Probabilistic Proof Systems
(A Survey)*

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

May 1995

Abstract

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last decade. In this exposition, we concentrate on three such proof systems — *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs* — stressing the essential role of randomness in each of them.

*This exposition is an expanded version of a survey written for the proceedings of the International Congress of Mathematicians (*ICM94*) held in Zurich in 1994. It is hoped that this exposition may be accessible to a broad audience of computer scientists and mathematians.

†Partially supported by grant No. 92-00226 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel. Revised and expanded while visiting BRICS, Basic Research in Computer Science, Center of the Danish National Research Foundation.

1 Introduction

The glory given to the creativity required to find proofs, makes us forget that it is the less glorified procedure of verification which gives proofs their value. Philosophically speaking, proofs are secondary to the verification procedure; whereas technically speaking, proof systems are defined in terms of their verification procedures.

The notion of a verification procedure assumes the notion of computation and furthermore the notion of efficient computation. This implicit assumption is made explicit in the definition of $\mathcal{NP}$, in which efficient computation is associated with (deterministic) polynomial-time algorithms.

Definition 1 (NP-proof systems): *Let $S \subseteq \{0,1\}^*$ and $\nu : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}$ be a function so that $x \in S$ if and only if there exists a $w \in \{0,1\}^*$ such that $\nu(x,w) = 1$. If ν is computable in time bounded by a polynomial in the length of its first argument then we say that S is an* NP-set *and ν defines an* NP-proof system.

For example, in propositional calculus a proof is a sequence of assertions, each being a form of an axiom or is obtained by applying an inference rule on previous assertions. Thus, the verification procedure consists of checking the justification of each assertion in the sequence. Clearly, this procedure can be implemented by a very efficient algorithm. In contrast, it is widely believed that there exists no efficient algorithm for finding proofs to given assertions in propositional calculus (since the task is NP-Hard – see below).

Traditionally, NP is defined as the class of NP-sets (cf., [22]). Yet, each such NP-set can be viewed as a proof system. For example, consider the set of satisfiable Boolean formulae. Clearly, a satisfying assignment π for a formula ϕ constitutes an NP-proof for the assertion "ϕ is satisfiable" (the verification procedure consists of substituting the variables of ϕ by the values assigned by π and computing the value of the resulting Boolean expression).

The formulation of NP-proofs restricts the "effective" length of proofs to be polynomial in length of the corresponding assertions (since the running-time of the verification procedure is restricted to be polynomial in the length of the assertion). However, longer proofs may be allowed by padding the assertion with sufficiently many blank symbols. So it seems that NP gives a satisfactory formulation of proof systems (with efficient verification procedures). This is indeed the case if one associates efficient procedures with *deterministic* polynomial-time algorithms. However, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures. In particular,

- Randomized and interactive verification procedures, giving rize to *interactive proof systems*, seem much more powerful (i.e., "expressive") than their deterministic counterparts.

- Such randomized procedures allow the introduction of *zero-knowledge proofs* which are of great theoretical and practical interest.

- NP-proofs can be efficiently transformed into a (redundant) form which offers a trade-off between the number of locations examined in the NP-proof and the confidence in its validity (see *probabilistically checkable proofs*).

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

Basic background from computational complexity

The following are standard complexity classes

- $\mathcal{P}$ denotes the class of sets in which membership can be decided in (deterministic) polynomial-time. Namely, for every $S \in \mathcal{P}$ there exists a (deterministic) polynomial-time algorithm A so that $x \in S$ iff $A(x) = 1$, for all $x \in \{0,1\}^*$. Note that $\mathcal{P}$ is a subset of $\mathcal{NP}$ consiting of these NP-sets for which proofs of membership (i.e., NP-proofs) can be efficiently found (rather than merely exist).
- $\mathcal{RP}$ (resp., $\mathcal{BPP}$) denotes the class of sets in which membership can be decided in probabilistic polynomial-time with one-sided (resp., two-sided) error probability. Specifically,
  - for every $S \in \mathcal{RP}$ there exists a probabilistic polynomial-time algorithm A so that

$$\begin{array}{ll} \text{for every } x \in S & \text{Prob}(A(x)=1) \geq \frac{1}{2} \\ \text{whereas} & \\ \text{for every } x \notin S & \text{Prob}(A(x)=1) = 0 \end{array}$$

    where the probability is taken uniformly over all possible outcomes of the internal coin tosses of algorithm A.
  - for every $S \in \mathcal{BPP}$ there exists a probabilistic polynomial-time algorithm A so that

$$\begin{array}{ll} \text{for every } x \in S & \text{Prob}(A(x)=1) \geq \frac{2}{3} \\ \text{whereas} & \\ \text{for every } x \notin S & \text{Prob}(A(x)=1) \leq \frac{1}{3} \end{array}$$

In both cases, the non-trivial probability bounds may be changed in various ways preserving the complexity class.

- $\mathcal{NP}$ denotes the class of NP-sets and co$\mathcal{NP}$ denotes the class of their complements (i.e., $S \in \text{co}\mathcal{NP}$ iff $\overline{S} \in \mathcal{NP}$, where $\overline{S} \stackrel{\text{def}}{=} \{0,1\}^* - S$).
- A set S is *polynomial-time reducible* to a set T if there exists a polynomial-time computable function f so that $x \in S$ iff $f(x) \in T$, for every x. A set is *NP-hard* if every NP-set is polynomial-time reducible to it. A set is *NP-complete* if it is both NP-hard and in $\mathcal{NP}$.
- $\mathcal{PSPACE}$ denotes the class of sets in which membership can be decided in polynomial-space (i.e., the work-space taken by the decider is polynomial in length of the input).

Obviously, $P \subseteq \mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{PSPACE}$. It is not hard to see that $\mathcal{RP} \subseteq \mathcal{NP}$ and that $\mathcal{NP} \subseteq \mathcal{PSPACE}$. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{NP} \neq \mathcal{PSPACE}$. Furthermore, it is also believed that $\mathcal{NP} \neq \text{co}\mathcal{NP}$. NP-hard sets (or tasks) are assumed to be infeasible, since if an NP-hard set is in $\mathcal{P}$ then $\mathcal{NP} = \mathcal{P}$ (by virtue of the reductions of all NP-sets to each NP-hard set).

Conventions

When presenting a proof system, we state all complexity bounds in terms of the length of the assertion to be proven (which is viewed as an input to the verifier). Namely, polynomial-time means time polynomial in the length of this assertion. Note that this convention is consistent with our definition of NP-proofs.

Denote by `poly` the set of all integer functions bounded by a polynomial and by `log` the set of all integer functions bounded by a logarithmic function (i.e., $f \in \texttt{log}$ iff $f(n) = O(\log n)$).

Basic Background from combinatorics

A (simple) graph, G, is a pair (V, E) where E is a set of 2-subsets of V; i.e., for every $e \in E$ it holds $|e \cap V| = 2$. The elements of V are called *vertices* and the elements of E are called *edges*. In this exposition we consider only simple finite graphs.

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 so that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$. The ("edge preserving") mapping ϕ, if existing, is called an *isomorphism* between the graphs.

A graph $G = (V, E)$ is said to be *3-colorable* if there exists a function $\pi : V \mapsto \{1, 2, 3\}$ so that $\pi(v) \neq \pi(u)$ for every $\{u, v\} \in E$. Such a function, π, is called a *3-coloring* of the graph.

2 Interactive Proof Systems

In light of the growing acceptability of randomized and distributed computations, it is only natural to associate the notion of efficient computation with probabilistic and interactive polynomial-time computations. This leads naturally to the notion of interactive proof systems 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 "tricky" questions asked by the verifier to which the prover has to reply "convincingly". The above discussion, as well as the following definition, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proofs).

2.1 Definition

Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion. Specifically, the verifier is probabilistic polynomial-time. It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). On the other hand, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter what strategy is being employed by the prover. A sketch of the formal definition is given in Item (1) below. Items (2) and (3) introduce additional complexity measures which can be ignored in first reading.

Definition 2 (Interactive Proofs – IP) [30]:

1. *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 *which executes a computationally unbounded strategy* (denoted P)*, satisfying*
   - 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 potential strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$, after interacting with P^* on common input x.*
2. *Let m and r be integer functions. The complexity class $\mathcal{IP}(m(\cdot), r(\cdot))$ consists of sets having an interactive proof system in which, on common input x, the verifier makes at most $r(|x|)$ coin tosses and the total number of messages exchanged between the parties is bounded by $m(|x|)$.*
3. *Let M and R be sets of integer functions. Then, $\mathcal{IP}(M, R)$ denotes $\cup_{m\in M, r\in R}\mathcal{IP}(m(\cdot), r(\cdot))$. Finally, $\mathcal{IP}(m(\cdot)) \stackrel{\rm def}{=} \mathcal{IP}(m(\cdot), \texttt{poly})$ and $\mathcal{IP} \stackrel{\rm def}{=} \mathcal{IP}(\texttt{poly})$.*

In Item (1), we have followed the standard definition which specifies strategies for both the verifier and the prover. An alternative presentation only specifies the verifier's strategy while rephrasing the completeness condition as follows:

> there exists a prover strategy P so that, for every $x \in S$, the verifier V always accepts after interacting with P on common input x.

Arthur-Merlin games[1] introduced in [4] are a special case of interactive proofs; yet, as shown in [31], this restricted case has essentially[2] the same power as the general case previously introduced in [30]. Also, in some sources interactive proofs are defined so that two-sided error probability is allowed; yet, this does not increase their power [21].

2.2 The role of randomness

Randomness is essential to the formulation of interactive proofs; if randomness is not allowed (or if it is allowed but zero error is required in the soundness condition) then interactive proof system collapse to NP-proof systems (i.e., $\mathcal{IP}(\mathtt{poly}, 0)$ equals $\mathcal{NP}$). The reason being that the prover can predict the verifier's part of the interaction and thus it suffices to let the prover send the full transcript of the interaction and let the verifier check that the interaction is indeed valid. (In case the verifier is not deterministic, the transcript sent by the prover may not match the outcome of the verifier coin tosses.) The moral is that there is no point to interact with predictable parties which are also computationally weaker[3].

2.3 The power of interactive proofs

A simple example demonstrating the power of interactive proofs follows. Specifically, we present an interactive proof for proving that two graphs are not isomorphic. It is not known whether such a statement can be proven via an NP-proof system.

Construction 1 (Interactive proof system for Graph Non-Isomorphism) [24]:

- Common Input: *A pair of two graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$*. Suppose, without loss of generality, that* $V_1 = \{1, 2, ..., |V_1|\}$*, and similarly for* V_2*.*

[1] In Arthur-Merlin games, the verifier must send the outcome of any coin it tosses (and thus need not send any other information).

[2] Here and in the next sentence, not only $\mathcal{IP}$ remains invariant under the various definitions, but also $\mathcal{IP}(m(\cdot))$, for every integer function satisfying $m(n) \geq 2$ for every n. However, it is not known whether $\mathcal{IP}(m(\cdot), r(\cdot))$ is preserved as well.

[3] This moral represents the prover's point of view. Certainly, from the verifier's point of view it is benefitial to interact with the prover, since it is computationally stronger.

- 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*
$$E \stackrel{\rm def}{=} \{\{\pi(u),\pi(v)\} : \{u,v\} \in E_\sigma\}$$
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* $\tau \in \{1,2\}$ *so 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 presented above is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's strategy, but this is not required. The motivating remark justifies the claim that Construction 1 constitutes an interactive proof system for the set of pairs of non-isomorphic graphs, which is a coNP-set (not known to be in $\mathcal{NP}$).

Interactive proofs are powerful enough to prove any coNP assertion (e.g., that a graph is not 3-colorable) [36]. 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 [43].

Theorem 1 [43]: $\mathcal{IP} = \mathcal{PSPACE}$.

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

Concerning the finer structure of the IP hierarchy it is known that this hierarchy has a "linear speed-up" property [7]. Namely, for every integer function, f, so 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 $\mathcal{IP}(2)$. It is conjectured that co$\mathcal{NP}$ is *not* contained in $\mathcal{IP}(2)$, and consequently that interactive proofs with unbounded number of message exchanges are more powerful than interactive proofs in which only a bounded (i.e., constant) number of messages are exchanged. Yet, the class $\mathcal{IP}(2)$ contains sets not known to be in $\mathcal{NP}$; e.g., Graph Non-Isomorphism (as shown above).

2.4 How powerful should the prover be?

Assume that a set S is in $\mathcal{IP}$. This means that there is a verifier V that can be convinced to accept any input in S but cannot be convinced to accept any input not in S (except with small probability). One may ask how powerful should a prover be so that it can convince the verifier V to accept any input in S. More interestingly, considering all possible verifiers which give rise to an interactive proof system for S, what is the minimum power required from a prover which satisfies the completeness requirement with respect to one of these verifiers? We stress that, unlike the case of computationally-sound proof systems (see Sec. 5), 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 which meet the completeness condition. The term 'relatively efficient prover' has been given three different interpretations.

1. A prover is considered *relatively efficient* if, when given an auxiliary input (in addition to the common input in S), it works in (probabilistic) polynomial-time. Specifically, in case $S \in \mathcal{NP}$, the auxiliary input maybe an NP-proof that the common input is in the set[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 conducted some computation correctly resulting in some string x. In these cases the NP-proof is just the transcript of the procedure by which x has been computed and thus the auxiliary input is available to the proving party. See [24].

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 has this property.) This interpretation generalizes the notion of self-reducibility of NP-sets. (By self-reducibility of an NP-set we mean that the search problem of finding an NP-witness is polynomial-time reducible to deciding membership in the set.) See [10].

3. A prover is considered *relatively efficient* if it can be implemented by a probabilistic machine which runs in time which is polynomial in the deterministic complexity of the set. This interpretation relates the difficulty of convincing a "lazy verifier" to the complexity of finding the truth alone. Hence, in contrast to the first interpretation which is adequate in settings where assertions are generated along with their NP-proofs, the current interpretation is adequate in settings in which the prover is given only the assertion and has to find a proof to it by itself (before trying to convince a lazy verifier of its validity). See [38].

[4]Still, even in this case the interactive proof need not consist of the prover sending the auxiliary input to the verifier; e.g., an alternative procedure may allow the prover to be zero-knowledge (see Construction 2).

3 Zero-Knowledge Proof Systems

Zero-knowledge proofs, introduced in [30], are central to cryptography. Furthermore, zero-knowledge proofs are very intruiging from a conceptual point of view, since they exhibit an extreme contrast between being convinced of the validity of a statement and learning anything in addition while receiving such a convincing proof. Namely, zero-knowledge proofs have the remarkable property of being both convincing while yielding nothing to the verifier, beyond the fact that the statement is valid. Formally, the fact that "nothing is gained by the interaction" is captured by stating that whatever the verifier can efficiently compute after interacting with a zero-knowledge prover, can be efficiently computed from the assertion itself without interacting with anyone.

3.1 A sample definition

Zero-knowledge is a property of some interactive proof systems, or more acurately of some specified prover strategies. The formulation of the zero-knowledge condition considers two ensembles of probability distributions, each ensemble associates a probability distribution to each valid assertion. The first ensemble respresents the output distribution of the verifier after interacting with the prover strategy P, where the verifier is not necessarily employing the specified strategy (i.e., V) – but rather any efficient strategy. The second ensemble represents the output distribution of some probabilistic polynomial-time algorithm (which does not interact with anyone). The basic paradigm of zero-knowledge asserts that for every ensemble of the first type there exist a "similar" ensemble of the second type. The specific variants differ by the interpretation given to 'similarity'. The most strict interpretation, leading to *perfect zero-knowledge*, is that similarity means equality. Namely,

Definition 3 (perfect zero-knowledge) [30]: *A prover strategy, P, is said to be* perfect zero-knowledge *over a set S if for every probabilistic polynomial-time verifier strategy, V^*, there exists a probabilistic polynomial-time algorithm, M^*, such that*

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

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

A somewhat more relaxed interpretation, leading to *almost-perfect 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). Since the notion of computational indistinguishability is a fundamental one, it is indeed in place to present a definition of it.

Definition 4 (computational indistinguishability) [29, 44]: *An integer function, f, is called* negligible *if for every positive polynomial p and all sufficiently large n, it holds that* $f(n) < \frac{1}{p(n)}$. (Thus, multiplying a negligible function by any fixed polynomial yields a negiligible function.)
Two probability ensembles, $\{A_x\}_{x\in S}$ and $\{B_x\}_{x\in S}$, are indistinguishable *by an algorithm D if*

$$d(n) \stackrel{\rm def}{=} \max_{x\in S\cap\{0,1\}^n}\{|prob(D(A_x)=1) - \mathrm{Prob}(D(B_x)=1)|\}$$

is a negligible function. The ensembles $\{A_x\}_{x\in S}$ and $\{B_x\}_{x\in S}$ are computationally indistinguishable *if they are indistinguishable by every probabilistic polynomial-time algorithm.*

The definitions presented above are a simplified version of the actual definitions. For example, in order to guarantee that zero-knowledge is preserved under sequential composition it is necessary to slightly augment the definitions. For details see [26].

3.2 The power of zero-knowledge

A simple example, demonstrating the power of zero-knowledge proofs, follows. Specifically, we will present a simple zero-knowledge proof for proving that a graph is 3-colorable. The interactive proof will be described using "boxes" in which information can be hidden and later revealed. Such "boxes" can be implemented using one-way functions (see below).

Construction 2 (Zero-knowledge proof of 3-colorability) [24]:

- 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 $\phi(v) \stackrel{\rm def}{=} \pi(\psi(v))$, for each $v \in V$. Hence, the prover forms a random relabelling of the 3-coloring ψ. The prover sends the verifier a sequence of $|V|$ locked and nontransparent boxes so that the $v^{\rm th}$ box contains the value $\phi(v)$;*
- Verifier's first step: *The verifier uniformly selects an edge $\{u,v\} \in E$, and sends it to the prover;*
- Motivating Remark: *The verifier asks to inspect the colors of vertices u and v;*
- Prover's second step: *The prover sends to the verifier the keys to boxes u and v;*
- Verifier's second step: *The verifier opens boxes u and v, and accepts if and only if they contain two different elements in $\{1,2,3\}$;*

The verifier strategy presented above is easily implemented in probabilistic polynomail-time. The same holds with respect to the prover's strategy, provided it is given a 3-coloring of G as auxiliary input. Clearly, if the input graph is 3-colorable then the prover can cause the verifier to accept always. On the other hand, if the input graph is not 3-colorable then any contents put in the boxes must be invalid on at least one edge, and consequently the verifier will reject with probability at least $\frac{1}{|E|}$. Hence, the above game exhibits a non-negligible gap in the accepting probabilities between the case of 3-colorable graphs and the case of non-3-colorable graphs. To increase the gap, the game may be repeated sufficiently many times (of course, using independent coin tosses in each repetition). The zero-knowledge property follows easily, in this abstract setting, since one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. This indeed demonstrates that the verifier learns nothing from the interaction (since it expects to see a random pair of different colors and indeed this is what it sees). We stress that this simple argument is not possible in the digital implementation since the boxes are not totally ineffected by their contents (but are rather effected, yet in an indistinguishable manner).

As stated above, the "boxes" need to be implemented digitally, and this is done using an adaquately defined "commitment scheme". Loosely speaking, such a scheme is a two phase game beteen a sender and a receiver so 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. 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. Such commitment schemes can be implemented assuming the existence of one-way functions (i.e., loosely speaking, functions that are easy to compute but hard to invert, such as multiplication of two large primes) [39, 32].

Using the fact that 3-colorability is NP-complete, one gets zero-knowledge proofs for any NP-set.

Theorem 2 [24]: *Assuming the existence of one-way functions, any NP-proof can be efficiently transformed into a* (computational) *zero-knowledge interactive proof.*

Thm. 2 has a dramatic effect on the design of cryptographic protocols (cf., [24, 25]). In a different vein and for the sake of elegancy, we mention that, using further ideas and under the same assumption, any interactive proof can be efficiently transformed into a zero-knowledge one [33, 11].

The above results may be contrasted with the results regarding the complexity of *almost-perfect* zero-knowledge proof systems; namely, that almost-perfect zero-knowledge proof systems exist only for sets in $\mathcal{IP}(2) \cap \mathrm{co}\mathcal{IP}(2)$ [19, 1], and thus are unlikely to exist for all NP-sets. Also, a very recent result seems to indicate that one-way functions are essential for the existence of zero-knowledge proofs for "hard" sets (i.e., sets which cannot be decided in average polynomial-time)[40].

3.3 The role of randomness

Again, randomness is essential to all the above mentioned (positive) results. Namely, if either verifier or prover is required to be deterministic then only BPP-sets can be proven in a zero-knowledge manner [26]. However, BPP-sets have trivial zero-knowledge proofs in which the prover sends nothing and the verifier just test the validity of the assertion by itself[5]. Thus, randomness is essential to the usefulness of zero-knowledge proofs.

4 Probabilistically Checkable Proof Systems

When viewed in terms of an interactive proof system, the probabilistically checkable proof setting consists of a prover which is memoryless. Namely, one can think of the prover as being an oracle and of the messages sent to it as being queries. A more appealing interpretation is to view the probabilistically checkable proof setting as an alternative way of generalizing $\mathcal{NP}$. Instead of receiving the entire proof and conducting a deterministic polynomial-time computation (as in the case of $\mathcal{NP}$), the verifier may toss coins and query the proof only at location of its choice. Potentially, this allows the verifier to utilize very long proofs (i.e., of super-polynomial length) or alternatively examine very few bits of an NP-proof.

4.1 Definition

Loosely speaking, a probabilistically checkable proof system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a proof in redundent 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. Again, it is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adaquate oracle); whereas, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter which oracle is used. The basic definition of the PCP setting is given in Item (1) below. Yet, the complexity measures introduced in Items (2) and (3) are of key importance for the subsequent discussions, and should not be ignored.

Definition 5 (Probabilistic Checkable Proofs – PCP):

1. *A* probabilistic checkable proof system (pcp) for a set *S is a probabilistic polynomial-time oracle machine* (called verifier)*, denoted V, satisfying*

   - Completeness*: For every $x \in S$ there exists an oracle set π_x so that V, on input x and access to oracle π_x, always accepts x.*

[5] Actually, this is slightly inaccurate since the resulting "interactive proof" may have two-sided error, whereas we have required interactive proofs to have only one-sided error. Yet, since the error can be made negligible by successive repetitions this issue is insignificant. Alternatively, one can use ideas in [21] to eliminate the error by letting the prover send some random-looking help.

- Soundness: *For every $x \notin S$ and every oracle set π, machine V, on input x and access to oracle π, rejects x with probability at least $\frac{1}{2}$.*

2. *Let r and q be integer functions. The complexity class $\mathcal{PCP}(r(\cdot), q(\cdot))$ consists of sets having a probabilistic checkable proof system in which the verifier, on any input of length n, makes at most $r(n)$ coin tosses and at most $q(n)$ oracle queries. We stress that here, as usual in complexity theory, the oracle answers are always binary (i.e., either 0 or 1).*

3. *Let R and Q be sets of functions. Then, $\mathcal{PCP}(R, Q)$ denotes $\cup_{r\in R, q\in Q}\mathcal{PCP}(r(\cdot), q(\cdot))$.*

The above model was suggested in [20] and shown related to a multi-prover model introduced previously in [12]. The fine complexity measures were introduced and motivated in [17], and further advocated in [3]. A related model was presented in [6], stressing the applicability to program checking.

We stress that the oracle π_x in a pcp system constitutes a proof in the standard mathematical sense[6]. Yet, this oracle has the extra 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).

4.2 The power of probabilistically checkable proofs

Clearly, $\mathcal{PCP}(\mathtt{poly}, 0)$ 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. In particular,

Proposition 1 : *$\mathcal{PCP}(\mathtt{log}, \mathtt{poly})$ is contained in $\mathcal{NP}$.*

These upper bounds turn out to be tight, but proving this is much more difficult (to say the least). The following result is a culmination of a sequence of great works [5, 6, 17, 3, 2].[7]

Theorem 3 [2]: *$\mathcal{NP}$ is contained in $\mathcal{PCP}(\mathtt{log}, O(1))$.*

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 the complexity class $\mathcal{NP}$. It follows that NP-proofs can be transformed into NP-proofs which

[6] Jumping ahead, the oracles in pcp systems characterizing $\mathcal{NP}$ have the property of being NP proofs themselves.

[7] The sequence has started with the characterization of $\mathcal{PCP}(\mathtt{poly}, \mathtt{poly})$ as equal non-deterministic exponential-time [5], and continued with its scaled-down in [6, 17] which led to the $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{polylog}, \mathtt{polylog})$ result of [17]. The first PCP-characterization of $\mathcal{NP}$, by which $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, \mathtt{log})$, has appeared in [3] and the cited result was obtained in [2]. This sequence of works, directly related to the stated theorem, was built on and inspired by works from various settings such as interactive proofs [36, 43, 18], program-checking [14, 23, 42], and private computation with oracles [8].

offer a trade-off between the portion of the proof being read and the confidence it offers. Specifically, if the verifier is willing to tolerate an error probability of ϵ then it suffices to let it examine $O(\log(1/\epsilon))$ bits of the (transformed) NP-proof. These bit locations need to be selected at random.

The characterization of $\mathcal{NP}$ in terms of probabilistically checkable proofs plays a central role in recent developments concerning the difficulty of approximation problems (cf., [17], [2] and [37]). To demonstrate this relationship, we first note that Theorem 3 can be rephrased without mentioning the class $\mathcal{PCP}$ altogether. Instead, a new type of polynomial-time reductions, which we call *amplifying*, emerges.

Theorem 4 (Theorem 3 — Rephrased): *There exists a constant $\epsilon > 0$, and a polynomial-time computable function f, mapping the set of 3CNF formulae*[8] *to itself so that*

- *As usual, f maps satisfiable 3CNF formulae to satisfiable 3CNF formulae; and*
- *f maps non-satisfiable 3CNF formulae to* (non-satisfiable) *3CNF formulae for which every truth assignment satisfies at most a $1-\epsilon$ fraction of the clauses.*

The function f is called an amplifying reduction.

proof sketch (Thm. 3 $\Rightarrow$ Thm. 4): Start by considering the pcp for a satisfiable 3CNF (guaranteed by Theorem 3). Use the fact that the pcp system used in the proof of Theorem 3 is non-adaptive[9] (i.e., the queries are determined as a function of the input and the random-tape – and do not depend on answers to previous queries). Next, associate the bits of the oracle with Boolean variables and introduce a (constant size) Boolean formula for each possible outcome of the sequence of $O(\log n)$ coin tosses, describing whether the verifier would have accepted given this outcome. Finally, using auxiliary variables, convert each of these formulae into a 3CNF formula and obtain (as the output of the reduction) the conjunction of all these polynomially many clauses. $\square$

As an immediate corollary one gets results concerning the intractability of approximation. For example,

Corollary 1 : *There exists a constant $\epsilon > 0$, so that the following approximation problem* (known as Max3Sat) *is "NP-hard"* (i.e., cannot be solved in polynomial-time unless $\mathcal{P} = \mathcal{NP}$)

> *Given a satisfiable 3CNF formulae, find a truth assignment which satisfies at least a $1-\epsilon$ fraction of its clauses.*

[8] A 3CNF formula is a Boolean formula consisting of a conjunction of clauses, where each clause is a disjunction of upto 3 literals. (A literal is variable or its negation.).

[9] Actually, this assumption is not essential since one can easily convert an adaptive system into a non-adaptive one, while incurring an exponential blowup in the query complexity (which in our case is a constant).

4.3 The role of randomness

No trade-off between the number of bits examined and the confidence is possible if one requires the verifier to be deterministic. In particular, $\mathcal{PCP}(0, q(\cdot))$ contains only sets that are decidable by a deterministic algorithms of running time $2^{q(n)} \cdot \text{poly}(n)$. It follows that $\mathcal{PCP}(0, \texttt{log}) = \mathcal{P}$. Furthermore, since it is unlikely that all NP-sets can be decided by (deterministic) algorithms of running time, say, $2^n \cdot \text{poly}(n)$, it follows that $\mathcal{PCP}(0, n)$ cannot contain $\mathcal{NP}$.

5 Other Probabilistic Proof Systems

In this section, we shortly review some variants on the basic model of interactive proofs. This variants include models in which the prover is restricted in its choice of strategy, a model in which the prover-verifier interaction is restricted, and a model in which one proves "knowledge" rather than "facts".

5.1 Restricting the prover's strategy

We stress that the restrictions discussed here refer to the strategies employed by the prover both in case it tries to prove valid assertions (i.e., the completeness condition) and in case it tries to fool the verifier to believe false statements (i.e., the soundness condition). Thus, the validity of the verifier decision (concerning false statements) depends on whether this restriction (concerning "cheating" prover strategies) really holds. The reason to consider these restricted models is that they enable to achieve results which are not possible in the general model of interactive proofs (cf., [12, 15, 34, 38]). We consider restrictions of two types: computational or physical.

We start with a physical restriction. In the so-called *multi-prover interactive proof* model, denoted MIP (cf., [12]), the prover is split into several (say, two) entities and the restriction (or assumption) is that these entities cannot interact with each other. Actually, the formulation allows them to coordinate their strategies prior to interacting with the verifier[10] but it is crucial that they don't exchange messages among themselves while interacting with the verifier. The multi-prover model is reminiscent of the common police procedure of isolating collaborating suspects and interrogating each of them separately. On the other hand, the multi-prover model is related to the pcp model [20]. Interestingly, the multi-prover model allows to present (perfect) zero-knowledge proofs for all NP-sets, without relying on any comutational assumptions [12]. Furthermore, these proofs can be made very efficient in terms of communication complexity [16].

We now turn to computational restrictions. Since the effect of this restriction is more noticable in the soundness condition, we refer to these proof systems as being *computationally-sound*. Two variants have been suggested. In *argument* systems [15],

[10] This is implicit in the universal quantifier used in the soundness condition.

the prover stategy is restricted to be probabilistic polynomial-time with auxiliary input (analogously to item (1) in Sec. 2.4). In *CS-proofs* [38], the prover stategy is restricted to be probabilistic and run in time polynomial in the time required to validate the assertion (analogously to item (3) in Sec. 2.4). Interestigly, computationally-sound interactive proofs can be much more communication-efficient than (regular) interactive proofs; cf. [34, 38].

5.2 Non-interactive zero-knowledge proofs

Actualy the term "non-interactive" is somewhat misleading. The model, introduced in [13], consists of three entities: a prover, a verifier and a uniformly selected sequence of bits (which can be thought of as being selected by a trusted third party). Both verifier and prover can read the random sequence, and each can toss additional coins. The interaction consists of a single message sent from the prover to the verifier, who then is left with the decision (whether to accept or not). Based on some reasonable complexity assumptions, one may construct non-interactive zero-knowledge proof systems for every NP-set (cf., [13, 18, 35]).

5.3 Proofs of knowledge

The concept of a proof of knowledge, introduced in [30], is very appealing; yet, its precise formulation is much more complex than one may expect (cf. [9]). Loosely speaking, a knowledge-verifier for a relation R guarantees the existence of a "knowledge extractor" that on input x and access to any interactive machine P^* outputs a y, so that $(x, y) \in R$, within complexity related to the probability that the verifier accepts x when interacting with P^*. By convincing such a knowledge-verifier, on common input x, one proves that he knows a y so that $(x, y) \in R$. It can be shown that the protocol which results by successively applying Construction 2 suffiently many time constitutes a "proof of knowledge" of a 3-coloring of the input graph.

5.4 Knowledge complexity

Zero-knowledge is the lowest level of a knowledge-complexity hierarchy which quantifies the "knowledge revealed in an interaction" [30]. *Knowledge complexity* may be defined as the minimum number of oracle-queries required in order to (efficiently) simulate an interaction with the prover [28]. Preliminary results concerning this measure have appeared in [27].

Acknowledgement

I am grateful to Shafi Goldwasser for suggesting the essential role of randomness as the unifying theme for this exposition. Thanks also for Dana Ron and Uri Zwick for pointing out some errors in earlier versions.

References

[1] W. Aiello and J. Hastad. Perfect Zero-Knowledge Languages can be Recognized in Two Rounds. In *28th FOCS*, pages 439–448, 1987.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. In *33rd FOCS*, pages 14–23, 1992.

[3] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. In *33rd FOCS*, pages 1–13, 1992.

[4] L. Babai. Trading Group Theory for Randomness. In *17th STOC*, pages 421–420, 1985.

[5] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. In *31st FOCS*, pages 16–25, 1990.

[6] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd STOC*, pages 21–31, 1991.

[7] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *JCSS*, Vol. 36, pp. 254–276, 1988.

[8] D. Beaver and J. Feigenbaum. Hiding Instances in Multioracle Queries. In *7th STACS*, Springer Verlag, LNCS Vol. 415, pages 37–48, 1990.

[9] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *Crypto92*.

[10] M. Bellare and S. Goldwasser. The Complexity of Decision versus Search. *SIAM Journal on Computing*, Vol. 23, pages 97–119, 1994.

[11] 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*, Springer Verlag, LNCS Vol. 403, pages 37–56, 1990

[12] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. In *20th STOC*, pages 113–131, 1988.

[13] M. Blum, P. Feldman and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In *20th STOC*, pages 103–112, 1988.

[14] M. Blum, M. Luby and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. In *22nd STOC*, 1990.

[15] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, pages 156–189, 1988. Extended abstract, by Brassard and Crépeau, in *27th FOCS*, 1986.

[16] C. Dwork, U. Feige, J. Kilian, M. Naor and S. Safra, Low Communication 2-Prover Zero-Knowledge Proofs for NP. In *Crypto92*.

[17] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. In *32nd FOCS*, pages 2–12, 1991.

[18] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *31st FOCS*, pages 308–317, 1990.

[19] L. Fortnow, The Complexity of Perfect Zero-Knowledge. In *19th STOC*, pages 204–209, 1987.

[20] L. Fortnow, J. Rompel and M. Sipser. On the Power of Multi-Prover Interactive Protocols. In *Proc. 3rd IEEE Symp. on Structure in Complexity Theory*, pages 156–161, 1988.

[21] M. Furer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, "On Completeness and Soundness in Interactive Proof Systems", *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pp. 429–442, 1989.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[23] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-Testing/Correcting for Polynomials and for Approximate Functions. In *23th STOC*, pages 32–42, 1991.

[24] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38, No. 1, pages 691–729, 1991. Extended abstract in *27th FOCS*, 1986.

[25] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game or a Completeness Theorem for Protocols with Honest Majority. In *19th STOC*, pages 218–229, 1987.

[26] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.

[27] O. Goldreich, R. Ostrovsky and E. Petrank. Knowledge Complexity and Computational Complexity. In *26th STOC*, 1994.

[28] O. Goldreich and E. Petrank. Quantifying Knowledge Complexity. In *32nd FOCS*, pp. 59–68, 1991.

[29] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984. Extended abstract in *14th STOC*, 1982.

[30] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Extended abstract in *17th STOC*, 1985.

[31] S. Goldwasser and M. Sipser. Private Coins versus Public Coins in Interactive Proof Systems. In *18th STOC*, pages 59–68, 1986.

[32] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. Manuscript, 1993. See preliminary versions by Impagliazzo et. al. in *21st STOC* and Håstad in *22nd STOC*.

[33] R. Impagliazzo and M. Yung. Direct Zero-Knowledge Computations. In *Crypto87*, Springer Verlag, LNCS Vol. 293, pages 40–51, 1987.

[34] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *24th STOC*, pages 723–732, 1992.

[35] J. Kilian. On the Complexity of Bounded-Interaction and Noninteractive Zero-Knowledge Proofs. to appear in *35th FOCS*, 1994.

[36] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. In *31st FOCS*, pages 2–10, 1990.

[37] C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems, In *25th STOC*, pages 286–293, 1993.

[38] S. Micali. CS Proofs. to appear in *35th FOCS*, 1994.

[39] M. Naor. Bit Commitment using Pseudorandom Generators. In *Crypto89*, pages 123–132, 1990

[40] R. Ostrovsky and A. Wigderson. One-Way Functions are essential for Non-Trivial Zero-Knowledge, In *Proc. 2nd Israel Symp. on Theory of Computing and Systems* (*ISTCS93*), IEEE Computer Society Press, pages 3–17, 1993.

[41] C. H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *20th STOC*, pages 229–234, 1988.

[42] R. Rubinfeld and M. Sudan. Testing Polynomial Functions Efficiently and over Rational Domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 23–32, 1992.

[43] A. Shamir. IP=PSPACE. In *31st FOCS*, pages 11–15, 1990.

[44] A.C. Yao. Theory and Application of Trapdoor Functions. In *23st FOCS*, pages 80–91, 1982.