 $\mathcal{NP}$, where efficient computation is associated with deterministic polynomial-time algorithms. However, as argued next, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures.

In this chapter, we shall study three types of probabilistic proof systems, called *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. In each of these three cases, we shall present fascinating results that cannot be obtained when considering the analogous deterministic proof systems.

**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. Indeed, these probabilistic proof systems carry a probability of error (which is explicitly bounded and can be reduced by successive application of the proof system), yet 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 theoretical 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 nothing else (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 are closely related to the study of the complexity of numerous natural approximation problems.

## Introduction and Preliminaries

Conceptually speaking, proofs are secondary to the verification process. Indeed, both in mathematics and in real-life, proofs are meaningful only with respect to commonly agreed principles of reasoning, and the verification process amounts to checking that these principles were properly applied. Thus, these principles, which are typically taken for granted, are more fundamental than any specific proof that applies them; that is, the mere attempt to reason about anything is based on commonly agreed principles of reasoning.

The commonly agreed principles of reasoning are associated with a verification procedure that distinguishes proper applications of these principles from improper ones. A *line of reasoning* is considered valid with respect to such fixed principles (and is thus deemed a proof) if and only if it proceeds by a proper applications of these principles. Thus, a line of reasoning is considered valid if and only if it is accepted by the corresponding verification procedure. This means that, technically speaking, proofs are defined in terms of a predetermined verification procedure (or are define with respect to such a procedure) . Indeed, this state of affairs is best illustrated in the formal study of proofs (i.e., *logic*), which is actually the study of formally defined proof systems: The point is that these proof systems are defined (often explicitly and sometimes only implicitly) in terms of their verification procedures.

The notion of a verification procedure presumes the notion of computation. This fact explains the historical interest of logicians in computer science (cf. [225, 55]). Furthermore, the verification of proofs is supposed to be relatively easy, and hence a natural connection emerges between verification procedures and the notion of efficient computation. This connection was made explicit by complexity theorists, and is captured by the definition of $\mathcal{NP}$ and NP-proof systems (cf. Definition 2.5), which targets all efficient verification procedures.[1]

Recall that Definition 2.5 identifies efficient (verification) procedures with deterministic polynomial-time algorithms, and that it 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).

Indeed, NP-proofs provide the ultimate formulation of efficiently verifiable proofs (i.e., proof systems with efficient verification procedures), provided that one associates efficient procedures with *deterministic* polynomial-time algorithms. However, as we shall see, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* (polynomial-time) algorithms and, in particular, *probabilistic* verification procedures. In particular:

- 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 conceptual 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.

[1] In contrast, traditional proof systems are formulated based on rules of inference that seem natural in the relevant context. The fact that these inference rules yield an efficient verification procedure is merely a consequence of the correspondence between processes that seem natural and efficient computation.

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 proved (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. Indeed, as will become evident, this is *the* natural choice in all the cases that we consider. Note that this convention is consistent with the foregoing discussion of NP-proof systems.[2]

**Notational Conventions.** We denote by `poly` the set of all integer functions that are upper-bounded by a polynomial, and by `log` the set of all integer functions bounded by a logarithmic function (i.e., $f \in \texttt{log}$ if and only if $f(n) = O(\log n)$). All complexity measures mentioned in this chapter 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.

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

## 9.1 Interactive Proof Systems

In light of the growing acceptability of randomized and interactive 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.

[2] Recall that Definition 2.5 refers to polynomial-time verification of alleged proofs, which in turn must have length that is bounded by a polynomial in the length of the assertion.

The foregoing discussion, as well as the definition provided in Section 9.1.2, 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 this issue as well as some other conceptual issues.

### 9.1.1 Motivation and Perspective

We shall discuss the various interpretations given to the notion of a proof in different human contexts, and the attitudes that underly and/or accompany these interpretations. This discussion is aimed at emphasizing that the motivation for the definition of interactive proof systems is not replacing the notion of a mathematical proof, but rather capturing other forms of proofs that are of natural interest. Specifically, we shall contrast "written proofs" with "interactive proofs", highlight the roles of the "prover" and the "verifier" in any proof, and discuss the notions of completeness and soundness which underly any proof. (Some readers may find it useful to return to this section after reading Section 9.1.2.)

#### 9.1.1.1 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 mathematical proofs: these proofs are fixed* (*static*) *objects*.

In contrast, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, in many settings, 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, standing a cross-examination by an opponent, who may ask tough and/or tricky questions, is considered a proof of the facts claimed by the witness. Likewise, various 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 applies even in scientific debates. Needless to say, *a key property of such debates is their interactive* ("dynamic") *nature*. Interestingly, the appealing nature of such "interactive proofs" is reflected in the fact that they are mimicked (in a rigorous manner) in some mathematical *proofs by contradiction*, which emulate an imaginary debate with a potential (generic) skeptic.

Another difference between mathematical proofs and various forms of "daily proofs" is that, while the former aim at certainty, the latter are intended ("only") for establishing claims *beyond any reasonable doubt*. Arguably, an explicitly bounded error probability (as present in our definition of interactive proof systems) is an *extremely strong* form of establishing a claim beyond any reasonable doubt.

We also note that, in mathematics, proofs are often considered more important 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 foregoing set of daily attitudes (rather than the mathematical ones) 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 that may occur in the world, 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 may 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 proved in the traditional mathematical sense.

#### 9.1.1.2 Prover and Verifier

The wide interpretation of the notion of a proof system, which includes interactive processes of verification, calls for the explicit introduction of two interactive players, called the *prover* and the *verifier*. The verifier is the party that employs the verification procedure, which underlies the definition of any proof system, while the prover is the party that tries to convince the verifier. In the context of static (or non-interactive) proofs, the prover is the transcendental entity providing the proof, and thus in this context the prover is often not mentioned at all (when discussing the verification of alleged proofs). Still, explicitly mentioning potential provers may be beneficial even when discussing such static (non-interactive) proofs.

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 should envision 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 that it is not fooled by the prover. (See further discussion in §9.1.1.3.) Note that the verifier is "trusted" to protect its own interests by employing the predetermined verification procedure; indeed, the asymmetry with respect to who we trust is an artifact of our focus on the verification process (or task). In general, each party is trusted to protect its own interests (i.e., the verifier is trusted to protect its own interests), but no party is trusted to protect the interests of the other party (i.e., the prover is not trusted to protect the verifier's interest of not being fooled by the prover).

Another asymmetry between the two parties is that our discussion focuses on the complexity of the verification task and ignores (as a first approximation) the complexity of the proving task (which is only discussed in §9.1.5.1). Note that this asymmetry is reflected in the definition of NP-proof systems; that is, verification is required to be efficient, whereas for sets $\mathcal{NP} \setminus \mathcal{P}$ finding adequate proofs is infeasible. Thus, as a first approximation, we consider the question of what can be efficiently verified when interacting with an arbitrary prover (which may be infinitely powerful). Once this question is resolved, we shall also consider the complexity of the proving task (indeed, see §9.1.5.1).

#### 9.1.1.3 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 proved (while no false statement can be "proved"). 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).

### 9.1.2 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 (whereas no computational restrictions apply to the prover's 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.)

We formalize the interaction between parties by referring to the *strategies* that the parties employ.[3] A strategy for a party is a *function mapping the party's view of the interaction so far to a description of this party's next move*; that is, such a strategy describes (or rather prescribes) 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), *the party's internal coin tosses, and all messages it has received so far.* Note that this formulation presumes (implicitly) 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 move), the corresponding ($t$-round) 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.[4] 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):[5] *An* interactive proof system for a set $S$ *is a two-party game, between a* verifier *executing a* probabilistic polynomial-time strategy, *denoted* $V$, *and a* prover *that executes a* (computationally unbounded) *strategy, denoted* $P$, *satisfying the following two conditions:*

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

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

[3] An alternative formulation refers to the interactive machines that capture the behavior of each of the parties (see, e.g., [91, Sec. 4.2.1.1]). Such an interactive machine invokes the corresponding strategy, while handling the communication with the other party and keeping a record of all messages received so far.

[4] 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 $x$.

[5] 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* $P$ *such that, for every* $x \in S$, *the verifier* $V$ *always accepts after interacting with* $P$ *on common input* $x$.

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 $k = \mathrm{poly}(|x|)$. Variants on the basic definition are discussed in Section 9.1.4.

**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). Thus, the case of zero soundness error reduces to the case of deterministic verifiers.

Now, 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 $x \in S$) can be (pre)determined (without interacting with $V$) *based solely on the common input $x$*.[6] Hence, $x \in S$ 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 (of prover's messages) 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).[7] The foregoing condition can be checked in polynomial-time, and so a "passing sequence" constitutes an NP-witness for $x \in S$. It follows that $S \in \mathcal{NP}$. ■

[6] As usual, we do not care about the complexity of determining such a sequence, since no computational bounds are placed on the prover.

[7] Recall that in the case that $V$ is randomized, its final decision also depends on its internal coin tosses (and not only on the common input and on the sequence of prover's messages). In that case, the verifier's own messages may reveal information about the verifier's internal coin tosses, which in turn may help the prover to answer with convincing messages.

**Reflection.** The moral of the reasoning underlying the proof Proposition 9.2 is that *there is no point to interact with a party whose moves are easily predictable*, because such moves can be determined without any interaction. This moral represents the prover's point of view (regarding interaction with deterministic verifiers). In contrast, even an infinitely powerful party (e.g., a prover) may gain by interacting with an unpredictable party (e.g., a randomized verifier), because this interaction may provide useful information (e.g., information regarding the verifier's coin tosses, which in turn allows the prover to increase its probability of answering convincingly). Furthermore, 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 the case that they are predictable in an information theoretic sense).

## 9.1.3 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.

### 9.1.3.1 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 (e.g., clearly, showing that a particular mapping fails does not suffice, while enumerating all possible mappings (and showing that each fails) does not yield a succinct proof). More generally, it is typically easy to prove the existence of an easily verifiable structure in a 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 a 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, $\phi$, 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 $\phi$, 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 proved 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* $\pi$ *from the set of permutations over the vertex set* $V_\sigma$*. The verifier constructs a graph with vertex set* $V_\sigma$ *and edge set*

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

  *and sends* $(V_\sigma, E)$ *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* $G_\tau$*. (If both* $\tau=1,2$ *satisfy the condition then* $\tau$ *is selected arbitrarily. In case no* $\tau \in \{1,2\}$ *satisfies the condition,* $\tau$ *is set to 0). The prover sends* $\tau$ *to the verifier.*

- Verifier's second step (V2): *If the message,* $\tau$*, received from the prover equals* $\sigma$ *(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.[8] Recall that the latter is a co$\mathcal{NP}$-set (which is not known to be in $\mathcal{NP}$).

#### 9.1.3.2 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

We first show that co$\mathcal{NP} \subseteq \mathcal{IP}$, by presenting an interactive proof system for the co$\mathcal{NP}$-complete set of unsatisfiable CNF formulae. Next we extend this proof system to obtain one for the $\mathcal{PSPACE}$-complete set of unsatisfiable Quantified Boolean Formulae. Finally, we observe that $\mathcal{IP} \subseteq \mathcal{PSPACE}$. Indeed, proving that some co$\mathcal{NP}$-complete set has an interactive proof system is the core of the proof of Theorem 9.4 (see Exercise 9.2).

We show that the set of unsatisfiable CNF formulae has an interactive proof system by using algebraic methods, which are *applied to an arithmetic generalization of the said Boolean problem* (rather than to the problem itself). That is, in order to demonstrate that this Boolean problem has an interactive proof system, we first introduce an arithmetic generalization of CNF formulae, and then construct an interactive proof system for the resulting arithmetic assertion (by capitalizing on the arithmetic formulation of the assertion). Intuitively, we present an iterative process, which involves interaction between the prover and the verifier, such that in each iteration the residual claim to be established becomes simpler (i.e., contains one variable less). This iterative process seems to be enabled by the fact that the various claims refer to the arithmetic problem rather than to the original Boolean

[8] 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 $\sigma$. 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 $\sigma$ (chosen by the verifier), and so no prover may convince the verifier with probability exceeding 1/2.

problem. (Actually, one may say that the key point is that these claims refer to a generalized problem rather than to the original one.)

**Teaching note:** We devote most of the presentation to establishing that co$\mathcal{NP} \subseteq \mathcal{IP}$, and recommend doing the same in class. Our presentation focuses on the main ideas, and neglects some minor implementation details (which can be found in [162, 205]).

**The starting point:** We prove that co$\mathcal{NP} \subseteq \mathcal{IP}$ by presenting an interactive proof system for the set of unsatisfiable CNF formulae, which is co$\mathcal{NP}$-complete. Thus, our starting point is a given Boolean CNF formula, which is claimed to be unsatisfiable.

**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, the Boolean values `false` and `true` are replaced by the integer values 0 and 1 (respectively), OR-clauses are replaced by sums, and the top level conjunction is replaced by a product. This translation is depicted in Figure 9.1. Note that the Boolean formula

| | BOOLEAN | ARITHMETIC |
|---|---|---|
| variable values | `false`, `true` | 0, 1 |
| connectives | $\neg x$, $\vee$ and $\wedge$ | $1-x$, $+$ and $\cdot$ |
| final values | `false`, `true` | 0, positive |

Figure 9.1: Arithmetization of CNF formulae.

is satisfied (resp., unsatisfied) by a specific truth assignment if and only if evaluating the resulting arithmetic expression at the corresponding 0-1 assignment yields a positive (integer) value (resp., yields the value zero). Thus, the claim that the original Boolean formula is unsatisfiable translates to the claim that the summation of the resulting arithmetic expression, over all 0-1 assignments to its variables, yields the value zero. 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 unsatisfiable if and only if the sum of the corresponding 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 unsatisfiable 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, yields an integer value in $[0, 2^n v^m]$.

**Moving to a Finite Field:** In general, 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, if $q$ is prime then 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 $\phi$ is a low-degree multi-variate polynomial (and $q$ can be represented using $O(|\phi|)$ bits). In the rest of this exposition, all arithmetic operations refer to the finite field of $q$ elements, denoted $\mathrm{GF}(q)$.

**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.)[9] The verifier checks that the polynomial (say, $p$) is of low degree, and that it corresponds to the current value (say, $v$) being claimed (i.e., 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.

[9]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., $\log_2 q = O(vm)$).

**A single iteration (detailed):** The $i^{\text{th}}$ 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 $i^{\text{th}}$ iteration consists of two steps (messages): a prover step followed by a verifier step. The prover is supposed to provide the verifier with the univariate polynomial $p_i$ that satisfies

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

Note that, module $q$, the value $p_i(0)+p_i(1)$ equals the l.h.s of Eq. (9.2). 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 proved in the next iteration is

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

where $v_i \stackrel{\text{def}}{=} p'_i(r_i) \bmod q$ is computed by each party.

**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. Indeed, let us focus on the $i^{\text{th}}$ iteration, and let $v_{i-1}$ and $p_i$ be as in Eq. (9.2) and Eq. (9.3), respectively; that is, $v_{i-1}$ is the (wrong) value claimed at the beginning of the $i^{\text{th}}$ iteration and $p_i$ is the polynomial representing the expression obtained when stripping the current variable (as in Eq. (9.3)). Let $p'_i(\cdot)$ be any potential answer by the prover. We may assume, without loss of generality, that $p'_i(0) + p'_i(1) \equiv v_{i-1} \pmod{q}$ and that $p'_i$ is of low-degree (since otherwise the verifier will definitely reject). Using our hypothesis (that the entry claim of Eq. (9.2) is false), we know that $p_i(0) + p_i(1) \not\equiv v_{i-1} \pmod{q}$. Thus, $p'_i$ and $p_i$ are different low-degree polynomials, and so they may agree on very few points (if at all). Now, if the verifier's instantiation (i.e., its choice of a random $r_i$) does not happen to be one of these few points (i.e., $p_i(r_i) \not\equiv p'_i(r_i) \pmod{q}$), then the ending claim (i.e., Eq. (9.4)) is false too (because the new value (i.e., $v_i$) is set to $p'_i(r_i) \bmod q$, while the residual expression evaluates to $p_i(r_i)$). Details are left as an exercise (see Exercise 9.3).

This establishes that the set of unsatisfiable CNF formulae has an interactive proof system. Actually, a similar proof system (which uses a related arithmetization – see Exercise 9.5) 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, as outlined next.

**Obtaining interactive proofs for PSPACE (the 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).[10] 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 foregoing 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).[11] 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 9.7) that if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3, \mathrm{poly}(\log M)]$. Thus, we let the verifier selects a random prime $q$ of length that is linear in the length of the original formula, and the two parties consider the arithmetic expression reduced modulo this $q$. The second difficulty is resolved by

[10] 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, an interactive proof system for QBF can be obtained by extending the related proof system presented in Exercise 9.5.

[11] This high degree causes two difficulties, where only the second one is acute. The first difficulty is that the soundness of the corresponding protocol will require working in a finite field that is sufficiently larger than this high degree, but we can afford doing so (since the degree is at most exponential in the formula's length). The second (and more acute) difficulty is that the polynomial may have a large (i.e., exponential) number of non-zero coefficients and so the verifier cannot afford to read the standard representation of this polynomial (as a list of all non-zero coefficients). Indeed, other succinct and effective representations of such polynomials may exist in some cases (as in the following example), but it is unclear how to obtain such representations in general.

noting that $\mathcal{PSPACE}$ is actually reducible to a special form of (non-canonical) 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 9.6). 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, in polynomial space, all possible interactions of the prescribed verifier with any fixed polynomial-space prover strategy (e.g., an optimal one).

**Proposition 9.5** *Let $V$ be a probabilistic polynomial-time* (verifier) *strategy. 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 $\gamma$ of the interaction so far, the strategy[12] $f$ determines an optimal next-message for the prover by considering all possible coin tosses of the verifier that are consistent with $(x, \gamma)$. Specifically, $f$ is determined *recursively* such that $f(x, \gamma) = m$ if $m$ maximizes the number of outcomes of the verifier's coin-tosses that are consistent with $(x, \gamma)$ and lead the verifier to accept when subsequent prover moves are determined by $f$ (which is where recursion is used). That is, the verifier's random sequence $r$ 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 $(x, \gamma)$, 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 the subsequent prover moves are determined by $f$, which means at termination (i.e., after $T$ rounds) it holds 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 $\beta_i = V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_i)$.

[12]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 $V$).

Thus, $f(x,\gamma) = m$ if $m$ maximizes the value of $\mathsf{E}[\xi_{f,V}(x,R_\gamma,\gamma,m)]$, where $R_\gamma$ is selected uniformly among the $r$'s that are consistent with $(x,\gamma)$ and $\xi_{f,V}(x,r,\gamma,m)$ indicates whether or not $V$ accepts $x$ in the subsequent interaction with $f$ (which refers to randomness $r$ and partial transcript $(\gamma,m)$). It follows that the value $f(x,\gamma)$ can be computed in polynomial-space when given oracle access to $f(x,\gamma,\cdot,\cdot)$. The proposition follows by standard composition of space-bounded computations (i.e., allocating separate space to each level of the recursion, while using the same space in all recursive calls of each level). $\square$

### 9.1.4 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).

#### 9.1.4.1 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, by Theorem 9.4 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.* (A stronger statement appears at the end of §9.1.4.3.)

Indeed, public-coin proof systems are a syntactically restricted type of interactive proof systems. This restriction may make the design of such systems more difficult, but potentially facilitates their analysis (and especially when the analysis refers to 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.

#### 9.1.4.2 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 the soundness conditions. Note that sets having such generalized (two-sided error) interactive proofs are also in $\mathcal{PSPACE}$, and thus (by Theorem 9.4) allowing two-sided error does not increase the power of interactive proofs. See further discussion at the end of §9.1.4.3.

#### 9.1.4.3 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).[13]

**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.*[14]
- *For a set of integer functions, $M$, we let $\mathcal{IP}(M) \stackrel{\rm def}{=} \bigcup_{m\in M} \mathcal{IP}(m)$. 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 $\mathcal{IP}(0) = \text{co}\mathcal{RP}$.)

Definition 9.6 gives rise to a natural hierarchy of interactive proof systems, where different "levels" of this hierarchy correspond to different "growth rates" of the round-complexity of these systems. The following results are known regarding this hierarchy.

- A linear speed-up (see Appendix **??** (or [23] and [111])): 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 that are 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 [167]).
- If $\text{co}\mathcal{NP} \subseteq \mathcal{IP}(2)$ then the Polynomial-Time Hierarchy collapses (see [45]).

[13] An even finer structure emerges when considering also the total length of the messages sent by the prover (see [106]).

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

It is conjectured that 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.[15]

The class $\mathcal{IP}(1)$, also denoted $\mathcal{MA}$, seems to be *the* "real" randomized (and yet non-interactive) version of $\mathcal{NP}$: Here the prover supplies a candidate (polynomial-size) "proof", and the verifier assesses its validity probabilistically (rather than deterministically).

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

1. Emulating $\mathcal{IP}(\cdot)$ by $\mathcal{AM}(\cdot)$ (see §**??** or [111]): $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$.

2. Linear speed-up for $\mathcal{AM}(\cdot)$ (see §**??** or [23]): $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f+1)$.

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 shorthand $\mathcal{AM}(2)$ by $\mathcal{AM}$, which is indeed inconsistent with the convention of using $\mathcal{IP}$ as shorthand of $\mathcal{IP}(\mathtt{poly})$.

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 9.8).

#### 9.1.4.4 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 soundness of its verdict by assumptions concerning the potential prover(s). This should be contrasted with other notions of proof systems, such as computationally-sound ones (see §9.1.5.2), in which the soundness of the verifier's verdict depends on assumptions concerning the potential prover(s).

### 9.1.5 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:

[15] 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. 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).

#### 9.1.5.1 How powerful should the prover be?

Suppose that a set $S$ is in $\mathcal{IP}$. This means that there exists 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 (for convincing any fixed verifier $V$) can be implemented in polynomial-space, and that we cannot expect any better for a generic set in $\mathcal{PSPACE} = \mathcal{IP}$ (because the emulation of the interaction of $V$ with any optimal prover strategy yields a decision procedure for the set). Still, we may seek better upper-bounds on the complexity of some prover strategy 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 wish to upper-bound the computational power that suffices for convincing any of these verifiers (to accept any input in $S$).

We stress that, unlike the case of computationally-sound proof systems (see §9.1.5.2), 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 party that plays the role of the prover.

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 (and see also Exercise 9.10).

   This interpretation generalizes the notion of self-reducibility of NP-proof systems. Recall that by self-reducibility of an NP-set (or rather of the corresponding NP-proof system) we mean that the search problem of finding an NP-witness is polynomial-time reducible to deciding membership in the set (cf. Definition 2.14). Here we require that implementing the prover strategy (in the relevant interactive proof) be polynomial-time reducible to deciding membership in the set.

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 time-complexity of convincing a "lazy person" (i.e., a verifier) to the time-complexity of determining the truth (i.e., deciding membership in the set).

   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).

#### 9.1.5.2 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. The verifier's verdict in such a system is not absolutely sound, but is rather sound *provided that the potential cheating prover does not exceed the presumed complexity limits*. As in §9.1.5.1, the notion of "relative efficiency" can be given different interpretations, the most popular one being that the cheating prover strategy can be implemented by a (non-uniform) family of polynomial-size circuits. The latter interpretation coincides with the first interpretation used in §9.1.5.1 (i.e., a probabilistic polynomial-time strategy that is given an auxiliary input (of polynomial length)). Specifically, in this case, 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

Standard mathematical proofs are believed to yield (extra) knowledge and not merely establish the validity of the assertion being proved; that is, 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 hard to compute (even when assuming that the input is in $S$). For example, a 3-coloring of a graph constitutes an NP-proof that the graph is 3-colorable, but it yields information (i.e., the coloring) that seems infeasible to compute (when given an arbitrary 3-colorable graph).

A natural question that arises is whether or not proving an assertion always requires giving away some extra knowledge. The setting of interactive proof systems enables a negative answer to this fundamental question: In contrast to NP-proofs, which seem to yield a lot of knowledge, zero-knowledge (interactive) proofs yield no knowledge at all; that is, *zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proved.* For example, a zero-knowledge proof of 3-colorability does not yield any information about the graph (e.g., partial information about a 3-coloring) that is infeasible to compute from the graph itself. Thus, zero-knowledge proofs exhibit an extreme contrast between being convincing (of the validity of a assertion) and teaching anything on top of the validity of the assertion.

Needless to say, the notion of zero-knowledge proofs is fascinating (e.g., since it differentiates proof-verification from learning). Still, the reader may wonder whether such a phenomenon is desirable, because in many settings we do care to learn as much as possible (rather than learn as little as possible). However,

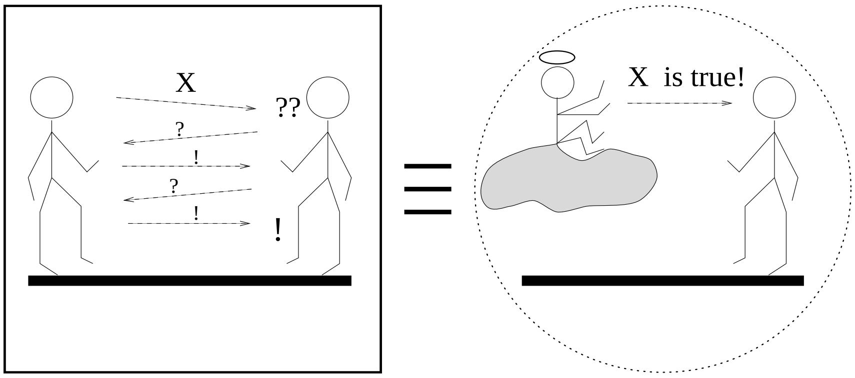


Figure 9.2: Zero-knowledge proofs – an illustration.

in other settings (most notably in cryptography), we may actually wish to limit the gain that other parties may obtained from a proof (and, in particular, limit this gain to the minimal level of being convinced in the validity of the assertion). Indeed, the applicability of zero-knowledge proofs in the domain of cryptography is vast; they are typically used as a tool for forcing (potentially malicious) parties to behave according to a predetermined protocol (without having them reveal their own private inputs). The interested reader is referred to discussions in §C.4.3.2 and §C.7.3.2 (and to detailed treatments in [91, 92]). We also mention that, in addition to their direct applicability in Cryptography, zero-knowledge proofs serve as a good bench-mark for the study of various questions regarding cryptographic protocols.

**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 [91, 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.[16] We wish to state that

[16]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

no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). The question addressed here is how to formulate the "no gain" requirement.

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. The answer 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* (or prescribed) *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 party" (which believes the assertion) can obtain from the assertion itself.

The foregoing interpretation of "gaining nothing" means that any feasible adversarial behavior can be "simulated" by a benign behavior (and thus there is no gain in the former). This line of reasoning is called the simulation paradigm, and is pivotal to many definitions in cryptography (e.g., it underlies the definitions of security of encryption schemes and cryptographic protocols); for further details see Appendix C.

#### 9.2.1.2 The basic definitions

We turn back to the concrete task of defining zero-knowledge. Firstly, we comment that zero-knowledge is a property of some prover strategies; actually, more generally, zero-knowledge is a property of some strategies. Fixing any strategy (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 strategy* 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 strategy 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 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 (and does not interact with anyone)). The basic paradigm of zero-knowledge asserts that for

assertions).

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):[17] *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, $A^*$, such that*

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

*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 $A^*(x)$ is a random variable representing the output of algorithm $A^*$ 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, $A^*$, *such that for every probabilistic polynomial-time* distinguisher, $D$, *it holds that*

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

*is a negligible function.*[18] *We denote by $\mathcal{ZK}$ the class of sets having zero-knowledge interactive proof systems.*

[17] In the actual definition one relaxes the requirement in one of the following two ways. The first alternative is allowing $A^*$ to run for *expected* (rather than strict) polynomial-time. The second alternative consists of allowing $A^*$ 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.

[18] 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, $d(n) \stackrel{\rm def}{=} 0$ if $S \cap \{0,1\}^n = \emptyset$.

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 $A^*$ with the same value of an arbitrary ($\text{poly}(|x|)$-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 9.13). 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 9.14). Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For further details, see [91, 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 [90, 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).[19] 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 [91, Sec. 4.5.2] or [228].) But first, we demonstrate the non-triviality 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 (yet, the result is significant only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$).

### 9.2.2.1 A simple example

> *A story not found in the Odyssey refers to the not so famous Labyrinth of the Island of Aeaea. The Sorceress Circe, daughter of Helius, challenged godlike Odysseus to traverse the Labyrinth from its North Gate to its South Gate. Canny Odysseus doubted whether such a path existed at all and asked beautiful Circe for a proof, to which she replied*

[19] Recall 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. Thus, the issue is the existence of zero-knowledge proofs for sets outside $\mathcal{BPP}$.

> *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. Still, as we shall shortly see, this set 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* $G_2 = (V_2, E_2)$.

  *If the input graphs are indeed isomorphic, then we let* $\phi$ *denote an arbitrary isomorphism between them; that is,* $\phi$ *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* $\pi$ *from the set of permutations over the vertex set* $V_2$*, and constructs a graph with vertex set* $V_2$ *and edge set*

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

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

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

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

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

  *(Indeed, the prover treats any $\sigma \neq 2$ as $\sigma = 1$. Thus, 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 $\psi$, received from the prover is an isomorphism between $G_\sigma$ 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. If the prover is given an isomorphism between the input graphs as auxiliary input, then 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. Thus, we focus on establishing the zero-knowledge property.

We consider first the special case in which the verifier actually follows the prescribed strategy (and selects $\sigma$ at random, and in particular obliviously of the graph $G'$ it receives). The view of this verifier can be easily simulated by selecting $\sigma$ and $\psi$ at random, constructing $G'$ as a random isomorphic copy of $G_\sigma$ (via the isomorphism $\psi$), and outputting the triple $(G', \sigma, \psi)$. 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 $\pi$), but its output distribution is identical to the verifier's view in the real interaction. However, the foregoing description assumes that the verifier follows the prescribed strategy, while in general the verifier may (adversarially) select $\sigma$ 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 $\sigma$ and completes 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 $G_\tau$ (and thus being able to present an isomorphism between $G_\tau$ 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., $\tau$).[20] Thus,

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

the value $\sigma$ selected by the adversarial verifier may depend on $G'$ but not on $\tau$, which implies that $\Pr[\sigma = \tau] = 1/2$. In other words, the simulator's guess (i.e., $\tau$) is correct (i.e., equals $\sigma$) 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.

**Digest: a few useful conventions.** We 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).[21]

   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. Indeed, this justification relies on the fact that the simulation refers to verifiers with arbitrary auxiliary inputs (of polynomial length).

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[22] 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.

[21]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).

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

#### 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 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. Still, as we shall shortly see, graph 3-colorability does have a zero-knowledge interactive proof system. This 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* $\psi$ *be a 3-coloring of* $G$. *The prover selects a random permutation,* $\pi$, *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* $\psi$. *The prover sends to the verifier a sequence of* $|V|$ *locked and non-transparent boxes such that the* $v^{\text{th}}$ *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).

So far we showed that Construction 9.10 constitutes (a weak form of) an interactive proof system for Graph 3-Colorability. The point, however, is that the prescribed prover strategy is zero-knowledge. This is easy to see in the abstract setting of Construction 9.10, because all that the verifier sees in the real interaction is a sequence of boxes and a random pair of *different* colors (which is easy to simulate). Indeed, the simulation of the real interaction proceeds by presenting a sequence of boxes and providing a random pair of different colors as the contents of the two boxes indicated by the verifier. Note that the foregoing argument relies on the fact that the boxes (indicated by the verifier) correspond to vertices that are connected by an edge in the graph.

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 affected, yet in an indistinguishable manner). Thus, the verifier's selection of the inspected edge may depend on the "outside appearance" of the various boxes, which in turn may depend (in an indistinguishable manner) on the contents of these boxes. Consequently, we cannot determine the boxes' contents after a pair of boxes are selected, and so the simple foregoing simulation is inapplicable. Instead, we simulate the interaction as follows.

1. 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).[23] Then, we hand all boxes to the verifier, which asks us to open a pair of boxes (corresponding to an edge).

2. 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 (i.e., repeat Step 1 with a new random guess and random colors). The key observation is that if the boxes hide the contents in the sense that a box's contents is indistinguishable based on it outside appearance, then our guess will succeed with probability approximately $1/|E|$. Furthermore, in this case, the simulated execution will be indistinguishable from the real interaction.

[23] 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.

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 [100] (or [91, Sec. 4.4])).

**Digital implementation (overview).** 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); see §C.4.3.1.

**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 systems for any NP-set.[24] 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, it holds that* $\mathcal{NP} \subseteq \mathcal{ZK}$. *Furthermore, every* $S \in \mathcal{NP}$ *has a* (computational) *zero-knowledge interactive proof system in which the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided that it is given as auxiliary-input an NP-witness for membership of the common input in* $S$.

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, does not necessarily preserve the complexity of the prover.)

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

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

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 ("negative") hardness result (cf., Section 2.2.4).[25]

**Perfect and Statistical Zero-Knowledge.** The foregoing results, which refer to computational zero-knowledge proof systems, should be contrasted with the known 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 hand, the class Statistical Zero-Knowledge is known to contain some seemingly 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 [227].

### 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). Note that the entity asserting knowledge is actually the prover's strategy, which is an automated computing device, hereafter referred to as a machine. This raises the question of what do we mean by saying that a *machine knows something*.

[25] Historically, the proof of Theorem 9.11 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 study of statistical zero-knowledge (cf. [227]).

#### 9.2.3.1 Abstract reflections

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 may say that a machine knows a string $\alpha$ if it *can* output the string $\alpha$. But this seems as total non-sense too: a machine has a well defined output – either the output equals $\alpha$ or it does not, so what can be meant by saying that *a machine can do something*?

Interestingly, a sound interpretation of the latter phrase does exist. Loosely speaking, by saying that *a machine can do something* we mean that the machine can be *easily modified* such that it (or rather its modified version) does whatever is claimed. More precisely, this means 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).

#### 9.2.3.2 A concrete treatment

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 not 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.*[26] 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 a strategy, $V$, 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 $P$ as a "black box"[27] 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 the strategy $P$ (i.e., the function specifying the message that $P$ sends in response to any sequence of messages it may receive). We require that the (*expected*) *running time of the extractor, on input $G$ and oracle access to $P$, 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 [91, 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 (see, e.g., [91, 92]). 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.*

## 9.3 Probabilistically Checkable Proof Systems

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

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

[26] For simplicity, the reader may consider graphs that have a unique 3-coloring (up-to 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.

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

proof. Actually, we focus on the latter probabilistic procedure, which in turn implies the existence of a deterministic verification procedure (obtained by going over all possible random choices of the probabilistic procedure and making the adequate examinations).

Modeling such probabilistic verification procedures, which may examine few locations in the alleged proof, requires providing these procedures with direct access to the individual bits of the alleged proof (so that they need not scan the proof bit-by-bit). Thus, the alleged proof is a string, as in the case of a traditional proof system, but the (probabilistic) verification procedure is given direct access to individual bits of this string (see Figure 9.3).

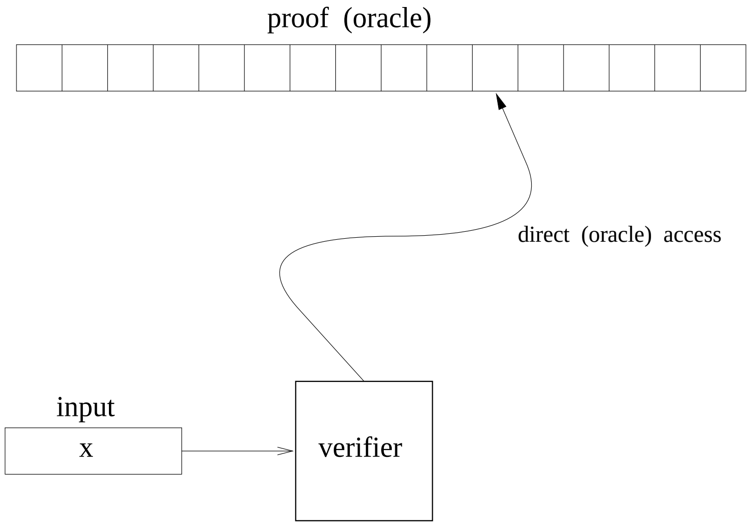


Figure 9.3: The PCP model – an illustration.

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. Such probabilistic verification procedures are called probabilistically checkable proof (PCP) systems.

The fact that one can (meaningfully) evaluate the correctness of proofs by examining few locations in them is indeed amazing and somewhat counter-intuitive. Needless to say, such proofs must be written in a somewhat non-standard format, because standard proofs cannot be verified without reading them in full (since a flaw may be due to a single improper inference). In contrast, proofs for a PCP system tend to be very redundant; they consist of superfluously many pieces of information (about the claimed assertion), but their correctness can be (meaningfully) evaluated by *checking the consistency of a randomly chosen collection of few related pieces.* We stress that by a "meaningful evaluation" we mean rejecting alleged proofs of false assertions with constant probability (rather than with probability that is inversely proportional to the length of the alleged proof).

The main complexity measure associated with PCPs 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* $\pi_x$ *such that, on input* $x$ *and access to oracle* $\pi_x$, *machine* $V$ *always accepts* $x$.
   - Soundness: *For every* $x \notin S$ *and every oracle* $\pi$, *on input* $x$ *and access to oracle* $\pi$, *machine* $V$ *rejects* $x$ *with probability at least* $\frac{1}{2}$.

2. *We say that a probabilistically checkable proof system has* query complexity $q:\mathbb{N}\rightarrow\mathbb{N}$ *if, on any input of length* $n$, *the verifier makes at most* $q(n)$ *oracle queries.*[28] *Similarly, the* randomness complexity $r:\mathbb{N}\rightarrow\mathbb{N}$ *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*

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

*complexity $q$. For sets of integer functions, $R$ and $Q$,*

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

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 of non-trivial query-complexity (cf. Section 9.3.2) provide a trade-off between the number of locations examined in the proof and the confidence in the validity of the assertion.

We note that the oracle $\pi_x$ referred to in the completeness condition of a PCP system constitutes a proof in the standard mathematical sense. Indeed any PCP system yields a standard proof system (with respect to a verification procedure that scans all possible outcomes of $V$'s internal coin tosses and emulates all the corresponding checks). Furthermore, the oracles in PCP systems of logarithmic randomness-complexity constitute NP-proofs (see Exercise 9.15). However, the oracles of a PCP system 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 examine very few bits of an NP-proof and even utilize very long proofs (i.e., of super-polynomial length).

**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 on the verifier's internal coin tosses). In contrast, non-adaptive verifiers determine all their queries based solely on their input and internal coin tosses. Note that $q$ adaptive (binary) queries can be emulated by $\sum_{i=1}^{q} 2^{i-1} < 2^q$ non-adaptive (binary) queries. We comment that most constructions of PCP systems use non-adaptive verifiers, and in fact in many sources PCP systems are defined as non-adaptive.

**Randomness versus proof length.** Fixing a verifier $V$, we say that location $i$ (in the oracle) is relevant to input $x$ if there exists a computation of $V$ on input $x$ in which location $i$ is queried (i.e., there exists $\omega$ and $\pi$ such that, on input $x$, randomness $\omega$ and access to the oracle $\pi$, the verifier queries location $i$). The effective proof length of $V$ is the smallest function $\ell:\mathbb{N}\rightarrow\mathbb{N}$ such that for every input $x$ there are at most $\ell(|x|)$ locations (in the oracle) that are relevant to $x$. We claim that the effective proof length of any PCP system is closely related to its randomness (and query) complexity. On one hand, *if the PCP system has randomness-complexity $r$ and query-complexity $q$, then its effective proof length is upper-bounded by $2^{r+q}$*, whereas a bound of $2^r \cdot q$ holds for non-adaptive systems (see Exercise 9.15). Thus, *PCP systems of logarithmic randomness complexity have* *effective proof length that is polynomial*, and hence yield NP-proof systems. On the other hand, in some sense, the randomness complexity of a PCP system can be upper-bounded by the logarithm of the (effective) length of the proofs employed (provided we allow non-uniform verifiers; see Exercise 9.16).

**On the role of randomness.** The PCP Theorem (i.e., $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, O(1))$) asserts that a meaningful probabilistic evaluation of proofs is possible based on a constant number of examined bits. We note that, unless $\mathcal{P} = \mathcal{NP}$, such a phenomena is impossible when requiring the verifier to be deterministic. Firstly, note that $\mathcal{PCP}(0, O(1)) = \mathcal{P}$ holds (as a special case of $\mathcal{PCP}(r, q) \subseteq \mathrm{D}\textsc{time}(2^{2^r q + r} \cdot \mathtt{poly})$; see Exercise 9.17). Secondly, as shown in Exercise 9.19, $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{NP}$ is not contained in $\mathcal{PCP}(o(\log), o(\log))$. Lastly, assuming that not all NP-sets have NP-proof systems that employs proofs of length $\ell$ (e.g., $\ell(n) = n$), it follows that if $2^{r(n)} q(n) < \ell(n)$ then $\mathcal{PCP}(r, q)$ does not contain $\mathcal{NP}$ (see Exercise 9.17 again).

### 9.3.2 The Power of Probabilistically Checkable Proofs

The celebrated PCP Theorem asserts that $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$, 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 $\mathcal{PCP}(\mathtt{poly}, 0)$ equals co$\mathcal{RP}$, whereas $\mathcal{PCP}(0, \mathtt{poly})$ 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 9.17):

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

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., $\mathcal{PCP}(\mathtt{log}, q)$) 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

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

is interesting (because it applies also to NP-sets having witnesses of length exceeding $q$). Needless to say, 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): $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, O(1))$.

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 $\mathcal{NP}$. This constant is essentially three (see §9.3.4.1). Before reviewing the proof of Theorem 9.16, we make a couple of comments.

**Efficient transformation of NP-witnesses to PCP oracles:** 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). That is, *for every* (NP-witness relation) *$R \in \mathcal{PC}$ there exists a PCP verifier $V$ as in Theorem 9.16 and a polynomial-time computable function $\pi$ such that for every $(x,y) \in R$ the verifier $V$ always accepts the input $x$ when given oracle access to the proof $\pi(x,y)$* (i.e., $\mathsf{Pr}[V^{\pi(x,y)}(x)=1] = 1$). Recalling that the latter oracles are themselves NP-proofs, it follows that 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 $\varepsilon$ 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.

The foregoing strengthening of Theorem 9.16 offers a wider range of applications than Theorem 9.16 itself. Indeed, Theorem 9.16 itself suffices for "negative" applications such as establishing the infeasibility of certain approximation problems (see Section 9.3.3). But for "positive" applications (see §9.3.4.2), typically some user (or a real entity) will be required to actually construct the PCP-oracle, and in such cases the strengthening of Theorem 9.16 will be useful.

**A 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): $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$.

**Road-map for 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), which 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, $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{poly}, O(1))$.

**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 strongly 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).[29] For an input consisting of a system of quadratic equations with $n$ variables, the oracle (of this PCP) is supposed to provide the evaluation of all quadratic expressions (in these $n$ variables) 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 all $2^n$ linear expressions and the second table to all $2^{n^2}$ 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"). Finally, we test that the assignment encoded in these tables satisfies the quadratic system that is given as input. This is done by taking a random linear combination of the quadratic equations that appear in the quadratic system, and obtaining the value assigned to the corresponding quadratic expression by the aforementioned tables (again, via self-correction). The key point is that each of the foregoing tests utilizes a constant number of Boolean queries, and has time (and randomness) complexity that is polynomial in the size of the input. Details follow.

**Teaching note:** The following text refers to notions such as the Hadamard encoding, testing and self-correction, which appear in other parts of this work (see, e.g., §E.1.2.2, Section 10.1.2. and §7.2.1.1, respectively). While a wider perspective (provided in the aforementioned parts) is always useful, the current text is self-contained.

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

**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 $\alpha$ and $\tau$ (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.

**Testing the Hadamard Code.** Note that $T_1$ is supposed to encode a linear function; that is, there must exist 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 9.20), where $T$ is $\varepsilon$-far from being linear if $T$ disagrees with any linear function $f$ on more than an $\varepsilon$ fraction of the domain (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] > \varepsilon$).

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 $\tau$ (resp., $\tau'$).[30] (Needless to say, this does *not* mean that $\tau'$ equals the outer product of $\tau$ with itself (i.e., $\tau'_{i,j}$ does not necessarily equal $\tau_i \tau_j$).)

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 $\tau$ (resp., $\tau'$) 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 $f_\tau$ (resp., $f_{\tau'}$).

**Self-correction of the Hadamard Code.** Suppose that $T$ is $\varepsilon$-close to a linear function $f : \mathrm{GF}(2)^m \to \mathrm{GF}(2)$ (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] \leq \varepsilon$). Then, we can recover

[30] Note that $\tau$ (resp., $\tau'$) is uniquely determined by $T_1$ (resp., $T_2$), because every two different linear functions $\mathrm{GF}(2)^m \to \mathrm{GF}(2)$ agree on exactly half of the domain (i.e., the Hadamard code has relative distance 1/2).

the value of $f$ at any desired point $x$, by making two (random) queries to $T$. Specifically, for a uniformly selected $r \in \mathrm{GF}(2)^m$, 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 $\varepsilon$-close to two different linear functions.)[31] Thus, assuming that $T_1$ is 0.02-close to $f_\tau$ (resp., $T_2$ is 0.02-close to $f_{\tau'}$) we may correctly recover (i.e., with error probability 0.04) the value of $f_\tau$ (resp., $f_{\tau'}$) at any desired point by making 2 queries to $T_1$ (resp., $T_2$). This process is called self-correction (cf., e.g., §7.2.1.1).

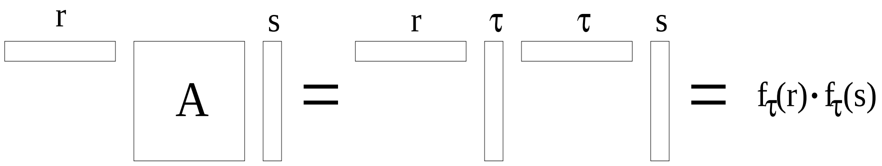


Figure 9.4: Detail for testing consistency of linear and quadratic forms.

**Checking consistency of $f_\tau$ 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 9.22) 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 9.4) 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 $\mathsf{Pr}_{r,s}[f_\tau(r)f_\tau(s) \neq f_{\tau'}(rs^\top)] \geq 1/4$.

Recall, however, that we do not have direct access to the functions $f_\tau$ and $f_{\tau'}$, but rather to tables (i.e., $T_1$ and $T_2$) that are 0.02-close to these functions. Still, using self-correction, we can obtain the values of $f_\tau$ and $f_{\tau'}$ at any desired point, with very high probability. Actually, when implementing the foregoing consistency test it suffices to use self-correction for $f_{\tau'}$, because we use the values of $f_\tau$ at two independently and uniformly distributed points in $\mathrm{GF}(2)^n$ (i.e., $r, s$) but the value $f_{\tau'}$ is required at $rs^\top$, which is not uniformly distributed in $\mathrm{GF}(2)^{n^2}$. Thus, we test the consistency of $f_\tau$ and $f_{\tau'}$ by selecting uniformly $r,s \in \mathrm{GF}(2)^n$ and $R \in \mathrm{GF}(2)^{n^2}$, and checking that $T_1(r)T_1(s) = T_2(rs^\top + R) - T_2(R)$.

By repeating the aforementioned (self-corrected) 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 $(\tau_i\tau_j)_{i,j} = (\tau'_{i,j})_{i,j}$.

[31] 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 that $\tau$ satisfies the quadratic system.** Suppose that we are given access to $f_\tau$ and $f_{\tau'}$ as in the foregoing (where, in particular, $\tau' = \tau\tau^\top$). A key observation is that if $\tau$ 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 $\tau$ satisfies the quadratic system (which is given as input), we create a single quadratic equation by taking such a random linear combination, and check whether this quadratic equation is satisfied by $\tau$. The punch-line is that *testing whether $\tau$ satisfies the quadratic equation $Q(x) = \sigma$ amounts to testing whether $f_{\tau'}(Q) = \sigma$*. Again, the actual checking is implemented by using self-correction (of the table $T_2$).

This completes the description of the verifier. Note that this verifier performs a constant number of codeword tests for the Hadamard Code, and a constant number of consistency and satisfiability tests, where each of the latter involves self-correction of the Hadamard Code. Each of the individual tests utilizes a constant number of queries (ranging between two and four) and uses randomness that is quadratic in the number of variables (and linear in the number of equations in the input). Thus, the query-complexity is a constant and the randomness-complexity is at most quadratic in the length of the input (quadratic system). Clearly, if the input quadratic system is satisfiable (by some $\tau$), 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 input 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 $\mathcal{NP} \subseteq \mathcal{PCP}(q, O(1))$, where $q$ is a quadratic polynomial.

**Reflection.** Indeed, the actual test of the satisfiability of the quadratic system that is given as input is facilitated by the fact that a satisfying assignment is encoded (in the oracle) in a very redundant manner, which fits the final test of satisfiability. But then the burden of testing moves to checking that this encoding is indeed valid. In fact, most of the tests performed by the foregoing verifier are aimed at verifying the validity of the encoding. Such a test of validity (of encoding) may be viewed as a test of consistency between the various parts of the encoding. All these themes are present also in more advanced constructions of PCP systems.

#### 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 complexity. 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:[32] In general, this paradigm allows to compose two proof systems such that the "inner" verifier is used for probabilistically verifying the acceptance criteria of the "outer" verifier. That is, the combined verifier selects coins for the "outer" verifier, determines the corresponding locations that the "outer" verifier wishes to inspect (in the proof), and verifies that the "outer" verifier would have accepted the values that reside in these locations. The latter verification is performed by invoking the "inner" verifier, *without reading the values residing in all the aforementioned locations*. Indeed, the aim is conducting this ("composed") verification while 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 the 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). The resulting verifier selects coins for the outer-verifier and uses the corresponding "inner" proof in order to verify that the outer-verifier would have accepted under this choice of coins. Note that such a choice of coins determines locations in the "outer" proof that the outer-verifier would have inspected, and the combined verifier provides the inner-verifier with oracle access to these locations (which the inner-verifier considers as its input) as well as with oracle access to the corresponding "inner" proof (which the inner-verifier considers as its proof-oracle). See Figure 9.5 (and further details that follow the current sketch).

   Note that 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

[32] Our presentation of the composition paradigm follows [35], rather than the original presentation of [16, 15].

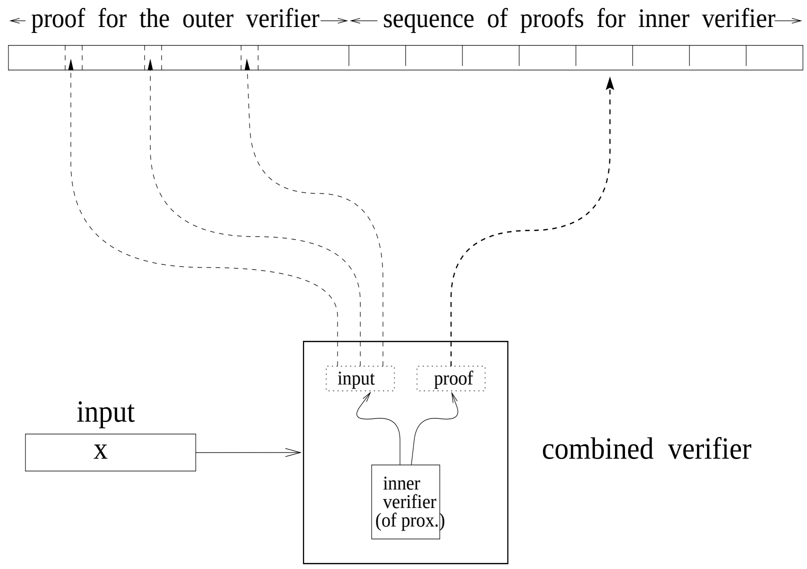


Figure 9.5: Composition of PCP system. The dashed arrows indicate pointers from the (virtual) input and proof oracles of the inner-verifier to the actual proof of the composed verifier. These pointers (as well as the residual predicate) are determined by an invocation of the outer-verifier.

> 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 $q'(n) = q''(n) = \mathrm{poly}(\log n)$] yielding a PCP system (for $\mathcal{NP}$) of randomness-complexity $r(n) = r'(n) + r''(q'(n)) = O(\log n)$ and query-complexity $q(n) = q''(q'(n)) = \mathrm{poly}(\log\log n)$. Composing the latter system (used as an "outer" system) with the PCP system of Step 2, yields a PCP system (for $\mathcal{NP}$) of randomness-complexity $r(n)+\mathrm{poly}(q(n)) = O(\log n)$ 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) was already presented 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. Next, we detail the composition of an "outer" robust-PCP with an "inner" PCP-of-proximity. 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 (direct) *access to two oracles, one representing an input and the other being an alleged proof, and its queries to both oracles are counted in its query-complexity.* 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 small constant)).

For example, consider a variant of the system of §9.3.2.1 in which the quadratic system is fixed[33] 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. We use a proof oracle as in §9.3.2.1, and a PCP verifier of proximity that proceeds 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 reads a uniformly selected bit of the input oracle and compares 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 $i^{\text{th}}$ 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. Hence, 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). That is, for every no-instance $x$ and every alleged proof $\pi = \pi_1\pi_2\cdots\pi_\ell \in \{0,1\}^\ell$, it is required that, with probability at least 1/2 over the verifier's choice of coins $\omega \in \{0,1\}^r$, it holds that $\pi_{i_{\omega,1}}\pi_{i_{\omega,2}}\cdots\pi_{i_{\omega,q}}$ is far from any assignment that satisfies $P_\omega$, where $i_{\omega,j}$ is the $j^{\text{th}}$ query made (non-adaptively) on coins $\omega$, and $P_\omega$ is the residual predicate that determines which sequences of answers are accepted in this case. Indeed, if the outer verifier is robust, then it suffices to distinguish 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 constant 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 postulated in Step 1 and outlined shortly).

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

**A closer look at proof composition.** Following the foregoing sketch, we further detail the proof composition operation that is employed in the current subsection (i.e., §9.3.2.2). We start by detailing the two PCPs being composed. Let $V_1$ be a *robust* verifier of randomness-complexity $r_1$ and query-complexity $q_1$, and suppose that its residual decision on input $x$ and random-tape $\omega \in \{0,1\}^{r_1(|x|)}$ can be described by a poly$(q_1(|x|))$-size circuit, denoted $C_\omega$. That is, on input $x$, access to an oracle $\pi = \pi_1\pi_2 \cdots \pi_\ell$, and random-tape $\omega \in \{0,1\}^{r_1(|x|)}$, the verifier $V_1$ accepts if and only if $C_\omega(\pi_{i_{\omega,1}}\pi_{i_{\omega,2}} \cdots \pi_{i_{\omega,q_1(|x|)}}) = 1$, where $i_{\omega,j}$ is the $j^{\rm th}$ query made (non-adaptively) on input $x$ and random-tape $\omega$. Note that membership in $C_\omega^{-1}(1)$ can be determined in time poly$(|C_\omega^{-1}|)$ = poly$(q_1(|x|))$. Let $V_2$ be a verifier *of proximity* for membership in $C_\omega^{-1}(1)$, and suppose that its proximity parameter equals (or is smaller than) the robustness parameter of $V_1$. Actually, the verifier $V_2$ should either depend on the circuit $C_\omega$ or get the description of $C_\omega$ as auxiliary input.[34] Turning to the combined verifier resulting from the composition, we first postulate that, on input $x$, this verifier utilizes proofs of the form $(\pi, (\pi^{(\omega)})_{\omega\in\{0,1\}^{r_1(|x|)}})$, where $\pi$ is a proof for $V_1$ (regarding the input $x$) and $\pi^{(\omega)}$ is a proof for $V_2$ (regarding membership of the string $\pi_{i_{\omega,1}}\pi_{i_{\omega,2}} \cdots \pi_{i_{\omega,q_1(|x|)}}$ in the set $C_\omega^{-1}(1)$). The combined verifier uniformly selects a random-tape $\omega \in \{0,1\}^{r_1(|x|)}$ (for $V_1$), determines the locations $i_{\omega,1}, i_{\omega,2}, ..., i_{\omega,q_1(|x|)}$ (which $V_1$ would query on input $x$ and random-tape $\omega$), and invokes $V_2$ while providing it with access to the input-oracle $\pi_{i_{\omega,1}}\pi_{i_{\omega,2}} \cdots \pi_{i_{\omega,q_1(|x|)}}$ and the proof-oracle $\pi^{(\omega)}$. That is, if $V_2$ queries the $j^{\rm th}$ bit of its input (resp., its proof) then the combined verifier queries the $i_{\omega,j}^{\rm th}$ bit of $\pi$ (resp., the $j^{\rm th}$ bit of $\pi^{(\omega)}$) and provides $V_2$ with the bit retrieved.

Clearly, if $x$ is a yes-instance then using the adequate proofs $\pi$ and $(\pi^{(\omega)})_{\omega\in\{0,1\}^{r_1(|x|)}}$ makes the combined verifier accept with probability 1. On the other hand, if $x$ is a no-instance then $V_1$ will "robustly rejects" any $\pi$ with probability at least 1/2 (i.e., with probability at least 1/2 over the choice of $\omega \in \{0,1\}^{r_1(|x|)}$, it holds that $\pi_{i_{\omega,1}}\pi_{i_{\omega,2}} \cdots \pi_{i_{\omega,q_1(|x|)}}$ is far from any string in the set $C_\omega^{-1}(1)$). Now, if $V_1$ "robustly rejects" $\pi$ when using the random-tape $\omega \in \{0,1\}^{r_1(|x|)}$, then (for any $\pi^{(\omega)}$) the corresponding executions of $V_2$ will reject with probability at least 1/2. It follows that, for any choice of its proof oracle (i.e., any $\pi$ and $(\pi^{(\omega)})_{\omega\in\{0,1\}^{r_1(|x|)}}$), the combined verifier rejects each no-instance with probability at least 1/4. Needless to say, the rejection probability can be increased by sequential repetitions.

[34] In the former case, $V_2$ is a circuit (with oracle access to its input and proof oracles), which incorporates the circuit $C_\omega$. In the latter case, the formulation of PCP of proximity should be extended so to account for inputs that are given in two parts such that the first part (e.g., $C_\omega$) is given explicitly (as an ordinary input) and the second part (e.g., the input to $C_\omega$) is given implicitly via oracle access. Either way, it is essential that the size of $C_\omega$ is polynomial in the length of its own input (i.e., $|C_\omega| = \text{poly}(q_1(|x|))$). In fact, an asymptotic treatment is facilitated by using the latter formulation (of two-part inputs). In this case, $V_2$ is actually an (extended) PCP of proximity for statements in $\mathcal{P} \subseteq \mathcal{NP}$, where the valid statements have the form $(C, \alpha)$ such that $C(\alpha) = 1$ (where $C$ is presented as explicit input and $\alpha$ is presented as implicit input).

**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 focus on showing that $NP \subseteq \mathcal{PCP}(f,f)$, for $f(n) = \text{poly}(\log n)$, and the claimed result will follow by a relatively minor modification (discussed afterwards). The proof system underlying $NP \subseteq \mathcal{PCP}(f,f)$ is based on an arithmetization of 3CNF formulae, which is different from the one used in §9.1.3.2 (for constructing an interactive proof system for co$\mathcal{NP}$). We start by describing this arithmetization, and later outline the PCP system that is based on it.

In the current arithmetization, the names of the variables (resp., clauses) of a 3CNF formula $\phi$ are represented by binary strings of logarithmic (in $|\phi|$) length, and a *generic* variable (resp., clause) of $\phi$ is represented by a logarithmic number of *new variables*, which are assigned values in a finite field $\mathrm{F} \supset \{0,1\}$. Indeed, throughout the rest of the description, we refer to the arithmetic operations of this finite field F (which will have cardinality $\text{poly}(|\phi|)$). The (structure of the) 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 $i^{\text{th}}$ literal in the $\alpha^{\text{th}}$ clause of $\phi$ has index $\beta_i = (\gamma_i,\sigma_i)$, which 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 $C_\phi$ over F, denoted $\Phi$; that is, $\Phi$ is the (unique) multi-linear polynomial that agrees with $C_\phi$ on $\{0,1\}^{O(\log n)} \subset \mathrm{F}^{O(\log n)}$.

Turning to the PCP, we first note that the verifier can reduce the original 3SAT-instance $\phi$ to the aforementioned arithmetic instance $\Phi$; that is, on input a 3CNF formula $\phi$, the verifier first constructs $C_\phi$ and $\Phi$ (as in Exercise 7.12). Part of the proof oracle for 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 $\phi$ (i.e., for every $\gamma \in \{0,1\}^{\log n} \equiv [n]$, the value $A(\gamma)$ is supposed to be the value of the $\gamma^{\text{th}}$ variable in such an assignment). Thus, we wish to check whether, for every $\alpha \in \{0,1\}^{\log|\phi|}$, it holds that

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

where $A'(\beta)$ is the value of the $\beta^{\text{th}}$ literal under the (variable) assignment $A$; that is, for $\beta = (\gamma,\sigma)$, where $\gamma \in \{0,1\}^{\log n}$ is a variable name and $\sigma \in \{0,1\}$ indicates the literal's type (i.e., whether the variable is negated), it holds that $A'(\beta) = (1-\sigma)\cdot A(\gamma) + \sigma \cdot (1 - A(\gamma))$. Thus, Eq. (9.7) holds if and only if the $\alpha^{\text{th}}$ clause is satisfied by the assignment induced by $A$ (because $A'(\beta) = 1$ must hold for at least one of the three literals $\beta$ that appear in this clause).[35]

[35]Note that, for this $\alpha$ there exists a unique triple $(\beta_1,\beta_2,\beta_3) \in \{0,1\}^{3\log 2n}$ such that

As in §9.3.2.1, we cannot afford to verify all $|\phi|$ instances of Eq. (9.7). Furthermore, unlike in §9.3.2.1, we cannot afford to take a random linear combination of these $|\phi|$ 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$\mathcal{NP}$); that is, we refer to stripping the $\ell$ binary summations in iterations, where in each iteration the verifier obtains a corresponding univariate polynomial and instantiates it at a random point. 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).[36] Note that after stripping the $\ell$ summations, the verifier end-ups 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 $\ell$ 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 $|F| \gg O(\ell)$, where $\ell = O(\log|\phi|)$.

Recall, however, that we may not assume that $A$ is a multi-variate polynomial of low degree. Instead, we must check that $A$ is indeed a multi-variate polynomial of low degree (or rather that it is close to such a polynomial), and use self-correction for retrieving the values of $A$ (which are needed for the foregoing summation test). Fortunately, a "low-degree test"[37] of complexities similar to those of the summation test does exist (and self-correction is also possible within these complexities). Thus,

---

$\Phi(\alpha, \beta_1, \beta_2, \beta_3) \neq 0$. This triple $(\beta_1, \beta_2, \beta_3)$ encodes the literals appearing in the $\alpha^{\text{th}}$ clause, and this clause is satisfied by $A$ if and only if $\exists i \in [3]$ s.t. $A'(\beta_i) = 1$.

[36]The query will also contain a sequence $(s_1, ..., s_{|\phi|}) \in S$, selected at random (by the verifier) and fixed for the rest of the process.

[37]By a **low-degree test**, we mean an oracle machine that accept any low-degree polynomial (over $F$) with probability 1, and rejects (with probability at least 1/2) any function that is far from all low-degree polynomials. An appropriate test is presented in [195] (see also Exercise 9.23).

using a finite field F of poly(log($n$)) elements, the foregoing yields $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$ for $f(n) \stackrel{\text{def}}{=} O(\log(n) \cdot \log\log(n))$.

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

**Warning: Robustness and PCP of proximity.** 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. [15]), which in turn depends heavily on highly efficient low-degree tests. An alternative approach (cf. [35]) capitalizes of the specific structure of the summation test (as well as on the evident robustness of a simple low-degree test).

**Reflection.** The PCP Theorem asserts a PCP system that obtains simultaneously the minimal possible randomness and query complexity (up to a multiplicative factor, assuming that $\mathcal{P} \neq \mathcal{NP}$). The foregoing construction obtains this remarkable result by combining two different PCPs: the first PCP obtains logarithmic randomness but uses poly-logarithmically many queries, whereas the second PCP uses a constant number of queries but has polynomial randomness complexity. We stress that *each of these two PCP systems is highly non-trivial and very interesting by itself.* We also highlight the fact that these PCPs are combined using a very simple composition method (which refers to auxiliary properties such as robustness and proximity testing).[38]

#### 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

[38] **Advanced comment:** We comment that the composition of PCP systems that lack these extra properties is possible, but is far more cumbersome and complex. In some sense, this alternative composition involves transforming the given PCP systems to ones having properties related to robustness and proximity testing.

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 (the quality 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 in each step we *double the rejection probability 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 (i.e., a PCP 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 $\Sigma$, 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 $\alpha$; that is, the value of $\alpha$ 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,*

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

*For various functions $\tau : \mathbb{N} \to (0,1]$, we will consider the promise problem $\mathtt{gapCSP}_\tau^\Sigma$, 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 $(G,\Phi)$ for which $\mathtt{vlt}(G,\Phi) \geq \tau(|G|)$ holds, where $|G|$ denotes the number of edges in $G$.*

Note that 3SAT is reducible to $\mathtt{gapCSP}_{\tau_0}^{\{1,\ldots,7\}}$ for $\tau_0(m) = 1/m$; see Exercise 9.24. Our goal is to reduce 3SAT (or rather $\mathtt{gapCSP}_{\tau_0}^{\{1,\ldots,7\}}$) to $\mathtt{gapCSP}_c^\Sigma$, for some fixed finite $\Sigma$ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}_c^\Sigma$; see Exercise 9.26. (The relationship between constraint satisfaction problems and the PCP Theorem is further discussed in Section 9.3.3.) The desired reduction of $\mathtt{gapCSP}_{\tau_0}^\Sigma$ to $\mathtt{gapCSP}_{\Omega(1)}^\Sigma$ is obtained by iteratively applying the following reduction logarithmically many times.

**Lemma 9.19** (amplifying reduction of gapCSP to itself): *For some finite $\Sigma$ and constant $c > 0$, there exists a polynomial-time computable function $f$ such that, for* *every instance* $(G,\Phi)$ *of* $\mathtt{gapCSP}^{\Sigma}$*, it holds that* $(G',\Phi')=f(G,\Phi)$ *is an instance of* $\mathtt{gapCSP}^{\Sigma}$ *and the two instances are related as follows:*

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. $|G'|=O(|G|)$*.*

That is, satisfiable instances are mapped to satisfiable instances, whereas instances that violate a $\nu$ fraction of the constraints are mapped to that violate at least a $\min(2\nu,c)$ fraction of the constraints. Furthermore, the mapping increases the number of edges (in the instance) by at most a constant factor. We stress that both $\Phi$ and $\Phi'$ consists of Boolean constraints defined over $\Sigma^2$. Thus, by iteratively applying Lemma 9.19 for a logarithmic number of times, we reduce $\mathtt{gapCSP}^{\Sigma}_{\tau_0}$ to $\mathtt{gapCSP}^{\Sigma}_{\Omega(1)}$ and $\mathtt{3SAT}\in\mathcal{PCP}(\mathtt{log},O(1))$ follows (as detailed in Exercise 9.24 and 9.26).

**Proof Outline:**[39] Before turning to the proof, let us highlight the difficulty that it needs to address. Specifically, the lemma asserts a "violation amplifying effect" (i.e., Items 1 and 2), while maintaining the alphabet $\Sigma$ and allowing only a moderate increase in the size of the graph (i.e., Item 3). Waiving the latter requirements allows a relatively simple proof that mimics (an augmented version of)[40] the "parallel repetition" of the corresponding PCP. Thus, the challenge is significantly decreasing the "size blow-up" that arises from parallel repetition and maintaining a fixed alphabet. The first goal (i.e., Item 3) calls for a suitable derandomization, and indeed we shall use the Expander Random Walk Generator (of Section 8.5.3). Those who read §9.3.2.2 may guess that the second goal (i.e., fixed alphabet) can be handled using the proof composition paradigm. (The rest of the overview is intended to be understood also by those who did not read Section 8.5.3 and §9.3.2.2.)

The lemma is proved by presenting a three-step reduction. The first step is a pre-processing step that makes the underlying graph suitable for further analysis (e.g., the resulting graph will be an expander). 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 increase $\mathtt{vlt}$ by any desired constant factor. This is done by a construction that corresponds to taking a random walk of constant length on the current graph. The latter step also increases the alphabet $\Sigma$, 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 $\Sigma$ 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

[39] For details, see [67].

[40] **Advanced comment:** The augmentation is used to avoid using the Parallel Repetition Theorem of [185]. In the augmented version, with constant probability (say half), a consistency check takes place between tuples that contain copies of the same variable (or query).

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).

We start with the pre-processing step. Our aim in this step is to reduce the input $(G, \Phi)$ of $\texttt{gapCSP}^{\Sigma}$ to an instance $(G_1, \Phi_1)$ such that $G_1$ is a $d$-regular expander graph.[41] Furthermore, each vertex in $G_1$ will have at least $d/2$ self-loops, the number of edges will be 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: essentially, the original vertices are replaced by expanders of size proportional to their degree, and a big (dummy) expander is superimposed on the resulting graph (see Exercise 9.27).

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. The intuition underlying this step is that the probability that a random ($t$-edge long) walk on the expander $G_1$ intersects a fixed set of edges is closely related to the probability that a random sample of ($t$) edges intersects this set. Thus, we may expect such walks to hit a violated edge with probability that is $\min(\Theta(t \cdot \nu), c)$, where $\nu$ is the fraction of violated edges. Indeed, the current step consists of reducing the instance $(G_1, \Phi_1)$ of $\texttt{gapCSP}^{\Sigma}$ to an instance $(G_2, \Phi_2)$ of $\texttt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$ and the following holds:

1. 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.

2. The constraints in $\Phi_2$ refer to each element of $\Sigma'$ as a $\Sigma$-labeling of the ("distance $\leq t$") neighborhood of a vertex (see Figure 9.6), and mandates that the two corresponding labelings (of the endpoints of the $G_2$-edge) are consistent as well as satisfy $\Phi_1$. That is, the following two types of conditions are enforced by the constraints of $\Phi_2$:

(consistency): If vertices $u$ and $w$ are connected in $G_1$ by a path of length at most $t$ and vertex $v$ resides on this path, then the $\Phi_2$-constraint associated with the $G_2$-edge between $u$ and $w$ mandates the equality of the entries corresponding to vertex $v$ in the $\Sigma'$-labeling of vertices $u$ and $w$.

(satisfying $\Phi_1$): If the $G_1$-edge $(v, v')$ is on a path of length at most $t$ starting at $u$, then the $\Phi_2$-constraint associated with the $G_2$-edge that corresponds to this path enforces the $\Phi_1$-constraint that is associated with $(v, v')$.

Clearly, $|G_2| = d^{t-1} \cdot |G_1| = O(|G_1|)$, because $d$ is a constant and $t$ will be set to a constant. (Indeed, the relatively moderate increase in the size of the graph

[41] 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, except for the largest eigenvalue (which equals $d$), all the eigenvalues of the corresponding adjacency matrix have absolute value that is bounded away from $d$. For further details, see §E.2.1.1.

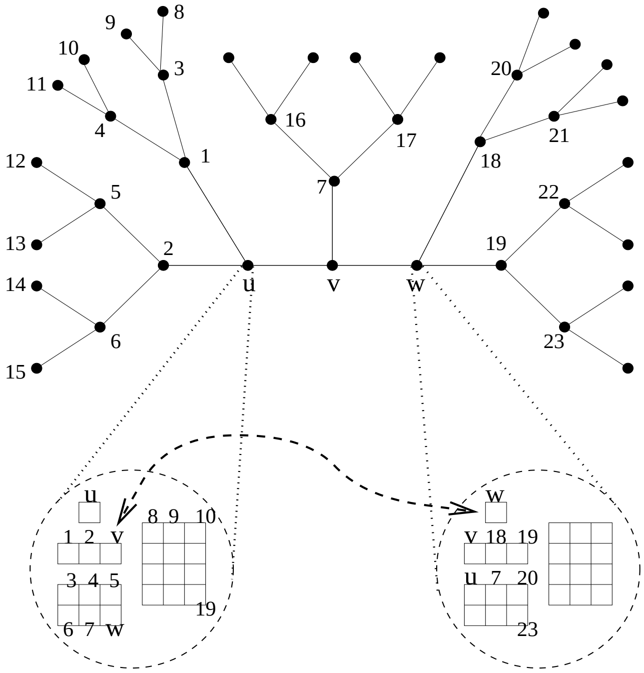


Figure 9.6: The amplifying reduction. The alphabet $\Sigma'$ as a labeling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown (and so the "effective" degree is three). The two-sided arrow indicates one of the edges in $G_1$ that will contribute to the edge constraint between $u$ and $w$ in $(G_2, \Phi_2)$.

corresponds to the low randomness-complexity of selecting a random walk of length $t$ in $G_1$.)

Turning to the analysis of this step, we note that $\mathtt{vlt}(G_1, \Phi_1) = 0$ implies $\mathtt{vlt}(G_2, \Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\mathtt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1)), c)$. Here we merely provide a rough intuition and refer the interested reader to [67]. We may focus on any $\Sigma'$-labeling to the vertices of $G_2$ that is consistent with some $\Sigma$-labeling of $G_1$, because relatively few inconsistencies (among the $\Sigma$-values assigned to a vertex by the $\Sigma'$-labeling of other vertices) can be ignored, while relatively many such inconsistencies yield violation of the "equality constraints" of many edges in $G_2$. Intuitively, relying on the hypothesis that $G_1$ is an expander, it follows that the set of violated edge-constraints (of $\Phi_1$) with respect to the aforementioned $\Sigma$-labeling causes many more edge-constraints of $\Phi_2$ to be violated (because each edge-constraint of $\Phi_1$ is enforced by many edge-constraints of $\Phi_2$). 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.)[42]

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 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 PCP-system[43] (analogously to Exercise 9.26), 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 $r \stackrel{\rm def}{=} \log_2|G_2| + O(d^t \log|\Sigma|)^2$ 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 9.24, for $\Sigma = \{0,1\}^{O(1)}$, 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 (which is used as the output $(G',\Phi')$ of the three-step reduction) is $O(2^r) = O(|G_2|)$, where the equality uses the fact that $d$ and $t$ are constants. Recalling that $\texttt{vlt}(G_2,\Phi_2) \geq \min(\Omega(\sqrt{t}\cdot\texttt{vlt}(G_1,\Phi_1)), c)$ and $\texttt{vlt}(G_1,\Phi_1) = \Omega(\texttt{vlt}(G,\Phi))$, this completes the (outline of the) proof of the entire lemma. □

**Reflection.** In contrast to the proof presented in §9.3.2.2, which combines two remarkable constructs by using a simple composition method, the current proof of the PCP Theorem is based on developing a powerful "combining method" that improves the quality of the main system to which it is applied. This new method, captured by the Amplification Lemma (Lemma 9.19), does not merely obtain the best of the combined systems, but rather obtains a better system than the one given. However, the quality-amplification offered by Lemma 9.19 is rather moderate, and thus many applications are required in order to derive the desired result. Taking the opposite perspective, one may say that remarkable results are obtained by a gradual process of many moderate amplification steps.

### 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 natural approximation problems (cf., Section 10.1.1). To demonstrate this relationship, we first note that any PCP system $V$ gives rise to an approximation problem that consists of estimating the maximum acceptance probability for a given input; that is, on input $x$, the task is approximating the probability that $V$ accepts $x$ when given oracle access to

[42]We mention 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 $\Omega(t\cdot\texttt{vlt}(G_1,\Phi_1))$.

[43]The PCP-system referred to here has arbitrary soundness error (i.e., it rejects the instance $(G_2,\Phi_2)$ with probability $\texttt{vlt}(G_2,\Phi_2) \in [0,1]$).

the best possible $\pi$ (i.e., we wish to approximate $\max_\pi\{\Pr[V^\pi(x)=1]\}$). Thus, *if $S \in \mathcal{PCP}(r,q)$ then deciding membership in $S$ is reducible to approximating the maximum among* $\exp(2^{r+q})$ *quantities* (corresponding to all effective oracles), where each quantity can be evaluated in time $2^r \cdot \text{poly}$. For (the validity of) this reduction, *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 $\max_\pi\{\Pr[V^\pi(x)=1]\}$). 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 $2^{r(|x|)}$ Boolean 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 of constant size.* 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* $\mathtt{gapGSAT}_\varepsilon^q$ *refers to instances that are each a sequence of $q$-variable Boolean formulae* (i.e., each formula depends on at most $q$ variables). *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* $\mathtt{gapSAT}_\varepsilon^q$ *is defined analogously, except that in this case each instance is a sequence of disjunctive clause* (i.e., each formula in each sequence consists of a single disjunctive clause).

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

2. *There exists a constant $q$ such that* $\mathtt{gapGSAT}^q_{1/2}$ *is* $\mathcal{NP}$*-hard.*

3. *There exists a constant $\varepsilon > 0$ such that* $\mathtt{gapSAT}^3_\varepsilon$ *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 9.18 and Section 10.1.1).

**Proof:** We first show that the PCP Theorem implies the NP-hardness of 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 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 $\omega$, that specifies which $q$-bit long sequence of oracle answers makes the verifier accept $x$ on coins $\omega$. 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 $\mathtt{gapGSAT}^q$. 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 $\mathtt{gapGSAT}^q_{1/2}$.

Item 2 implies Item 3 by a standard reduction of GSAT to 3SAT. Specifically, $\mathtt{gapGSAT}^q_{1/2}$ reduces to $\mathtt{gapSAT}^q_{2^{-(q+1)}}$, which in turn reduces to $\mathtt{gapSAT}^3_\varepsilon$ for $\varepsilon = 2^{-(q+1)}/(q-2)$. Note that Item 3 implies Item 2 (e.g., given an instance of $\mathtt{gapSAT}^3_\varepsilon$, 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 argument shows that Item 2 implies Item 1.) This is done by showing that $\mathtt{gapSAT}^3_\varepsilon$ is in $\mathcal{PCP}(\varepsilon^{-1}\log, 3\varepsilon^{-1})$, and using the reduction of $\mathcal{NP}$ to $\mathtt{gapSAT}^3_\varepsilon$ to derive a corresponding PCP for each set in $\mathcal{NP}$. In fact, we show that $\mathtt{gapGSAT}^q_\varepsilon$ is in $\mathcal{PCP}(\varepsilon^{-1}\log, \varepsilon^{-1}q)$, 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 $\varepsilon$ (no matter which oracle is used). To amplify the rejection probability (to the desired threshold of 1/2), we invoke the foregoing verifier $\varepsilon^{-1}$ times (and note that $(1-\varepsilon)^{1/\varepsilon} < 1/2$). ■

**Gap amplifying reductions – a reflection.** Item 2 (resp., Item 3) of Theorem 9.21 implies that GSAT (resp., 3SAT) can be reduce to $\mathtt{gapGSAT}_{1/2}$ (resp., to $\mathtt{gapSAT}^3_\varepsilon$). This means that there exist "gap amplifying" reductions of problems like 3SAT to themselves, where these reductions map yes-instances to yes-instances (as usual), while mapping no-instances to no-instances that are "far" from being yes-instances. That is, no-instances are mapped 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 $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$ and $\mathcal{NP} = \mathcal{PCP}(0, \mathtt{poly})$, respectively.

**Proposition 9.22** *For every $S \in \mathcal{NP}$, there exists a logarithmic function $\ell$* (i.e., $\ell \in \mathtt{log}$) *such that, for every integer function $k$ that satisfies $0 \le k(n) \le \ell(n)$, it holds that $S \in \mathcal{PCP}(\ell - k, O(2^k))$.* (Recall that $\mathcal{PCP}(\mathtt{log}, \mathtt{poly}) \subseteq \mathcal{NP}$.)

**Proof Sketch:** By Theorem 9.16, we have $S \in \mathcal{PCP}(\ell, O(1))$. To show that $S \in \mathcal{PCP}(\ell - k, O(2^k))$, we consider an emulation of the corresponding verifier in which we try all possibilities for the $k(n)$-bit long prefix of its random-tape. □

Following the establishment of Theorem 9.16, numerous variants of the PCP Characterization of NP were explored. These variants refer to a finer analysis of various parameters of probabilistically checkable proof systems (for sets in $\mathcal{NP}$). Following is a brief summary of some of these studies.[44]

**The length of PCPs.** Recall that the effective length of the oracle in any $\mathcal{PCP}(\mathtt{log}, \mathtt{log})$ 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*

[44] With the exception of works that appeared after [90], we provide no references for the results quoted here. We refer the interested reader to [90, Sec. 2.4.4].

*to a poly-logarithmic factor.* (For details see [36, 67].) 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. [119]), whereas only sets in $\mathcal{P}$ have PCPs of logarithmic randomness and amortized query-complexity less than 1.

**Free-bit complexity.** The motivation to the notion of free bits came from the PCP–to–MaxClique connection (see Exercise 9.18 and [29, 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 be emulated by binary queries, but the converse does not necessarily hold.[45] For this reason, "parallel repetition" is highly non-trivial

[45]**Advanced comment:** The source of trouble is the adversarial settings (implicit in the soundness condition), which means that 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

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 Stronger forms of PCP systems for NP

Although the PCP Theorem is famous mainly for its negative applications to the study of natural approximation problems (see Section 9.3.3 and §10.1.1.2), its potential for direct positive applications is fascinating. Indeed, the vision of speeding-up the verification of mundane proofs is exciting, where these proofs may refer to mundane assertions such as the correctness of a specific computation. Enabling such a speed-up requires a strengthening of the PCP Theorem such that it mandates efficient verification time rather than "merely" low query-complexity of the verification task. Such a strengthening is possible.

**Theorem 9.23** (Theorem 9.16 – strengthened): *Every set $S$ in $\mathcal{NP}$ has a PCP system $V$ of logarithmic randomness-complexity, constant query-complexity, and quadratic time-complexity. Furthermore, NP-witnesses for membership in $S$ can be transformed in polynomial-time to corresponding proof-oracles for $V$.*

The furthermore part was already stated in Section 9.3.2 (as a strengthening of Theorem 9.16). Thus, the novelty in Theorem 9.23 is that it provides quadratic verification time, rather than polynomial verification time (where the polynomial may depend arbitrarily on the set $S$). Theorem 9.23 is proved by noting that that the CNF formulae that is obtained by reducing $S$ to 3SAT are highly uniform, and thus the verifier $V$ that is outlined in §9.3.2.2 can be implemented in quadratic time. Indeed, the most time-consuming operation required of $V$ is evaluating the low-degree extension $\Phi$ (of $C_\phi$), which corresponds to the input formula $\phi$, at a few points. In the context of §9.3.2.2, evaluating $\Phi$ in exponential-time suffices (since this means time that is polynomial in $|\phi|$). Theorem 9.23 follows by showing that a variant of $\Phi$ can be evaluated in polynomial-time (since this means time that is polylogarithmic in $|\phi|$); for details, see Exercise 9.30.

**PCPs of Proximity.** Clearly, we cannot expect a PCP system (or any standard proof system for that matter) to have sub-linear verification time (since linear-time is required for merely reading the input). Nevertheless, we may consider a relaxation of the verification task (regarding proofs of membership in a set $S$). In this relaxation the verifier is only required to reject any input that is "far" from

the same binary query depending on the other queries packed with it). This trouble becomes acute in the case of PCPs, because they do not correspond to a full information game. Indeed, in contrast, parallel repetition is easy to analyze in the case of interactive proof systems, because they can be modeled as full information games: this is obvious in the case of public-coin systems, but also holds for general interactive proof systems (see Exercise 9.1).

$S$ (regardless of the alleged proof), and, as usual, accept any input that is in $S$ (when accompanied with an adequate proof). Specifically, in order to allow sub-linear time verification, we provide the verifier $V$ with direct access to the bits of the input (which is viewed as an oracle) as well as with direct access to the usual (PCP) proof-oracle, and require that the following two conditions hold (with respect to some constant $\varepsilon > 0$):

Completeness: For every $x \in S$ there exists a string $\pi_x$ such that, when given access to the oracles $x$ and $\pi_x$, machine $V$ always accepts.

Soundness with respect to proximity $\varepsilon$: For every string $x$ that is $\varepsilon$-far from $S$ (i.e., for every $x' \in \{0,1\}^{|x|} \cap S$ it holds that $x$ and $x'$ differ on at least $\varepsilon|x|$ bits) and every string $\pi$, when given access to the oracles $x$ and $\pi$, machine $V$ rejects with probability at least $\frac{1}{2}$.

Machine $V$ is called a PCP of proximity, and its queries to both oracles are counted in its query-complexity. (Indeed, a PCP of proximity was used in §9.3.2.2, and the notion is analogous to a relaxation of decision problems that is reviewed in Section 10.1.2.)

We mention that *every set in $\mathcal{NP}$ has a PCPs of proximity of logarithmic randomness-complexity, constant query-complexity, and polylogarithmic time-complexity*. This follows by using ideas as underlying the proof of Theorem 9.23 (see also Exercise 9.30).

#### 9.3.4.3 PCP with super-logarithmic randomness

Our focus so far 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 (or rather Theorem 9.23) scales up.[46]

**Theorem 9.24** (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 $\textsc{Ntime}$ 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 [90, 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 [109]

[46] Note that the sketched proof of Theorem 9.23 yields verification time that is quadratic in the length of the input and polylogarithmic in the length of the NP-witness.

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 [109]. Independently of [109], Babai [18] suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games*. Syntactically, Arthur-Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones (cf., [111]). The speed-up result (i.e., $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$) is due to [23] (improving over [18]).

The first evidence to the power of interactive proofs was given by Goldreich, Micali, and Wigderson [100], 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., [101]). 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 [32, 130].

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 [33]. Again, the main motivation came from the zero-knowledge perspective; specifically, presenting 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 was demonstrated by using algebraic methods. The basic technique was introduced by Lund, Fortnow, Karloff and Nisan [162], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [205] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai, Fortnow and Lund [20] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 9.4 follows [205].)

The aforementioned multi-prover proof system of Babai, Fortnow and Lund [20] (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 [21] and Feige, Goldwasser, Lovász, and Safra [73]. 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.* [21] 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.* [21] 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.* [73, 74] 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 [80]), where the refinement is obtained by specifying the randomness and query complexities (see Definition 9.14). Hence, whereas the BFL proof system [20] can be reinterpreted as establishing $\mathcal{NEXP} = \mathcal{PCP}(\mathtt{poly}, \mathtt{poly})$, the work of Feige *et. al.* [74] 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.* [21] implies that $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, \mathtt{polylog})$.)

Interest in the new complexity class became immense since Feige *et. al.* [73, 74] demonstrated its relevance to proving the intractability of approximating some natural 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 the 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 $\mathcal{PCP}(\mathtt{log}, \mathtt{log})$. This challenge was met by Arora and Safra [16]. Actually, they showed that $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, q)$, 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 natural approximation problems. The new challenge was met by Arora, Lund, Motwani, Sudan and Szegedy [15], and is captured by the PCP Characterization Theorem, which asserts that $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$.

Indeed the PCP Characterization Theorem is a culmination of a sequence of impressive works [162, 20, 21, 74, 16, 15]. 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 [15, 16].[47] The alternative proof outlined in §9.3.2.3 is due to Dinur [67].

We 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 [185], the use of the Long-Code [29], and the application of Fourier analysis in this setting [116, 117]. We also highlight the notions of PCPs of proximity and robustness (see [35, 68]).

[47] Our presentation also benefits from the notions of PCPs of proximity and robustness, put forward in [35, 68].

**Computationally-Sound Proof Systems.** Argument systems were defined by Brassard, Chaum and Crépeau [49], with the motivation of providing *perfect* zero-knowledge arguments (rather than zero-knowledge *proofs*) for $\mathcal{NP}$. A few years later, Kilian [145] demonstrated their significance beyond the domain of zero-knowledge by showing that, under some reasonable intractability assumptions, every set in $\mathcal{NP}$ has a computationally-sound proof in which the randomness and communication complexities are poly-logarithmic.[48] 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 [165] suggested a different type of computationally-sound proof systems (which he called CS-proofs).

**Final comment:** The current chapter is a revision of [90, Chap. 2]. In particular, more details are provided here for the main topics, whereas numerous secondary topics discussed in [90, Chap. 2] are not mentioned here (or are only briefly mentioned here). We note that a few of the research directions that were mentioned in [90, Sec. 2.4.4] have received considerable attention in the period that elapsed, and improved results are currently known. In particular, the interested reader is referred to [35, 36, 67] for a study of the length of PCPs, and to [119] for a study of their amortized query complexity. Likewise, a few open problems mentioned in [90, Sec. 2.6.3] have been resolved; specifically, the interested reader is referred to [25, 172] for breakthrough results regarding zero-knowledge.

# Exercises

**Exercise 9.1 (parallel error-reduction for interactive proof systems)** By $t$ parallel repetitions of the proof system $(P, V)$ we mean an interaction in which $t$ copies of the basic system are executed in parallel such that, at the $i^{\text{th}}$ move, the relevant party performs the $i^{\text{th}}$ move for each of these $t$ copies. Needless to say, a honest party (i.e., the verifier) will act in each copy independently of the other copies, but a dishonest prover may determine its action in each copy based on the execution of all copies. Nevertheless, prove that the error probability (in the soundness condition) decreases exponentially with the number of parallel repetitions (of the proof system).

**Guideline:** As a warm-up, consider the special case of public-coin interactive proof systems. Next, generalize the analysis to arbitrary interactive proof systems, by considering (as a mental experiment) a "powerful verifier" that emulates the original verifier while behaving as in the public-coin model. (A direct proof appears in [90, Apdx. C.1].)

**Exercise 9.2** Prove that if $S$ is Karp-reducible to a set in $\mathcal{IP}$, then $S \in \mathcal{IP}$. Prove that if $S$ is Cook-reducible to a set $S'$ such that both $S'$ and $\{0,1\}^* \setminus S'$ are in $\mathcal{IP}$, then $S \in \mathcal{IP}$.

**Exercise 9.3** 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, suppose that the protocol for

[48] We comment that interactive proofs are unlikely to have such low complexities; see [106].

unsatisfiability is applied to a CNF formula with $n$ variables and $m$ clauses. Then, what is the length of the messages sent by the two parties? What is the soundness error?

**Exercise 9.4** Present an interactive proof system for unsatisfiability such that on input a CNF formula having $n$ variables the parties exchange $n/O(\log n)$ messages.

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

**Exercise 9.5 (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 $(1 - ((1-x)\cdot y \cdot (1-z)))$. The point is that this arithmetization maps Boolean assignments that satisfy the CNF formula to 0-1 assignments that when substituted in the corresponding arithmetic expression yield the value 1 (rather than yielding a somewhat arbitrary positive integer).

**Exercise 9.6** Show that `QBF` can be reduced to a special form of (non-canonical)[49] `QBF` in which no variable appears both to the left and to the right of more than one universal quantifier.

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

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

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).[50]

---

[49] See Appendix G.2.

[50] 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

$$\begin{aligned} &\exists z_1 \forall z'_2 \exists z'_1\, [(z'_1 = z_1) \wedge \exists z'_3 \forall z'_4 \exists z''_1 \exists z''_2 \exists z''_3 \\ &\quad [(\wedge_{i=1}^{3} (z''_i = z'_i)) \wedge \exists z'_5 \forall z'_6 \phi(z''_1, z''_2, z''_3, z'_4, z'_5, z'_6)]]. \end{aligned}$$

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

**Exercise 9.7** 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 suppose that $P_1, ..., P_t$ is an enumeration of all the primes that satisfy $a \equiv b \pmod{P_i}$. Using the Chinese Reminder Theorem, prove that $Q \stackrel{\mathrm{def}}{=} \prod_{i=1}^{t} P_i \leq M$ (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 9.8 (on interactive proofs with two-sided error (following [82]))** 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 $x$. Specifically, suppose that a suitable prover may cause every yes-instance to be accepted with probability at least 2/3 (rather than 1), while no cheating prover can cause a no-instance to be accepted with probability greater than 1/3 (rather than 1/2). 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 **??**, 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 $\mathcal{BPP} = \mathcal{AM}'(0)$).

Using the Round Speed-up Theorem (i.e., Theorem **??**), 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 (for Part 2):** Fixing an optimal prover strategy for the given two-sided error public-coin interactive proof, consider the set of verifier coins that make the verifier accept any fixed yes-instance, and apply the ideas underlying the transformation of $\mathcal{BPP}$ to $\mathcal{MA} = \mathcal{AM}(1)$. For further details, see [82].

**Exercise 9.9** In continuation to Exercise 9.8, 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 9.10** Prove that every $\mathcal{PSPACE}$-complete set $S$ has an interactive proof system in which the designated prover can be implemented by a probabilistic polynomial-time oracle machine that is given oracle access to $S$.

**Guideline:** Use Theorem 9.4 and Proposition 9.5.

**Exercise 9.11 (checkers (following [39]))** A probabilistic polynomial-time oracle machine $C$ is called a checker for the decision problem $\Pi$ if the following two conditions hold:

1. For every $x$ it holds that $\mathsf{Pr}[C^{\Pi}(x)=1] = 1$, where (as usual) $C^f(x)$ denotes the output of $A$ on input $x$ when given oracle access to $f$.
2. For every $f : \{0,1\}^* \to \{0,1\}$ and every $x$ such that $f(x) \neq \Pi(x)$ it holds that $\mathsf{Pr}[C^f(x)=1] \leq 1/2$.

Note that nothing is required in the case that $f(x) = \Pi(x)$ but $f \neq \Pi$. Prove that *if both $S_1 = \{x : \Pi(x)=1\}$ and $S_0 = \{x : \Pi(x)=0\}$ have interactive proof systems in which the designated prover can be implemented by a probabilistic polynomial-time oracle machine that is given oracle access to $\Pi$, then $\Pi$ has a checker.* Using Exercise 9.10, conclude that any $\mathcal{PSPACE}$-complete problem has a checker.

**Guideline:** On input $x$ and oracle access to $f$, the checker first obtains $\sigma \stackrel{\text{def}}{=} f(x)$. The claim $\Pi(x) = \sigma$ is then checked by combining the verifier of $S_\sigma$ with the probabilistic polynomial-time oracle machine that describes the designated prover, while referring its queries to the oracle $f$.

**Exercise 9.12 (weakly optimal deciders for checkable problems (following [133]))** Prove that if a decision problem $\Pi$ has a checker (as defined in Exercise 9.11) then there exists a probabilistic algorithm $A$ that satisfies the following two conditions:

1. $A$ solves the decision problem $\Pi$ (i.e., for every $x$ it holds that $\mathsf{Pr}[A(x) = \Pi(x)] \geq 2/3$).
2. For every probabilistic algorithm $A'$ that solves the decision problem $\Pi$, there exists a polynomial $p$ such that for every $x$ it holds that $t_A(x) = p(|x|) \cdot \max_{|x'| \leq p(|x|)}\{t_{A'}(x')\}$, where $t_A(z)$ (resp., $t_{A'}(z)$) denotes the number of steps taken by $A$ (resp., $A'$) on input $z$.

Note that, compared to Theorem 2.33, the claim of optimality is weaker, but on the other hand it applies to decision problems (rather than to candid search problems).

**Guideline:** Use the ideas of the proof of Theorem 2.33, noting that the correctness of the answers provided by the various candidate algorithms can be verified by using the checker. That is, $A$ invokes copies of the checker, while using different candidate algorithms as oracles in the various copies.

**Exercise 9.13 (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., the 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 9.14 (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 9.15 (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 $2^{r(|x|)} \cdot q(|x|)$.

**Guideline:** In the non-adaptive case, all $q$ queries are determined by $V$'s internal coin tosses.

**Exercise 9.16 (on the effective randomness of PCPs)** Suppose that a set $S$ has a PCP of query-complexity $q$ that utilizes proof oracles of length $\ell$. 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 $r(n) = \log_2(\ell(n) + n) + O(1)$. 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 the bits of a non-uniform $\mathrm{poly}(\ell(n) + n)$-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 $R_n$ (of the said size) is bad, where $R_n$ is bad if $V$ accept $x$ with probability $0.5 + \varepsilon$ when selecting its coins in $R_n$ and using the oracle $\pi$.)

**Exercise 9.17 (on the complexity of sets having certain 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[51] that, on input of length $n$, makes at most $2^{r(n)} \cdot q(n)$ truly non-deterministic steps (i.e., choosing between different alternatives) and halts within a total number of $2^{r(n)} \cdot \mathrm{poly}(n)$ steps. Conclude that $S \in \mathrm{NTIME}(2^r \cdot \mathrm{poly}) \cap \mathrm{DTIME}(2^{2^r q + r} \cdot \mathrm{poly})$.

**Guideline:** For each input $x \in S$ and each possible value $\omega \in \{0,1\}^{r(|x|)}$ of the verifier's 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).

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

**Exercise 9.18 (The FGLSS-reduction [74])** 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 $(\omega, \alpha)$ 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 mutually *inconsistent* views 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 $\omega$ (resp., $\omega'$) 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 $G_x$ is upper-bounded by $2^{r(|x|)+f(|x|)}$, where $f \le q$ 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 2.26.)
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 $\pi'$ such that $V$ accepts $x$ when using coins in $R$ and accessing any oracle that is consistent with $\pi'$. 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 9.19** Using the ideas of Exercise 9.18, prove that, for any $t(n) = o(\log n)$, if $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$ then $\mathcal{P} = \mathcal{NP}$.

**Guideline:** We only use the fact that the FGLSS-reduction maps instances of $S \in \mathcal{PCP}(t,t)$ to instances of the `Clique` problem (and ignore the fact that we actually get a stronger reduction to a "gap-Clique" problem). The key observation is that, when applies to $n$-bit long instances of a problem in $\mathcal{PCP}(t,t)$, the FGLSS-reduction runs in polynomial-time and produces instances of size $2^{2t(n)} \ll n$. Thus, the hypothesis $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$ implies that the FGLSS-reduction maps instances of the `Clique` problem to shorter instances of the same problem. Hence, iteratively applying the FGLSS-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 9.20 (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 $\mathsf{Pr}_{x\in G}[f(x) \neq h(x)]$. 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., $\delta(f) = \mathsf{Pr}_{x\in G}[f(x) \neq h(x)]$). Prove that $\varepsilon(f) = \mathsf{Pr}_{x,y\in G}[f(x) + f(y) \neq f(x+y)]$ is lower-bounded by $3 \cdot \mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)]$.

   (Hint: consider three out of four *disjoint* cases (regarding $f(x) \stackrel{?}{=} h(x)$, $f(y) \stackrel{?}{=} h(y)$, and $f(x+y) \stackrel{?}{=} h(x+y)$) 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 $\mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)] \geq \delta(f) - 2\delta(f)^2$.

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

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 9.21.

**Exercise 9.21 (a better analysis of the BLR Linearity Test (cf. [40]))** In continuation to Exercise 9.20, use the following guidelines in order to prove that $\varepsilon(f) \geq \min(1/6, \delta(f)/2)$. Specifically, focusing on the case that $\varepsilon(f) < 1/6$, show that $f$ is $2\varepsilon(f)$-close to some homomorphism (and thus $\varepsilon(f) \geq \delta(f)/2$).

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

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

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

2. Prove that $\phi$ 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 $p_{x,y} \stackrel{\rm def}{=} \mathsf{Pr}_{r\in G}[\phi(x) + \phi(y) \neq \phi(x + y)]$, and showing that $p_{x,y} < 1$ (and hence $\phi(x) + \phi(y) \neq \phi(x + y)$ is false). Prove that $p_{x,y} < 1$, by showing that

$$p_{x,y} \;\leq\; \mathsf{Pr}_r \left[ \begin{array}{l} \phi(x) \neq f(x + r) - f(r) \\ \vee\ \phi(y) \neq f(r) - f(r - y) \\ \vee\ \phi(x + y) \neq f(x + r) - f(r - y) \end{array} \right] \tag{9.10}$$

   and using Item 1 (and some variable substitutions) for upper-bounding by $2\varepsilon(f) < 1/3$ the probability of each of the three events in Eq. (9.10).

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

   **Extra guideline:** Denoting $B = \{x \in G : \mathsf{Pr}_{y\in G}[f(x) \neq \phi_y(x)] \geq 1/2\}$, 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 9.22 (testing matrix identity)** Let $M$ be a non-zero $m$-by-$n$ matrix over GF$(p)$. Prove that $\mathsf{Pr}_{r,s}[r^\top M s \neq 0] \geq (1 - p^{-1})^2$, where $r$ (resp., $s$) is a random $m$-ary (resp., $n$-ary) vector.

**Guideline:** Prove that if $v \neq 0^n$ then $\mathsf{Pr}_s[v^\top s = 0] = p^{-1}$, and that if $M$ has rank $\rho$ then $\mathsf{Pr}_r[r^\top M = 0^n] = p^{-\rho}$.

**Exercise 9.23 (low-degree tests (following [195]))** For a field of prime cardinality $F$ and integers $m$ and $d < |F| - 1$, we consider the set, denoted $P_{m,d}$, of all $m$-variate polynomials of *total degree* at most $d$ over $F$. We consider the low-degree test that, when given oracle access to any function $f : F^m \to F$, selects uniformly $\overline{x}, \overline{y} \in F^m$, queries $f$ at the points $(\overline{x} + i \cdot \overline{y})_{i=0,\ldots,d+1}$, and accepts if and only if $\sum_{i=0}^{d+1} \alpha_i f(\overline{x} + i \cdot \overline{y}) = 0$, where $\alpha_i = (-1)^{i+1} \cdot \binom{d+1}{i}$. It is well-known (cf. [195]) that $f \in P_{m,d}$ if and only if for every $\overline{x}, \overline{y} \in F^m$ it holds that $\sum_{i=0}^{d+1} \alpha_i f(\overline{x} + i \cdot \overline{y}) = 0$.

1. Following the outline of Exercise 9.20, prove that the test rejects $f$ with probability at least $(d + 2) \cdot \delta(f) - (d + 2)(d + 1) \cdot \delta(f)^2$, where $\delta(f) = \min_{g\in P_{m,d}}\{\mathsf{Pr}_{\overline{x}\in F^m}[f(\overline{x}) \neq g(\overline{x})]\}$.

2. Following the outline of Exercise 9.21, prove that $\varepsilon(f) \geq \min((d+2)^{-2}, \delta(f))/2$, where $\varepsilon(f)$ denotes the probability that the test rejects $f$. That is, prove that if $\varepsilon(f) < (d + 2)^{-2}/2$ then $f$ is $2\varepsilon(f)$-close to some function in $P_{m,d}$.

**Guideline:** Define $\phi_y(x) \stackrel{\rm def}{=} \sum_{i=1}^{d+1} \alpha_i f(\overline{x} + i \cdot \overline{y})$, and note that $\varepsilon(f) = \mathsf{Pr}_{\overline{x},\overline{y}\in F^m}[f(\overline{x}) \neq \phi_{\overline{y}}(\overline{x})]$. Part 1 follows by lower-bounding the probability that, for random $\overline{x}, \overline{y} \in F^m$, there exists a unique $i \in \{0, 1, ..., d + 1\}$ such that $f(\overline{x} + i \cdot \overline{y}) \neq g(\overline{x} + i \cdot \overline{y})$, where $g \in P_{m,d}$ is the low-degree polynomial closest to $f$. Part 2 follows by defining $\phi(\overline{x}) =$

$\text{argmax}_{v \in F}\{|\{\overline{y} \in F^m : \phi_{\overline{y}}(\overline{x}) = v\}|\}$, and proceeding analogously to the three steps in the proof of Exercise 9.21. For example, analogously to the first step, prove that for every $\overline{x} \in F^m$ it holds that $\Pr_{\overline{y} \in F^m}[\phi(\overline{x}) = \phi_{\overline{y}}(\overline{x})] \geq 1 - 2(d+1) \cdot \varepsilon(f)$. (Extra hint: Prove that $\Pr_{\overline{y}_1, \overline{y}_2 \in F^m}[\phi_{\overline{y}_1}(\overline{x}) = \phi_{\overline{y}_2}(\overline{x})] \geq 1 - 2(d+1) \cdot \varepsilon(f)$.)[52]

**Exercise 9.24 (3SAT and CSP with two variables)** Show that 3SAT is reducible to $\texttt{gapCSP}_{\tau}^{\{1,\ldots,7\}}$ 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 $\psi$ of 3SAT, consider the graph in which vertices correspond to clauses of $\psi$, 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 9.18.

**Exercise 9.25 (CSP with two Boolean variables)** In contrast to Exercise 9.24, 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 9.26** Show that, for any fixed finite $\Sigma$ 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 $(G, \Phi)$, 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 $i^{\text{th}}$ 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 9.27** For any constant $\Sigma$ 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 $(G, \Phi)$ is reduced to $(G_1, \Phi_1)$ 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., $\phi(\sigma, \tau) = 1$ if and only if $\sigma = \tau$)

[52] In the following probabilistic statements, we shall refer to uniformly distributed $\overline{y}_1, \overline{y}_2 \in F^m$. Note that $\phi_{\overline{y}_1}(\overline{x}) = \sum_{i_1=1}^{d+1} \alpha_{i_1} f(\overline{x} + i_1 \cdot \overline{y}_1)$ which with probability at least $1 - (d-1) \cdot \varepsilon(f)$ equals $\sum_{i_1=1}^{d+1} \alpha_{i_1} \phi_{\overline{y}_2}(\overline{x} + i_1 \cdot \overline{y}_1)$. The latter expression equals $\sum_{i_1=1}^{d+1} \sum_{i_2=1}^{d+1} \alpha_{i_1} \alpha_{i_2} f(\overline{x} + i_1 \cdot \overline{y}_1 + i_2 \cdot \overline{y}_2) = \sum_{i_2=1}^{d+1} \alpha_{i_2} \phi_{\overline{y}_1}(\overline{x} + i_2 \cdot \overline{y}_2)$, which with probability at least $1 - (d-1) \cdot \varepsilon(f)$ equals $\sum_{i_2=1}^{d+1} \alpha_{i_2} f(\overline{x} + i_2 \cdot \overline{y}_2) = \phi_{\overline{y}_2}(\overline{x})$.

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., $\phi(\sigma,\tau)=1$ for all $\sigma,\tau\in\Sigma$). 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 9.28 (free-bit complexity zero)** Note that only sets in co$\mathcal{RP}$ have PCPs of *query* complexity zero. Furthermore, Exercise 9.17 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 9.29 (free-bit complexity one)** In continuation to Exercise 9.28, 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.

**Exercise 9.30 (Proving Theorem 9.23)** Using the following guidelines, provide a proof of Theorem 9.23. Let $S\in\mathcal{NP}$ and consider the 3CNF formulae that are obtained by the standard reduction of $S$ to `3SAT` (i.e., the one provided by the proofs of Theorems 2.21 and 2.22). Decouple the resulting 3CNF formulae into pairs of formulae $(\psi_x,\phi)$ such that $\psi_x$ represents the "hard-wiring" of the input $x$ and $\phi$ represents the computation itself. Referring to the mapping of 3CNF formulae to low-degree extensions presented in §9.3.2.2, show that the low-degree extension $\Phi$ that correspond to $\phi$ can be evaluated in polynomial-time (i.e., polynomial in the length of the input to $\Phi$, which is $O(\log|\phi|)$). Conclude that the low-degree extension that corresponds to $\psi_x\wedge\phi$ can be evaluated in time $|x|^2$. Alternatively, note that it suffices to show that the assignment-oracle $A$ (considered in §9.3.2.2) satisfies $\Phi$ and is consistent with $x$ (and is a low-degree polynomial).

**Guideline:** Note that the circuit constructed in the proof of Theorem 2.21 is highly uniform. In particular, the relation between wires and gates in this circuit can be represented by constant-depth circuits of unbounded fan-in and polynomial-size (i.e., size that is polynomial in the length of the indices of wires and gates).