Foundations and Trends® in
Theoretical Computer Science
3:1

Probabilistic Proof Systems: A Primer

Oded Goldreich

now
the essence of knowledge

Foundations and Trends® in
Theoretical Computer Science
Vol. 3, No. 1 (2007) 1–91
© 2008 O. Goldreich
DOI: 10.1561/0400000023

now
the essence of knowledge

Probabilistic Proof Systems: A Primer

Oded Goldreich

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, oded.goldreich@weizmann.ac.il

Abstract

Various types of probabilistic proof systems have played a central role in the development of computer science in the last couple of decades. These proof systems deviate from the traditional concept of a proof by introducing randomization and interaction into the verification process. Probabilistic proof systems carry an error probability (which is explicitly bounded and can be decreased by repetitions), but they offer various advantages over deterministic proof systems.

This primer concentrates on three types of probabilistic proof systems: interactive proofs, zero-knowledge proofs, and Probabilistically Checkable Proofs (PCP). Surveying the basic results regarding these proof systems, we stress the essential role of randomness in each of them.

Conventions and Organization

Most results surveyed in this text hold unconditionally. However, these results are only interesting if $\mathcal{NP} \neq \mathcal{P}$.

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 definition of NP-proof systems.

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 section are assumed to be constructible in polynomial-time.

Organization. In Section 1, we present the basic definitions and results regarding interactive proof systems. The definition of an interactive proof system is the starting point for a discussion of zero-knowledge proofs, which is provided in Section 2. Section 3, which presents the

basic definitions and results regarding probabilistically checkable proofs (PCP), can be read independently of the other sections.

The study of probabilistic proof system is part of complexity theory (cf. e.g., [27]); in fact, the current text is an abbreviated (and somewhat revised) version of [27, Sec. 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.

The foregoing discussion, as well as the definition provided in Section 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.

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 underlie any proof. (Some readers may find it useful to return to this section after reading Section 1.2.)

1.1.1 A Static Object Vs. 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 upon" (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, withstanding 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 text, 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 to 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.

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" toward 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 skeptical 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 Section 1.1.3.) Note that the verifier is "trusted" to protect his own interests by employing the predetermined verification procedure; indeed, the asymmetry with respect to whom we trust is an artifact of our focus on the verification process (or task). In general, each party is trusted to protect his 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 Section 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 Section 1.5.1).

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 $\mathcal{NP}$ is 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).

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 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.[1] 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 "interactive move"), the corresponding (t-round) interaction transcript (on common input x and randomness r_A and r_B) is $\alpha_1, \beta_1, \ldots, \alpha_t, \beta_t$, where $\alpha_i = A(x, r_A, \beta_1, \ldots, \beta_{i-1})$ and $\beta_i = B(x, r_B, \alpha_1, \ldots, \alpha_i)$. The corresponding final decision of A is defined as $A(x, r_A, \beta_1, \ldots, \beta_t)$.

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 common input x, the strategy may only consider a polynomial in $|x|$ many messages, which are each of $\mathrm{poly}(|x|)$ length.[2] Intuitively, if the other party exceeds an *a priori* (polynomial in $|x|$) upper bound on the total length of the messages that it is allowed to send, then the execution is suspended.

---

[1] An alternative formulation refers to the interactive machines that capture the behavior of each of the parties (see, e.g., [25, 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.

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

Definition 1.1. (Interactive Proof Systems — IP)[3]: An *interactive proof system for a set S* is a two-party game, between a *verifier* executing a *probabilistic polynomial-time strategy*, denoted V, and a *prover* that executes a (computationally unbounded) strategy, denoted P, satisfying the following two conditions:

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

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

The error probability (in the soundness condition) can be reduced by successive applications of the proof system. In particular, repeating the proving process for k times, reduces the probability that the verifier is fooled (i.e., accepts a false assertion) to 2^{-k}, and we can afford doing so for any $k = \text{poly}(|x|)$. Variants on the basic definition are discussed in Section 1.4.

Note that NP-proof systems are obtained as a special case of interactive proof systems by eliminating interaction and randomness (i.e., restricting the communication to be uni-directional (from the prover to the verifier) and restricting the verifier to deterministic strategies). As we shall see next, interaction may be beneficial only if the verifier is probabilistic.

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

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

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, 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 1.2. 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 $S \in \mathcal{NP}$.

Reflection. The uselessness of interacting with a deterministic verifier suggests a general moral by which 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 questions, 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[4] (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).

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

[4] Or, just possesses secret information (regarding the common input).

coNP-set that is not known to have an NP-proof system, and next prove the celebrated result $\mathcal{IP} = \mathcal{PSPACE}$, which provides stronger evidence for the belief that interactive proofs are more powerful than NP-proofs.

1.3.1 A Simple Example

> One day on Olympus, bright-eyed Athena claimed that Nectar poured from the new silver-coated jars tastes less good than Nectar poured from the older gold-decorated jars. Mighty Zeus, who was forced to introduce the new jars by the practically minded 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 1.3. 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 1.3 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, ϕ, from the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$. This ("edge preserving") mapping ϕ, in case it exists, is called an *isomorphism* between the graphs. The following protocol specifies a way of proving that two graphs are not isomorphic, while it is not known whether such a statement can be proved via a non-interactive process (i.e., via an NP-proof system).

Construction 1.3. (Interactive Proof for Graph Non-Isomorphism):

- *Common Input*: A pair of graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
- *Verifier's First Step* (*V1*): The verifier selects at random one of the two input graphs, and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly $\sigma \in \{1, 2\}$, and a random permutation π from the set of permutations over the vertex set V_σ. The verifier constructs a graph with vertex set V_σ and edge set

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

  and sends (V_σ, E) to the prover.
- *Motivating Remark*: If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient algorithm) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic, then a random isomorphic copy of one graph is distributed identically to a random isomorphic copy of the other graph.
- *Prover's Step*: Upon receiving a graph, $G' = (V', E')$, from the verifier, the prover finds a $\tau \in \{1, 2\}$ such that the graph G' is isomorphic to the input graph G_τ. (If both $\tau = 1, 2$

satisfy the condition then τ is selected arbitrarily. In case no $\tau \in \{1,2\}$ satisfies the condition, τ is set to 0.) The prover sends τ to the verifier.

- *Verifier's Second Step* (*V2*): If the message, τ, received from the prover equals σ (chosen in Step V1) then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).

The verifier's strategy in Construction 1.3 is easily implemented in probabilistic polynomial-time. We do not know of a probabilistic polynomial-time implementation of the prover's strategy, but this is not required. The motivating remark justifies the claim that Construction 1.3 constitutes an interactive proof system for the set of pairs of non-isomorphic graphs. Recall that the latter set is not known to be in $\mathcal{NP}$.

1.3.2 The Full Power of Interactive Proofs

The interactive proof system of Construction 1.3 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 $\mathcal{NP} \neq \text{co}\mathcal{NP}$, 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 1.4. (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 1.4

We first show that $\text{co}\mathcal{NP} \subseteq \mathcal{IP}$, by presenting an interactive proof system for the $\text{co}\mathcal{NP}$-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 $\mathcal{IP} \subseteq \mathcal{PSPACE}$.

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

The Starting Point: We prove that $\text{co}\mathcal{NP} \subseteq \mathcal{IP}$ by presenting an interactive proof system for the set of unsatisfiable CNF formulae, which is $\text{co}\mathcal{NP}$-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 1.1. Note that the Boolean formula 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

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

Fig. 1.1 Arithmetization of CNF formulae.

zero. 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 $x_1, \ldots, x_n \in \{0,1\}$) 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:

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi(x_1, \ldots, x_n) \equiv 0 \pmod{q}, \tag{1.1}$$

where ϕ 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 Equation (1.1), 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.)[5] 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.

A Single Iteration (*detailed*): The ith iteration is aimed at proving a claim of the form:

$$\sum_{x_i=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,\ldots,r_{i-1},x_i,x_{i+1},\ldots,x_n) \equiv v_{i-1} \pmod{q}, \quad (1.2)$$

where $v_0 = 0$, and $r_1,\ldots,r_{i-1}$ and v_{i-1} are as determined in previous iterations. The ith 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

$$p_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,\ldots,r_{i-1},z,x_{i+1},\ldots,x_n) \mod q. \quad (1.3)$$

Note that, modulo q, the value $p_i(0) + p_i(1)$ equals the l.h.s of Equation (1.2). 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 $r_i \in \mathrm{GF}(q)$

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

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

$$\sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,\ldots,r_{i-1},r_i,x_{i+1},\ldots,x_n) \equiv v_i \pmod{q}, \quad (1.4)$$

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., Equation (1.1)) holds, the prover can supply the correct polynomials (as determined in Equation (1.3)), 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., Equation (1.2)) is false while the ending claim (i.e., Equation (1.4)) is valid. Indeed, let us focus on the ith iteration, and let v_{i-1} and p_i be as in Equations (1.2) and (1.3), respectively; that is, v_{i-1} is the (wrong) value claimed at the beginning of the ith iteration and p_i is the polynomial representing the expression obtained when stripping the current variable (as in Equation (1.3)). 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 Equation (1.2) 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., Equation (1.4)) 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 $p_i(r_i)$).

This establishes that the set of unsatisfiable CNF formulae has an interactive proof system. Actually, a similar proof system can be used to prove that a given formula has a given number of satisfying assignments, i.e., prove membership in the ("counting") set

$$\{(\phi,k) : |\{\tau : \phi(\tau) = 1\}| = k\}. \quad (1.5)$$

Using adequate reductions, it follows that every problem in $\#\mathcal{P}$ has an interactive proof system (i.e., for every NP-relation R, 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 $\mathcal{PSPACE}$. 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. First, 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). Second, when stripping a summation (or a product), the expression may be a polynomial of high degree (due to nested formal products that may appear in the remaining expression). For example, both phenomena occur in the following expression

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

which equals $\sum_{x=0,1} x^{2^{n-1}} \cdot (1+x)^{2^{n-1}}$. The first difficulty is easy to resolve by using the fact 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 select a random prime q of length that is linear in the length of the original formula, and the two parties consider the arithmetic expression reduced modulo this q. The second difficulty is resolved by noting that $\mathcal{PSPACE}$ is actually reducible to a special form of (non-canonical) QBF in which no variable appears both to the left and to the right of more than one universal quantifier. 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), and accept if and only if the majority of these interactions accept.

Proposition 1.5. 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 Idea. The strategy f can be defined recursively. Specifically, for each partial transcript of the interaction with V, the next message of f is determined such that the probability that V accepts the common input (when the subsequent prover messages are determined by f) is maximized. □

1.4 Variants and Finer Structure: An Overview

In this section we consider several variants on the basic definition of interactive proofs as well as finer complexity measures.

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 toward 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 1.4 are of the public-coin type, whereas this is not the case for the Graph Non-Isomorphism proof system (of Construction 1.3). Thus, although not all natural proof systems are of the public-coin type, by Theorem 1.4 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 Section 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 2.6.

1.4.2 Interactive Proof Systems With Two-sided Error

In Definition 1.1 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 1.4) allowing two-sided error does not increase the power of interactive proofs. See further discussion at the end of Section 1.4.3.

1.4.3 A Hierarchy of Interactive Proof Systems

Definition 1.1 only refers to the *total* computation time of the verifier, and thus allows an arbitrary (polynomial) number of messages to be exchanged. A finer definition refers to the number of messages being exchanged (also called the number of rounds).[6]

Definition 1.6. (The Round-Complexity of Interactive Proofs):

- For an integer function m, the complexity class $\mathcal{IP}(m)$ consists of sets having an interactive proof system in which, on common input x, at most $m(|x|)$ messages are exchanged between the parties.[7]
- For a set of integer functions, M, we let $\mathcal{IP}(M) \stackrel{\text{def}}{=} \bigcup_{m \in M} \mathcal{IP}(m)$. Thus, $\mathcal{IP} = \mathcal{IP}(\mathtt{poly})$.

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 $\mathcal{IP}(0) = \mathrm{co}\mathcal{RP}$.)

Definition 1.6 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 [6] and [33]): 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 $\mathcal{IP}(2)$.

[6] An even finer structure emerges when considering also the total length of the messages sent by the prover (see [31]).

[7] We count the total number of messages exchanged, regardless of the direction of communication. Note that, without loss of generality, the last message is sent by the prover, the penultimate message is sent by the verifier, etc.

- The class $\mathcal{IP}(2)$ contains sets that are not known to be in $\mathcal{NP}$, e.g., Graph Non-Isomorphism (see Construction 1.3). However, under plausible intractability assumptions, $\mathcal{IP}(2) = \mathcal{NP}$ (see [42]).
- If $\text{co}\mathcal{NP} \subseteq \mathcal{IP}(2)$ then the Polynomial-Time Hierarchy collapses (see [15]).

It is conjectured that $\text{co}\mathcal{NP}$ is *not* contained in $\mathcal{IP}(2)$, and consequently that interactive proofs with an unbounded number of message exchanges are more powerful than interactive proofs in which only a bounded (i.e., constant) number of messages are exchanged.[8]

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

The IP-hierarchy (i.e., $\mathcal{IP}(\cdot)$) equals an analogous hierarchy, denoted $\mathcal{AM}(\cdot)$, that refers to public-coin (a.k.a Arthur–Merlin) interactive proofs. That is, for every integer function f, it holds that $\mathcal{AM}(f) = \mathcal{IP}(f)$. For $f \geq 1$, it is also the case that $\mathcal{AM}(2f) = \mathcal{AM}(O(f))$; actually, the aforementioned linear speed-up for $\mathcal{IP}(\cdot)$ is established by combining the following two results:

1. *Emulating $\mathcal{IP}(\cdot)$ by $\mathcal{AM}(\cdot)$*: $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$ [33].
2. *Linear speed-up for $\mathcal{AM}(\cdot)$*: $\mathcal{AM}(2f+1) \subseteq \mathcal{AM}(f+1)$ [6].

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 $\mathcal{IP}(\texttt{poly})$.

The fact that $\mathcal{IP}(O(f)) = \mathcal{IP}(f)$ is proved by establishing an analogous result for $\mathcal{AM}(\cdot)$ demonstrates the advantage of the public-coin

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

setting for the study of interactive proofs. A similar phenomenon occurs when establishing that the IP-hierarchy equals an analogous two-sided error hierarchy [23].

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 Section 1.5.2), in which the soundness of the verifier's verdict depends on assumptions concerning the potential prover(s).

1.5 On Computationally Bounded Provers: An Overview

Recall that our definition of interactive proofs (i.e., Definition 1.1) makes no reference to the computational abilities of the potential prover. This fact has two opposite consequences:

1. The completeness condition does not provide any upper bound on the complexity of the corresponding proving strategy (which convinces the verifier to accept valid assertions).
2. The soundness condition guarantees that, regardless of the computational effort spend by a cheating prover, the verifier cannot be fooled to accept invalid assertions (with probability exceeding the soundness error).

Note that providing an upper-bound on the complexity of the (prescribed) prover strategy P of a specific interactive proof system (P,V) only strengthens the claim that (P,V) is an interactive 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.

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 1.5 asserts that an optimal prover strategy (for convincing any fixed verifier V) can be implemented in polynomial-space, and we cannot expect any better for a generic set in $\mathcal{PSPACE} = \mathcal{IP}$. 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 Section 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 2.4).

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 1.3 has this property.

   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. 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 test its validity by itself (before trying to convince a lazy verifier of this claim).

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 Section 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 Section 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 (see [9, 45]).

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 (see [39] vs. [31]).

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 an 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, in other settings (most notably in cryptography), we may actually wish to limit the gain that other parties may obtain from a proof (and, in particular, limit this gain to the minimal level of being convinced of 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 detailed treatments in [25, 26]. We also mention that, in addition to their direct applicability in cryptography, zero-knowledge proofs serve as a good benchmark for the study of various questions regarding cryptographic protocols.

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.

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.[1]

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

We wish to state that no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). 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; see [26]).

2.1.2 The Basic Definitions

We turn back to the concrete task of defining zero-knowledge. First, 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 and the details are follows.

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 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 2.1. (Perfect Zero-knowledge, Over-simplified)[2]: 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

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

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

[2] 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.

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 (cf. [27, Sec. C.3.1]), we obtain the following definition.

Definition 2.2. (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

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

is a negligible function.[3] We denote by $\mathcal{ZK}$ the class of sets having zero-knowledge interactive proof systems.

Definition 2.2 is a simplified version of the actual definition (presented, e.g., in [25, Sec. 4.3.3]). 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 (poly($|x|$)-bit long) auxiliary input).

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

[3] That is, d vanishes faster that the reciprocal of any positive polynomial (i.e., for every positive polynomial p and for sufficiently large n, it holds that $d(n) < 1/p(n)$). Needless to say, $d(n) \stackrel{\text{def}}{=} 0$ if $S \cap \{0,1\}^n = \emptyset$.

deterministic. The same holds for deterministic provers, provided that we consider "auxiliary-input" zero-knowledge. It can also be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which a single message is sent. Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For further details, see [25, 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 [30]).

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).[4] 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 [25, Sec. 4.5.2] or [50].) But first, we demonstrate the non-triviality of zero-knowledge by 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 $\mathcal{BPP}$).

2.2.1 A Simple Example

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.

[4] 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 $\mathcal{BPP}$.

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.[5]

Construction 2.3. (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 ϕ denote an arbitrary isomorphism between them; that is, ϕ is a 1-1 and onto mapping of the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$.
- *Prover's First Step* (*P1*): The prover selects a random isomorphic copy of G_2, and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation π from the set of permutations over the vertex set V_2, and constructs a graph with vertex set V_2 and edge set

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

  The prover sends (V_2, E) to the verifier.
- *Motivating Remark*: If the input graphs are isomorphic, as the prover claims, then the graph sent in Step $P1$ is isomorphic to both input graphs. However, if the input graphs are not isomorphic then no graph can be isomorphic to both of them.
- *Verifier's First Step* (*V1*): Upon receiving a graph, $G' = (V', E')$, from the prover, the verifier asks the prover to show an isomorphism between G' and one of the input graphs, chosen at random by the verifier. Namely, the verifier uniformly selects $\sigma \in \{1, 2\}$, and sends it to the prover (who is supposed to answer with an isomorphism between G_σ and G').
- *Prover's Second Step* (*P2*): If the message, σ, received from the verifier equals 2 then the prover sends π to the verifier.

---

[5] We mention that Construction 1.3 is zero-knowledge in a restricted sense (i.e., w.r.t the honest verifier), but is not known to be zero-knowledge (in the general sense). In particular, a cheating verifier may abuse the prover in order to learn whether or not G_1 is isomorphic to some third graph (which may be either given to it as auxiliary input or generated by it based on the common input).

Otherwise (i.e., $\sigma \neq 2$), the prover sends $\pi \circ \phi$ (i.e., the composition of π on ϕ, defined as $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$) to the verifier. (Indeed, the prover treats any $\sigma \neq 2$ as $\sigma = 1$. 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 ψ, received from the prover is an isomorphism between G_σ and G' then the verifier outputs 1, otherwise it outputs 0.

The verifier strategy in Construction 2.3 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 2.3 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 σ at random, and in particular obliviously of the graph G' it receives). The view of this verifier can be easily simulated by selecting σ and ψ at random, constructing G' as a random isomorphic copy of G_σ (via the isomorphism ψ), and outputting the triple (G', σ, ψ). Indeed (even in this case), the simulator behaves differently from the prescribed prover (which selects G' as a random isomorphic copy of G_2, via the isomorphism π), but its output distribution is identical to the verifier's view in the real interaction. However, the foregoing description assumes that the verifier follows the prescribed strategy, while in general the verifier may (adversarially) select σ depending on the graph G'. Thus, a slightly more complicated simulation (described next) is required.

A general clarification may be in place. Recall that we wish to simulate the interaction of an arbitrary verifier strategy with the prescribed prover. Thus, this simulator must depend on the corresponding verifier strategy, and indeed we shall describe the simulator while referring to such a generic verifier strategy. Formally, this means that the simulator's program incorporates the program of the corresponding

verifier strategy. Actually, the following simulator uses the generic verifier strategy as a subroutine.

Turning back to the specific protocol of Construction 2.3, the basic idea is that simulator tries to guess σ 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_τ (and thus being able to present an isomorphism between G_τ and G'). Recall that the simulator is analyzed only on yes-instances (i.e., the input graphs G_1 and G_2 are isomorphic). The point is that if G_1 and G_2 are isomorphic, then the graph G' does not yield any information regarding the simulator's guess (i.e., τ).[6] Thus, the value σ selected by the adversarial verifier may depend on G' but not on τ, which implies that $\mathsf{Pr}[\sigma = \tau] = 1/2$. In other words, the simulator's guess (i.e., τ) is correct (i.e., equals σ) with probability $1/2$. Now, if the guess is correct then the simulator can produce an output that has the correct distribution, and otherwise the entire process is repeated.

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).[7]

   This is justified by fixing any outcome of the verifier's coins, and observing that our ("uniform") simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.
2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e.,

[6] Indeed, this observation is identical to the observation made in the analysis of the soundness of Construction 1.3.

[7] 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).

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[8] 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.

2.2.2 The Full Power of Zero-Knowledge Proofs

The zero-knowledge proof system presented in Construction 2.3 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. This proof system will be described while referring to "boxes" in which

[8] A probability is called noticeable if it is greater than the reciprocal of some positive polynomial (in the relevant parameter).

information can be hidden and later revealed. Such boxes can be implemented using one-way functions (see Theorem 2.5).

Construction 2.4. (Zero-knowledge Proof of 3-Colorability, Abstract Description): The description refers to abstract non-transparent boxes that can be perfectly locked and unlocked such that these boxes perfectly hide their contents while being locked.

- *Common Input*: A simple graph $G = (V, E)$.
- *Prover's First Step*: Let ψ be a 3-coloring of G. The prover selects a random permutation, π, over $\{1,2,3\}$, and sets $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, for each $v \in V$. Hence, the prover forms a random relabeling of the 3-coloring ψ. The prover sends to the verifier a sequence of $|V|$ locked and non-transparent boxes such that the vth 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 2.4 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 noticeable 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 2.4 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 2.4, 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).[9] Then, we hand all boxes to the verifier, who 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 its 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.

Thus, it suffices to use boxes that hide their contents almost perfectly (rather than being perfectly opaque). Such boxes can be implemented digitally.

Digital Implementation (*overview*). We implement the abstract boxes (referred to in Construction 2.4) 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 (i.e., functions that are easy to evaluate but hard to invert even in the average-case sense), for details see, e.g., [25, Sec. 4.4.1].

Zero-knowledge Proofs for Other NP-sets. Using the fact that 3-colorability is NP-complete, one can derive (from Construction 2.4)

[9] 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.

zero-knowledge proof systems for any NP-set.[10] Furthermore, these proof systems employ relatively efficient prover strategies.

Theorem 2.5. (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 2.5 (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 (see [50]).

Theorem 2.5 has a dramatic effect on the design of cryptographic protocols (see, e.g., [26]). 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.)

Theorem 2.6. (The Ultimate ZK Theorem): Assuming the existence of (non-uniformly hard) one-way functions, it holds that $\mathcal{IP} = \mathcal{ZK}$.

Loosely speaking, Theorem 2.6 can be proved by recalling that $\mathcal{IP} = \mathcal{AM}(\mathtt{poly})$ 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 2.5 can be invoked for proving it.

[10] 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.

Reflection. The proof of Theorem 2.5 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 2.4). 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.[11]

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 [49].

2.3 Proofs of Knowledge — A Parenthetical Section[12]

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*.

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

[11] Historically, the proof of Theorem 2.5 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 1.4), probabilistically checkable proofs (see the proof of Theorem 3.3), and the study of statistical zero-knowledge (cf. [49]).

[12] Technically speaking, this topic belongs to Section 1, but its more interesting demonstrations refer to zero-knowledge proofs of knowledge — hence its current positioning.

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

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).

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 2.4 (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.*[13] 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"[14] 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 [25, Sec. 4.7.1].

[13] 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 which arise naturally in some (cryptographic) applications.

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

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., [25, 26]). These are enabled by the following general result.

Theorem 2.7. (Theorem 2.5, 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.

3

Probabilistically Checkable Proof Systems

Probabilistically checkable proof systems can be viewed as standard (deterministic) proof systems that are augmented with a probabilistic procedure capable of evaluating the validity of the assertion by examining few locations in the alleged proof. 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 3.1).

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

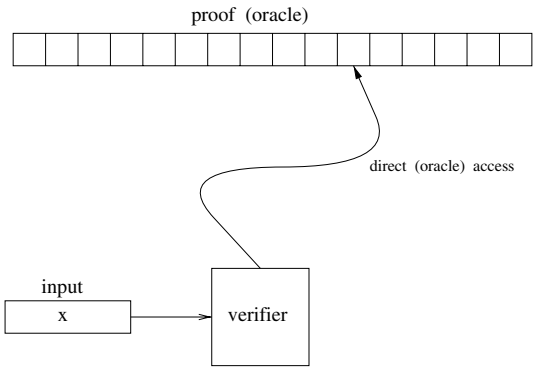


Fig. 3.1 The PCP model — An illustration.

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.

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 3.1. (Probabilistically Checkable Proofs — PCP):

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

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

(2) We say that a probabilistically checkable proof system has query complexity $q : \mathbb{N} \to \mathbb{N}$ if, on any input of length n, the verifier makes at most $q(n)$ oracle queries.[1] Similarly, the randomness complexity $r : \mathbb{N} \to \mathbb{N}$ upper-bounds the number of coin tosses performed by the verifier on a generic n-bit long input.

---

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

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

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

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 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 π_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. 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 vs. Non-adaptive Verifiers. Definition 3.1 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 vs. 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 ω and π such that, on input x, randomness ω and access to the oracle π, the verifier queries location i). The effective proof length of V is the smallest function $\ell:\mathbb{N}\to\mathbb{N}$ 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. 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).

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. In particular, note that $\mathcal{PCP}(0, O(1)) = \mathcal{P}$ holds (as a special case of $\mathcal{PCP}(r,q) \subseteq \text{DTIME}(2^{2^r q + r} \cdot \text{poly})$).

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:

Proposition 3.2. (Upper-bounds on the Power of PCPs): For every polynomially bounded integer function r, it holds that $\mathcal{PCP}(r,\mathtt{poly}) \subseteq \text{NTIME}(2^r \cdot \mathtt{poly})$. 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 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

$$\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log},q) \tag{3.1}$$

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 3.3. (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 Section 3.4.1). Before reviewing the proof of Theorem 3.3, we make a couple of comments.

Efficient Transformation of NP-witnesses to PCP Oracles: The proof of Theorem 3.3 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 there exists a PCP verifier V as in Theorem 3.3 and a polynomial-time computable function π 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 ε then it suffices to examine $O(\log(1/\varepsilon))$ bits of the (transformed) NP-proof. Indeed (as discussed in Section 3.1), these bit locations need to be selected at random.

The foregoing strengthening of Theorem 3.3 offers a wider range of applications than Theorem 3.3 itself. Indeed, Theorem 3.3 itself suffices for "negative" applications such as establishing the infeasibility of certain approximation problems (see Section 3.3). But for "positive" applications (see Section 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 3.3 will be useful.

A Characterization of NP: Combining Theorem 3.3 with Proposition 3.2 we obtain the following characterization of $\mathcal{NP}$.

Corollary 3.4. (The PCP Characterization of NP): $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$.

Road-map for the Proof of the PCP Theorem: Theorem 3.3 is a culmination of a sequence of remarkable works, each establishing meaningful and increasingly stronger versions of Equation (3.1). A presentation of the full proof of Theorem 3.3 is beyond the scope of the current text. Instead, we present an overview of the original proof (see Section 3.2.2) as well as of an alternative proof (see Section 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 3.3 and is also of independent interest. This weaker result (see Section 3.2.1) asserts that *every NP-set has a PCP system with constant query-complexity* (albeit with polynomial randomness complexity), that is, $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{poly}, O(1))$.

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

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.[2] 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.

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 $x_1, \ldots, x_n$, 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

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

of the n-ary vectors α and τ (i.e., $T_1(\alpha)$ is supposed to equal $\sum_{i=1}^{n} \alpha_i \tau_i$). The second part, denoted T_2, is supposed to provide all linear combinations of the values of the $\tau_i \tau_j$'s, that is, for every $\beta \in \mathrm{GF}(2)^{n^2}$ (viewed as an n-by-n matrix over $\mathrm{GF}(2)$), 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. Recall 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 $\mathrm{GF}(2)$. 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, where T is ε-far from being linear if T disagrees with any linear function f on more than an ε fraction of the domain (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] > \varepsilon$).

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 τ (resp., τ').[3] (Needless to say, this does *not* mean that τ' equals the outer product of τ with itself (i.e., $\tau'_{i,j}$ does not necessarily equal $\tau_i \tau_j$).)

[3] Note that τ (resp., τ') 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).

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

Self-correction of the Hadamard Code. Suppose that T is ε-close to a linear function $f : \mathrm{GF}(2)^m \to \mathrm{GF}(2)$ (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] \leq \varepsilon$). Then, we can recover the value of f at any desired point x, by making two (random) queries to T. Specifically, for a uniformly selected $r \in \mathrm{GF}(2)^m$, 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 ε-close to two different linear functions.)[4] Thus, assuming that T_1 is 0.02-close to f_τ (resp., T_2 is 0.02-close to $f_{\tau'}$) we may correctly recover (i.e., with error probability 0.04) the value of f_τ (resp., $f_{\tau'}$) at any desired point by making 2 queries to T_1 (resp., T_2). This process is called self-correction.

Checking Consistency of f_τ and $f_{\tau'}$. Suppose that we are given access to $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$, where $f_\tau(\alpha) = \sum_i \tau_i \alpha_i$ and $f_{\tau'}(\alpha') = \sum_{i,j} \tau'_{i,j} \alpha'_{i,j}$, and that we wish to verify that $\tau'_{i,j} = \tau_i \tau_j$ for every $i, j \in \{1, \ldots, n\}$. 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 that if $A \neq A'$ then $\mathsf{Pr}_{r,s}[r^\top A s \neq r^\top A' s] \geq 1/4$, 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 $r^\top A s = \sum_j (\sum_i r_i \tau_i \tau_j) s_j = f_\tau(r) f_\tau(s)$ and $r^\top A' s = \sum_j (\sum_i r_i \tau'_{i,j}) s_j = f_{\tau'}(r s^\top)$, where $r s^\top$ is the outer-product of s and r. Thus, (for $(\tau_i \tau_j)_{i,j} \neq (\tau'_{i,j})_{i,j}$) we have $\mathsf{Pr}_{r,s}[f_\tau(r) f_\tau(s) \neq f_{\tau'}(r s^\top)] \geq 1/4$.

Recall, however, that we do not have direct access to the functions f_τ 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_τ and $f_{\tau'}$ at any desired point, with very high probability.

[4] 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.

Actually, when implementing the foregoing consistency test it suffices to use self-correction for $f_{\tau'}$, because we use the values of f_τ 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_τ 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 $(\tau_i\tau_j)_{i,j} = (\tau'_{i,j})_{i,j}$.

Checking that τ Satisfies the Quadratic System. Suppose that we are given access to f_τ and $f_{\tau'}$ as in the foregoing (where, in particular, $\tau' = \tau\tau^\top$). A key observation is that if τ does not satisfy a system of (quadratic) equations then, with probability 1/2, it does not satisfy a random linear combination of these equations. Thus, in order to check whether τ satisfies the quadratic system (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 τ. The punch-line is that *testing whether τ 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 τ), then the verifier accepts the corresponding tables T_1 and T_2 (i.e., $T_1 = f_\tau$ and $T_2 = f_{\tau\tau^\top}$) with probability 1. On the other hand, if the 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 $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{poly}, O(1))$.

3.2.2 Overview of the First Proof of the PCP Theorem

The original proof of the PCP Theorem (Theorem 3.3) 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 prohibitively high. (Indeed, we showed such a construction in Section 3.2.1.) Furthermore, this proof system too has additional properties enabling proof composition as in Step 3.

(3) The proof composition paradigm[5]: 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

[5] Our presentation of the composition paradigm follows [12], rather than the original presentation of [1, 2].

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 guarantees 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 3.2 (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 $q'(n)$

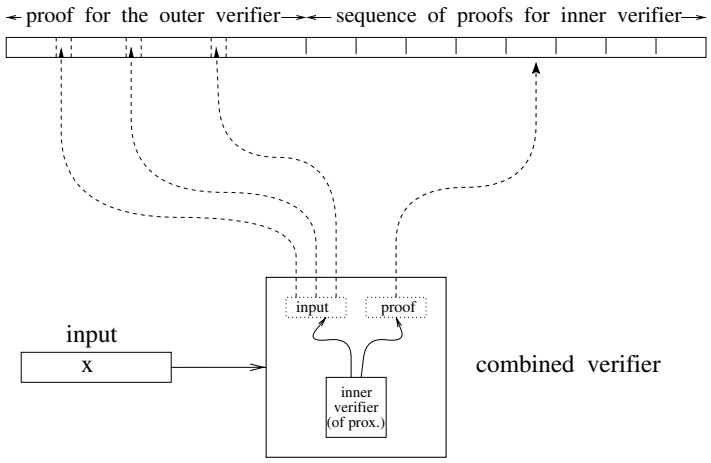


Fig. 3.2 Composition of PCP system. The dashed arrows indicate pointers from the (virtual) input and proof oracles of the inner-verifier to the actual proof of the composed verifier. These pointers (as well as the residual predicate) are determined by an invocation of the outer-verifier.

represents the length of the input (oracle) that is accessed by the inner-verifier. Recall that the outer-verifier is non-adaptive, and thus if the inner-verifier is non-adaptive (resp., robust) then so is the verifier resulting from the composition, which is important in case we wish to compose the latter verifier with another inner-verifier.

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

A More Detailed Overview

The 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 Section 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 Section 3.2.1 in which the quadratic system is fixed[6] 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 Section 3.2.1, and a PCP verifier of proximity that proceeds as in Section 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 $i \in \{1, \ldots, n\}$, we compare the ith 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).

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

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_ω, where $i_{\omega,j}$ is the jth query made (non-adaptively) on coins ω, and P_ω 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 Section 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).

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., Section 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 $\mathrm{poly}(q_1(|x|))$-size circuit, denoted C_ω. 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 jth query made (non-adaptively) on input x and random-tape ω. Note that membership in $C_\omega^{-1}(1)$ can be determined in time $\mathrm{poly}(|C_\omega|) = \mathrm{poly}(q_1(|x|))$. 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_ω or get the description of C_ω as auxiliary input.[7] 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 π 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 $i_{\omega,1}, i_{\omega,2}, \ldots, i_{\omega,q_1(|x|)}$ (which V_1 would query on input x and random-tape ω), 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 jth bit of its input (resp., its proof) then the combined verifier queries the $i_{\omega,j}$th bit of π (resp., the jth bit of $\pi^{(\omega)}$) and provides V_2 with the bit retrieved.

Clearly, if x is a yes-instance then using the adequate proofs π 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 reject" any π 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" π 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 π 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.

PCP of Logarithmic Randomness and Polylogarithmic Query Complexity for $\mathcal{NP}$. We focus on showing that $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, for

[7] In the former case, V_2 is a circuit (with oracle access to its input and proof oracles), which incorporates the circuit C_ω. 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_ω) is given explicitly (as an ordinary input) and the second part (e.g., the input to C_ω) is given implicitly via oracle access. Either way, it is essential that the size of C_ω is polynomial in the length of its own input (i.e., $|C_\omega| = \mathrm{poly}(q_1(|x|))$). 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, α) such that $C(\alpha) = 1$ (where C is presented as explicit input and α is presented as implicit input).

$f(n) = \mathrm{poly}(\log n)$, and the claimed result will follow by a relatively minor modification (discussed afterwards). The proof system underlying $\mathcal{NP} \subseteq \mathcal{PCP}(f,f)$ is based on an arithmetization of 3CNF formulae, which is different from the one used in Section 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 ϕ are represented by binary strings of logarithmic (in $|\phi|$) length, and a *generic* variable (resp., clause) of ϕ is represented by a logarithmic number of *new variables*, which are assigned values in a finite field $\mathrm{F} \supset \{0,1\}$. Indeed, throughout the rest of the description, we refer to the arithmetic operations of this finite field F (which will have cardinality $\mathrm{poly}(|\phi|)$). The (structure of the) 3CNF formula $\phi(x_1,\ldots,x_n)$ 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 ith literal in the αth clause of ϕ 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_ϕ over F, denoted Φ, that is, Φ is the (unique) multi-linear polynomial that agrees with C_ϕ on $\{0,1\}^{O(\log n)} \subset \mathrm{F}^{O(\log n)}$.

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

$$\sum_{\beta_1\beta_2\beta_3 \in \{0,1\}^{3\log 2n}} \Phi(\alpha,\beta_1,\beta_2,\beta_3) \cdot \prod_{i=1}^{3} \big(1 - A'(\beta_i)\big) = 0, \tag{3.2}$$

where $A'(\beta)$ is the value of the βth 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, Equation (3.2) holds if and only if the αth clause is satisfied by the assignment induced by A (because $A'(\beta) = 1$ must hold for at least one of the three literals β that appear in this clause).

As in Section 3.2.1, we cannot afford to verify all $|\phi|$ instances of Equation (3.2). Furthermore, unlike in Section 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 will do.[8] Denoting such a space (of size $\mathrm{poly}(|\phi| \cdot |F|)$ and bias at most 1/6) by $S \subset \mathrm{F}^{|\phi|}$, we may select uniformly $(s_1, \ldots, s_{|\phi|}) \in S$ and check whether

$$\sum_{\alpha\beta_1\beta_2\beta_3 \in \{0,1\}^\ell} s_\alpha \cdot \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} \big(1 - A'(\beta_i)\big) = 0, \tag{3.3}$$

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 Equation (3.2) then, with probability at least 1/3 (taken over the choice of $(s_1, \ldots, s_{|\phi|}) \in S$), it is the case that A fails to satisfy Equation (3.3). Since $|S| = \mathrm{poly}(|\phi| \cdot |F|)$ rather than $|S| = 2^{|\phi|}$, 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 $(s_1, \ldots, s_{|\phi|}) \in S$, Equation (3.3) holds.

Assuming (for a moment) that A is a low-degree polynomial, we can probabilistically verify Equation (3.3) by applying a "summation test" (as in the interactive proof for co$\mathcal{NP}$), that is, we refer to stripping the ℓ 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

[8] Here we refer to a probability space over $\mathrm{F}^{|\phi|}$ that cannot be distinguished from the uniform distribution by any linear test, that is, the bias of the distribution $(\zeta_1, \ldots, \zeta_{|\phi|})$ with respect to the linear test $(t_1, \ldots, t_{|\phi|}) \neq 0^{|\phi|}$ is defined as the absolute value of $\mathsf{E}[\omega^{\sum_{i \in [|\phi|]} t_i \zeta_i}]$, where ω denotes the |F|th complex root of unity. For further details, see [27, Sec. 8.5.2.3].

sequence of choices made so far in the summation test).[9] Note that after stripping the ℓ 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 ℓ queries that are each answered by a univariate polynomial of constant degree (over F), and three queries to A (each answered by an element of F)). Soundness of the summation test follows by setting $|F| \gg O(\ell)$, 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" of complexities similar to those of the summation test does exist (and self-correction is also possible within these complexities). Thus, using a finite field F of $\mathrm{poly}(\log(n))$ elements, the foregoing yields $\mathcal{NP} \subseteq \mathcal{PCP}(f,f)$ for $f(n) \stackrel{\mathrm{def}}{=} O(\log(n) \cdot \log\log(n))$.

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 $\{1,\ldots,\log n\}$, rather than by logarithmically long binary sequences. This requires using low degree polynomial extensions (i.e., polynomial of degree $(\log n)-1$), rather than multilinear extensions. We can still use a finite field of $\mathrm{poly}(\log(n))$ 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 $O(\log n)$ 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 $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, q)$ for $q(n) \stackrel{\mathrm{def}}{=} O(\log^2 n)$.

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

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 Section 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. [1]), which in turn depends heavily on highly efficient low-degree tests. An alternative approach (cf. [12]) capitalizes on 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).[10]

3.2.3 Overview of the Second Proof of the PCP Theorem

The original proof of the PCP Theorem focuses on the construction of two PCP systems that are highly non-trivial and interesting by themselves, and combines them in a natural manner. Loosely speaking, this combination (via proof composition) *preserves* the good features of each of the two systems, that is, it yields a PCP system that inherits the (logarithmic) randomness complexity of one system and the

[10] *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.

(constant) query complexity of the other. In contrast, the following alternative proof is focused on 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 3.5. (CSP with 2-Variable Constraints): For a fixed finite set Σ, an instance of $\texttt{CSP}$ consists of a graph $G = (V, E)$ (which may have parallel edges and self-loops) and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e \in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0,1\}$. The value of an assignment $\alpha : V \to \Sigma$ is the number of constraints satisfied by α, that is, the value of α is $|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 1\}|$. We denote by $\texttt{vlt}(G, \Phi)$ (standing for violation) the fraction of *unsatisfied* constraints under the best possible assignment, that is,

$$\texttt{vlt}(G, \Phi) = \min_{\alpha : V \to \Sigma} \left\{ \frac{|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 0\}|}{|E|} \right\}. \tag{3.4}$$

For various functions $\tau : \mathbb{N} \to (0,1]$, we will consider the promise problem $\texttt{gapCSP}^{\Sigma}_{\tau}$, having instances as above, such that the yes-instances are fully satisfiable instances (*i.e.,* $\texttt{vlt} = 0$) and the no-instances are

pairs (G,Φ) for which $\mathtt{vlt}(G,\Phi) \geq \tau(|G|)$ holds, where $|G|$ denotes the number of edges in G.

Note that 3SAT is reducible to $\mathtt{gapCSP}^{\Sigma_0}_{\tau_0}$ for $\Sigma_0 = \{\mathtt{F},\mathtt{T}\}^3$ and $\tau_0(m) = 1/m$ (e.g., replace each clause by a vertex, and use edge-constraints that enforce mutually consistent and satisfying assignments to each pair of clauses). Our goal is to reduce 3SAT (or rather $\mathtt{gapCSP}^{\Sigma_0}_{\tau_0}$) to $\mathtt{gapCSP}^{\Sigma}_{c}$, for some fixed finite Σ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}^{\Sigma}_{c}$. (The relationship between constraint satisfaction problems and the PCP Theorem is further discussed in Section 3.3.) The desired reduction of $\mathtt{gapCSP}^{\Sigma}_{\tau_0}$ to $\mathtt{gapCSP}^{\Sigma}_{\Omega(1)}$ is obtained by iteratively applying the following reduction logarithmically many times.

Lemma 3.6. (Amplifying Reduction of gapCSP to itself): For some finite Σ and constant $c > 0$, there exists a polynomial-time computable function f such that, for every instance (G,Φ) 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 ν fraction of the constraints are mapped to instances 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 Φ and Φ' consists of Boolean constraints defined over Σ^2. Thus, by iteratively applying Lemma 3.6 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.

Proof Outline.[11] Before turning to the proof, let us highlight the difficulty that it needs to address. Specifically, the lemma asserts a

[11] For details, see [17].

"violation amplifying effect" (i.e., Items 1 and 2), while maintaining the alphabet Σ 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)[12] 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 a "pseudorandom" generator based on random walks on expander graphs. Those who read Section 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 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 `vlt` may decrease during this step by a constant factor. The heart of the reduction is the second step in which we increase `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 Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems, e.g., the one presented in Section 3.2.1). Details follow.

We first stress that the aforementioned Σ and c, as well as the auxiliary parameters d and t (to be introduced in the following two paragraphs), are fixed constants that will be determined such that various conditions (which arise in the course of our argument) are satisfied. Specifically, t will be the last parameter to be determined (and it will be made greater than a constant that is determined by all the other parameters).

We start with the pre-processing step. Our aim in this step is to reduce the input (G, Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1, Φ_1) such that

[12] *Advanced comment*: The augmentation is used to avoid using the Parallel Repetition Theorem of [46]. 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).

G_1 is a d-regular expander graph.[13] 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 $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. 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.

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 ν is the fraction of violated edges. Indeed, the current step consists of reducing the instance (G_1, Φ_1) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\mathtt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$ 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 Φ_2 refer to each element of Σ' as a Σ-labeling of the ("distance $\leq t$") neighborhood of a vertex (see Figure 3.3), and mandates that the two corresponding labelings (of the endpoints of the G_2-edge) are consistent as well as satisfy Φ_1. That is, the following two types of conditions are enforced by the constraints of Φ_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 Φ_2-constraint associated with the G_2-edge between u and w mandates the equality of the

[13] 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.

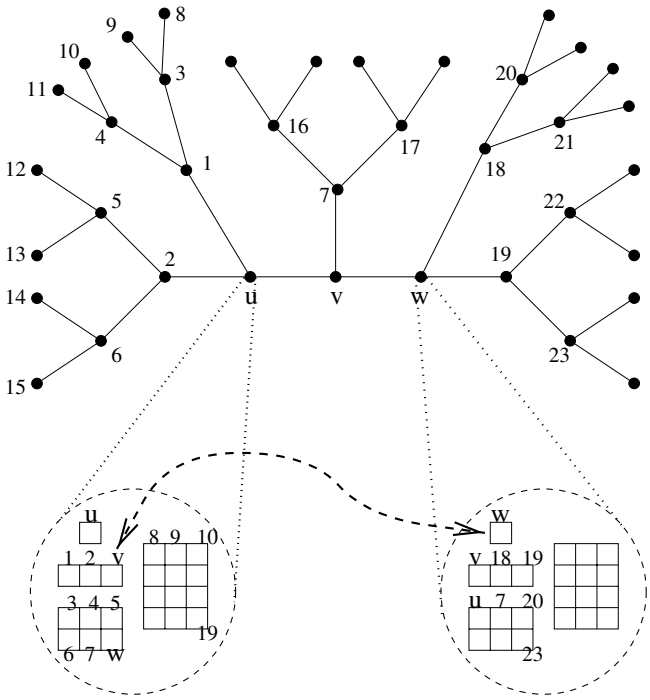


Fig. 3.3 The amplifying reduction. The alphabet Σ' as a labeling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown (and so the "effective" degree is three). The two-sided arrow indicates one of the edges in G_1 that will contribute to the edge-constraint between u and w in (G_2, Φ_2).

entries corresponding to vertex v in the Σ'-labeling of vertices u and w.

Satisfying Φ_1: If the G_1-edge (v, v') is on a path of length at most t starting at u, then the Φ_2-constraint associated with the G_2-edge that corresponds to this path enforces the Φ_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 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 [17]. We may focus on any Σ'-labeling to the vertices of G_2 that is consistent with some Σ-labeling of G_1, because relatively few inconsistencies (among the Σ-values assigned to a vertex by the Σ'-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 Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated (because each edge-constraint of Φ_1 is enforced by many edge-constraints of Φ_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.)[14]

The factor of $\Omega(\sqrt{t})$ gained in the second step makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step). Furthermore, for a suitable choice of the constant t, the aforementioned gain yields an overall constant factor amplification (of $\mathtt{vlt}$). However, so far we obtained an instance of $\mathtt{gapCSP}^{\Sigma'}$ rather than an instance of $\mathtt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\mathtt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\mathtt{gapCSP}^{\Sigma'}$ as a PCP-system,[15] and composing it with an inner-verifier using the proof composition paradigm outlined in Section 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 Section 3.2.1 will do. The resulting PCP-system uses randomness $r \stackrel{\text{def}}{=} \log_2|G_2| + O(d^t\log|\Sigma|)^2$ and a constant number of binary queries, and has rejection probability $\Omega(\mathtt{vlt}(G_2,\Phi_2))$, which is independent of the choice of the constant t. For $\Sigma = \{0,1\}^{O(1)}$, we can obtain an instance of $\mathtt{gapCSP}^{\Sigma}$, that

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

[15] The PCP-system referred to here has arbitrary soundness error (i.e., it rejects the instance (G_2,Φ_2) with probability $\mathtt{vlt}(G_2,\Phi_2)\in[0,1]$).

has a $\Omega(\mathtt{vlt}(G_2, \Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance (which is used as the output (G', Φ') 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 $\mathtt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1)), c)$ and $\mathtt{vlt}(G_1, \Phi_1) = \Omega(\mathtt{vlt}(G, \Phi))$, this completes the (outline of the) proof of the entire lemma. □

Reflection. In contrast to the proof presented in Section 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 3.6), 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 3.6 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.

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. 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 the best possible π (i.e., we wish to approximate $\max_\pi\{\mathsf{Pr}[V^\pi(x)=1]\}$). Thus, *if $S \in \mathcal{PCP}(r, q)$ then deciding membership in S is reducible to approximating the maximum among* $\exp(2^{r+q})$ *quantities* (corresponding to all effective oracles), where each quantity can be evaluated in time $2^r \cdot \mathrm{poly}$. 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 $\max_\pi\{\mathsf{Pr}[V^\pi(x)=1]\}$). This per se does not yield a "natural" approximation problem. In order to link PCP systems with natural approximation problems, we take a closer look at the approximation problem associated with $\mathcal{PCP}(r,q)$.

For simplicity, we focus on the case of non-adaptive PCP systems (i.e., all the queries are determined beforehand based on the input and the internal coin tosses of the verifier). Fixing an input x for such a system, we consider the $2^{r(|x|)}$ 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 3.3) yields *NP-hardness results regarding the approximation of the number of simultaneously satisfiable Boolean formulae of constant size*. This motivates the following definition.

Definition 3.7. (Gap Problems for SAT and Generalized-SAT): For constants $q\in\mathbb{N}$ and $\varepsilon>0$, the promise problem $\mathtt{gapGSAT}^q_\varepsilon$ 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 $\mathtt{gapSAT}^q_\varepsilon$ 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 $\mathtt{gapSAT}^q_\varepsilon$ is naturally viewed as q-CNF formulae, and we consider an assignment that satisfies as many clauses (of the input CNF) as possible. As hinted, $\mathcal{NP}\subseteq\mathcal{PCP}(\mathtt{log},O(1))$ implies that $\mathtt{gapGSAT}^{O(1)}_{1/2}$ is NP-complete, which in turn implies that for some

constant $\varepsilon > 0$ the problem $\mathtt{gapSAT}^3_\varepsilon$ is NP-complete. The converses hold too.

Theorem 3.8. (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 3.8 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.

Proof Sketch. Item 1 implies Item 2 via the argument outlined in the paragraph preceding Definition 3.7, whereas Item 2 implies Item 3 via the standard reduction of `CSAT` ("circuit SAT") to `3SAT`. To see that Item 2 (resp., Item 3) implies Item 1, we present simple PCP systems for the corresponding gap problems (and use their $\mathcal{NP}$-completeness to derive PCP systems for all of $\mathcal{NP}$). For example, we consider a PCP verifier that, when given an instance of $\mathtt{gapGSAT}^q_{1/2}$, selects at random a (q-variable) formula in this instance and inspects the values of the corresponding variables by making adequate queries to the oracle (which is supposed to consist of a satisfying assignment). □

Gap Amplifying Reductions — A Reflection. Item 2 (resp., Item 3) of Theorem 3.8 implies that GSAT (resp., 3SAT) can be reduced to $\mathtt{gapGSAT}^q_{1/2}$ (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.

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.

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

Proof Sketch. By Theorem 3.3, 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 3.3, 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.

The Length of PCPs. Recall that the effective length of the oracle in any $\mathcal{PCP}(\texttt{log},\texttt{log})$ system is polynomial (in the length of the input). Furthermore, in the PCP systems underlying the proof of Theorem 3.3 the queries refer only to a polynomially long prefix of the oracle, and so the actual length of these PCPs for $\mathcal{NP}$ is polynomial. Remarkably, *the length of PCPs for $\mathcal{NP}$ can be made nearly-linear* (in the combined length of the input and the standard NP-witness), *while maintaining constant query complexity, where by nearly linear we mean linear up to a poly-logarithmic factor* (for details see [13, 17]). This means that a *relatively modest amount of redundancy* in the proof oracle suffices for supporting probabilistic verification via a constant number of queries.

The Number of Queries in PCPs. Theorem 3.3 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 arbitrarily close to a soundness error of 1/2 (see [34]). 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 (in 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 $1 + \varepsilon$*, whereas only sets in $\mathcal{P}$ have PCPs of logarithmic randomness and amortized query-complexity less than 1 (see [37] and [8], respectively).

Free-bit complexity. The motivation for the notion of free bits came from the PCP-to-MaxClique connection (see [8, 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 Section 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 (see [35]).

Adaptive vs. 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 3.3) 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 $\mathcal{P}$ (see [34] and [51], respectively).

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.[16] For this reason, "parallel repetition" is highly non-trivial in the PCP setting. Still, a Parallel Repetition Theorem that refers to independent invocations of the same PCP is known [46], 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* (see [18]). (Currently, this is known only for sub-logarithmic answer lengths.)

[16] *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 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.

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, 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 3.10. (Theorem 3.3 — 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 3.2 (as a strengthening of Theorem 3.3). Thus, the novelty in Theorem 3.10 is that it provides quadratic verification time, rather than polynomial verification time (where the polynomial may depend arbitrarily on the set S). Theorem 3.10 is proved by noting that the CNF formulae that are obtained by reducing S to `3SAT` are highly uniform, and thus the verifier V that is outlined in Section 3.2.2 can be implemented in quadratic time. Indeed, the most time-consuming operation required of V is evaluating the low-degree extension Φ (of C_ϕ), which corresponds to the input formula ϕ, at a few points. In the context of Section 3.2.2, evaluating Φ in exponential-time suffices (since this means time that is polynomial in $|\phi|$). Theorem 3.10 follows by showing that a variant of Φ can be evaluated in polynomial-time (since this means time that is polylogarithmic in $|\phi|$).

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 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 π_x such that, when given access to the oracles x and π_x, machine V always accepts.

Soundness with respect to proximity ε: For every string x that is ε-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 π, when given access to the oracles x and π, 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 Section 3.2.2, and the notion is analogous to a relaxation of decision problems that is called "property testing").

We mention that *every set in $\mathcal{NP}$ has a PCP of proximity of logarithmic randomness-complexity, constant query-complexity, and polylogarithmic time-complexity*. This follows by using ideas as underlying the proof of Theorem 3.10.

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 3.10) scales up.[17]

[17] Note that the sketched proof of Theorem 3.10 yields verification time that is quadratic in the length of the input and polylogarithmic in the length of the NP-witness.

Theorem 3.11. (Theorem 3.3 — Generalized): Let $t(\cdot)$ be an integer function such that $n < t(n) < 2^{\mathrm{poly}(n)}$. Then, $\textsc{Ntime}(t) \subseteq \mathcal{PCP}(O(\log t), O(1))$.

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

Bibliographic Notes

Motivated by the desire to formulate the most general type of "proofs" that may be used within cryptographic protocols, Goldwasser, Micali and Rackoff [32] 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 [32]. Independently of [32],[1] Babai [3] suggested a different formulation of interactive proofs, which he called *Arthur–Merlin Games* (and conjectured to be "very close" to $\mathcal{NP}$). Syntactically, Arthur–Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones [33]. The speed-up result (i.e., $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$) is due to [6] (improving over [3]).

[1] Although [32] and [3] have appeared in the same conference (i.e., *17th STOC*, 1985), early versions of [32] have existed as early as 1982, and were rejected three times from major conferences (i.e., *FOCS83*, *STOC84*, and *FOCS84*). In contrast to the motivation of Goldwasser et al. [32], Babai's motivation was placing a group-theoretic problem, previously placed in $\mathcal{NP}$ under some group-theoretic assumptions, "as close to $\mathcal{NP}$ as possible" without using any assumptions.

The first evidence to the power of interactive proofs was given by Goldreich et al. [29], who presented an interactive proof system for Graph Non-Isomorphism (Construction 1.3). 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 2.5). This result has had a dramatic impact on the design of cryptographic protocols (cf., [28]). For further discussion of zero-knowledge and its applications to cryptography, see [25, 26]. Theorem 2.6 (i.e., $\mathcal{ZK} = \mathcal{IP}$) is due to [10, 38].

Probabilistically checkable proof (PCP) systems are related to *multi-prover interactive proof systems*, a generalization of interactive proofs that was suggested by Ben-Or et al. [11]. 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 latter class turned out to be equivalent to a class introduced in [22], which in turn coincides with the current formulation of *probabilistically checkable proofs* (i.e., $\mathcal{PCP}$).

The amazing power of interactive proof systems was demonstrated by using algebraic methods. The basic technique was introduced by Lund et al. [40], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [48] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai et al. [5] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 1.4 follows [48].)

The aforementioned multi-prover proof system of Babai et al. [5] (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 et al. [4] and Feige et al. [20]. 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. [4] 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. [4] 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. [20, 21] 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, where the refinement is obtained by specifying the randomness and query complexities (see Definition 3.1). Hence, whereas the BFL proof system [5] can be reinterpreted as establishing $\mathcal{NEXP} = \mathcal{PCP}(\mathtt{poly},\mathtt{poly})$, the work of Feige et al. [21] establishes $\mathcal{NP} \subseteq \mathcal{PCP}(f,f)$, where $f(n) = O(\log n \cdot \log\log n)$. (We note that the work of Babai et al. [4] implies that $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log},\mathtt{polylog})$.)

Interest in the new complexity class became immense since Feige et al. [20, 21] 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 [2]. 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 et al. [1], 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 [1, 2, 4, 5, 21, 40]. 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 [14, 47], self-correction [14], and pseudorandomness w.r.t linear tests [43]). Our overview of the original proof of the PCP Theorem (in Sections 3.2.1 and 3.2.2) is based on [1, 2].[2] The alternative proof outlined in Section 3.2.3 is due to Dinur [17].

We mention some of the ideas and techniques involved in deriving even stronger variants of the PCP Theorem (which are surveyed in Section 3.4.1). These include the Parallel Repetition Theorem [46], the use of the Long-Code [8], and the application of Fourier analysis in this setting [35, 36]. We also highlight the notions of PCPs of proximity and robustness (see [12, 19]).

Computationally Sound Proof Systems. Argument systems were defined by Brassard et al. [16], with the motivation of providing *perfect* zero-knowledge arguments (rather than zero-knowledge *proofs*) for $\mathcal{NP}$. A few years later, Kilian [39] 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.[3] Interestingly, these argument systems rely on the fact that $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, for $f(n) = \mathrm{poly}(\log n)$. We mention that Micali [41] suggested a different type of computationally sound proof systems (which he called CS-proofs).

Final Comment. The current text is a revision of [24, Sec. 2]. In particular, more details are provided here for the main topics, whereas numerous secondary topics discussed in [24, Sec. 2] are not mentioned

[2] Our presentation also benefits from the notions of PCPs of proximity and robustness, put forward in [12, 19].

[3] We comment that interactive proofs are unlikely to have such low complexities; see [31].

here (or are only briefly mentioned here). We note that a few of the research directions that were mentioned in [24, 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 [12, 13, 17] for a study of the length of PCPs, and to [37] for a study of their amortized query complexity. Likewise, a few open problems mentioned in [24, Sec. 2.6.3] have been resolved; specifically, the interested reader is referred to [7, 44] for breakthrough results regarding zero-knowledge.

Acknowledgments

We are grateful to an anonymous reviewer for carefully reading this text and making many useful suggestions.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and intractability of approximation problems," *Journal of the ACM*, vol. 45, pp. 501–555, 1998, (preliminary version in *33rd FOCS*, 1992).

[2] S. Arora and S. Safra, "Probabilistic checkable proofs: A new characterization of NP," *Journal of the ACM*, vol. 45, pp. 70–122, 1998, (preliminary version in *33rd FOCS*, 1992).

[3] L. Babai, "Trading group theory for randomness," in *17th ACM Symposium on the Theory of Computing*, pp. 421–429, 1985.

[4] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *23rd ACM Symposium on the Theory of Computing*, pp. 21–31, 1991.

[5] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity*, vol. 1, no. 1, pp. 3–40, 1991, (preliminary version in *31st STOC*, 1990).

[6] L. Babai and S. Moran, "Arthur–Merlin games: A randomized proof system and a hierarchy of complexity classes," *Journal of Computer and System Science*, vol. 36, pp. 254–276, 1988.

[7] B. Barak, *Non-Black-Box Techniques in Crypptography*. PhD thesis, Weizmann Institute of Science, 2004.

[8] M. Bellare, O. Goldreich, and M. Sudan, "Free bits, PCPs and non-approximability — towards tight results," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 804–915, 1998, (extended abstract in *36th FOCS*, 1995).

[9] M. Bellare, R. Impagliazzo, and M. Naor, "Does parallel repetition lower the error in computationally sound protocols?," in *38th IEEE Symposium on Foundations of Computer Science*, pp. 374–383, 1997.

[10] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is probable in zero-knowledge," in *Crypto88*, vol. 403, pp. 37–56, Springer-Verlag Notes in Computer Science, 1990.

[11] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-prover interactive proofs: How to remove intractability," in *20th ACM Symposium on the Theory of Computing*, pp. 113–131, 1988.

[12] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, "Robust PCPs of proximity, shorter PCPs, and applications to coding," *SIAM Journal on Computing*, vol. 36, no. 4, pp. 889–974, 2006, (extended abstract in *36th STOC*, 2004).

[13] E. Ben-Sasson and M. Sudan, "Simple PCPs with poly-log rate and query complexity," in *37th ACM Symposium on the Theory of Computing*, pp. 266–275, 2005.

[14] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *Journal of Computer and System Science*, vol. 47, no. 3, pp. 549–595, 1993.

[15] R. Boppana, J. Håstad, and S. Zachos, "Does Co-NP have short interactive proofs?," *Information Processing Letters*, vol. 25, pp. 127–132, May 1987.

[16] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *Journal of Computer and System Science*, vol. 37, no. 2, pp. 156–189, 1988, (preliminary version by Brassard and Crépeau in *27th FOCS*, 1986).

[17] I. Dinur, "The PCP theorem by gap amplification," in *38th ACM Symposium on the Theory of Computing*, pp. 241–250, 2006.

[18] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra, "Characterizations of NP: Towards a polynomially-small error-probability," in *31st ACM Symposium on the Theory of Computing*, pp. 29–40, 1999.

[19] I. Dinur and O. Reingold, "Assignment-testers: Towards a combinatorial proof of the PCP-Theorem," *SIAM Journal on Computing*, vol. 36, no. 4, pp. 975–1024, 2006, (extended abstract in *45th STOC*, 2004).

[20] U. Feige, S. Goldwasser, L. Lovász, and S. Safra, "On the complexity of approximating the maximum size of a clique," Unpublished manuscript, 1990.

[21] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Approximating clique is almost NP-complete," *Journal of the ACM*, vol. 43, pp. 268–292, 1996, (preliminary version in *32nd STOC*, 1991).

[22] L. Fortnow, J. Rompel, and M. Sipser, "On the power of multi-prover interactive protocols," in *3rd IEEE Symposium on Structure in Complexity Theory*, pp. 156–161, 1988. (See errata in *5th IEEE Symposium on Structure in Complexity Theory* pages 318–319, 1990).

[23] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, "On completeness and soundness in interactive proof systems," in *Advances in Computing Research: A Research Annual*, (S. Micali, ed.), pp. 429–442, 1989.

[24] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness,* Algorithms and Combinatorics Series*,* vol. 17. Springer, 1999.

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

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

[27] O. Goldreich, *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[28] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game — A completeness theorem for protocols with honest majority," in *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[29] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *Journal of the ACM*, vol. 38, no. 3, pp. 691–729, 1991, (preliminary version in *27th STOC*, 1986).

[30] O. Goldreich and E. Petrank, "Quantifying knowledge complexity," *Computational Complexity*, vol. 8, pp. 50–98, 1999.

[31] O. Goldreich, S. Vadhan, and A. Wigderson, "On interactive proofs with a laconic provers," *Computational Complexity*, vol. 11, pp. 1–53, 2002.

[32] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, pp. 186–208, 1989, (preliminary version in *17th STOC*, 1985. Earlier versions date to 1982).

[33] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems," in *Advances in Computing Research: A Research Annual*, (S. Micali, ed.), pp. 73–90, 1989, (extended abstract in *18th STOC*, 1986. Extended abstract in *18th STOC*, pp. 59–68, 1986).

[34] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan, "A tight characterization of NP with 3 query PCPs," in *39th IEEE Symposium on Foundations of Computer Science*, pp. 8–17, 1998.

[35] J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," *Acta Mathematica*, vol. 182, pp. 105–142, 1999, (preliminary versions in *28th STOC* (1996) and *37th STOC* (1996)).

[36] J. Håstad, "Getting optimal in-approximability results," *Journal of the ACM*, vol. 48, pp. 798–859, 2001, (extended abstract in *29th STOC*, 1997).

[37] J. Håstad and S. Khot, "Query efficient PCPs with pefect completeness," in *42nd IEEE Symposium on Foundations of Computer Science*, pp. 610–619, 2001.

[38] R. Impagliazzo and M. Yung, "Direct Zero-Knowledge Computations," in *Crypto87*, vol. 293, pp. 40–51, Springer-Verlag Lecture Notes in Computer Science, 1987.

[39] J. Kilian, "A note on efficient zero-knowledge proofs and arguments," in *24th ACM Symposium on the Theory of Computing*, pp. 723–732, 1992.

[40] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *Journal of the ACM*, vol. 39, no. 4, pp. 859–868, 1992, (preliminary version in *31st STOC*, 1990).

[41] S. Micali, "Computationally sound proofs," *SIAM Journal on Computing*, vol. 30, no. 4, pp. 1253–1298, 2000, (preliminary version in *35th STOC*, 1994).

[42] P. B. Miltersen and N. V. Vinodchandran, "Derandomizing Arthur–Merlin games using hitting sets," *Computational Complexity*, vol. 14, no. 3, pp. 256–279, 2005, (preliminary version in *40th STOC*, 1999).

[43] J. Naor and M. Naor, "Small-bias probability spaces: Efficient constructions and applications," *SIAM Journal on Computing*, vol. 22, pp. 838–856, 1993, (preliminary version in *22nd STOC*, 1990).

[44] M. Nguyen, S. J. Ong, and S. Vadhan, "Statistical zero-knowledge arguments for NP from any one-way function," in *47th IEEE Symposium on Foundations of Computer Science*, pp. 3–14, 2006.

[45] K. Pietrzak and D. Wikström, "Parallel repetition of computationally sound protocols, revisited," in *4th TCC,* vol. 4392, pp. 86–102, Springer, Lecture Notes in Computer Science, 2007.

[46] R. Raz, "A parallel repetition theorem," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 763–803, 1998, (extended abstract in *27th STOC*, 1995).

[47] R. Rubinfeld and M. Sudan, "Robust characterization of polynomials with applications to program testing," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 252–271, 1996.

[48] A. Shamir, "IP = PSPACE," *Journal of the ACM*, vol. 39, no. 4, pp. 869–877, 1992, (preliminary version in *31st STOC*, 1990).

[49] S. Vadhan, "A study of statistical zero-knowledge proofs," PhD Thesis, Department of Mathematics, MIT, available from http://www.eecs.harvard.edu/∼salil/papers/phdthesis-abs.html, 1999.

[50] S. Vadhan, "An unconditional study of computational zero knowledge," *SIAM Journal on Computing*, vol. 36, no. 4, pp. 1160–1214, 2006, (extended abstract in *45th STOC*, 2004).

[51] U. Zwick, "Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint," in *9th SODA*, pp. 201–210, 1998.