Probabilistic Proof Systems – Lecture Notes

Oded Goldreich[*]
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

September 1996

Abstract

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last decade. In these notes, we concentrate on three such proof systems — *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*.

Remark: These are lecture NOTES in the strict sense of the word. Surveys of mine on this subject can be obtained from URL `http://theory.lcs.mit.edu/~oded/pps.html`.

These notes were prepared for a series of lectures given in the Theory Student Seminar of the CS Department of UC-Berkeley.

[*]Visiting Miller Professor, Aug. – Sept. 1996, EECS Dept., UC–Berkeley.

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.

Traditionally, NP is defined as the class of NP-sets. 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. 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 rise 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.

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 actual 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 The Definition

Interaction: Going beyond the uni-directional "interaction" of the NP-proof system. (If the verifier does not toss coins then interaction can be collapsed to a single message.)

(computationally unbounded) Prover: As in NP, we start by not considering the complexity of proving.

(probabilistic polynomial-time) Verifier: We maintain the paradigm that verification ought to be easy, alas we allow random choices (in our notion of easiness).

Completeness and Soundness: We relax the traditional soundness condition by allowing small probability of being fooled by false proofs. The probability is taken over the verifier's random choices. (We still require "perfect completeness"; that is, that correct statements are proven with probability 1). Error probability, being a parameter, can be further reduced by successive repetitions.

Variations: Relaxing the "perfect completeness" requirement yields a two-sided error variant of $\mathcal{IP}$ (i.e., error probability allowed also in the completeness condition). Restricting the verifier to send only "random" (i.e., uniformly chosen) messages yields the restricted Arthur-Merlin interactive proofs (aka public-coins interactive proofs). Alas, both variants are essentially as powerful as the one above.

2.2 An Example: interactive proof of Graph Non-Isomorphism

The problem: (not known to be in $\mathcal{NP}$). Proving that two graphs are isomorphic can be done by presenting an isomorphism, but how do you prove that no such isomorphism exists?

The construction: the "two different object protocol" – if you claim that two objects are different then you should be able to tell which is which (when I present them to you in random order). In the context of the Graph Non-Isomorphism interactive proof, two (supposedly) different objects are defined by taking random isomorphic copies of each of the input graphs. If these graphs are indeed non-isomorphic then the objects are different (the distributions have distinct support) else the objects are identical.

2.3 Interactive proof of Non-Satisfiability

Arithmetization of Boolean (CNF) formulae: Observe that the arithmetic expression is a low degree polynomial. Observe that, in any case, the value of the arithmetic expression is bounded.

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

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

where ϕ is a low degree multi-variant polynomial.

The construction: stripping summations in iterations. In each iteration the prover is supposed to supply the polynomial describing the expression in one (currently stripped) variable. (By the above observation, this is a low degree polynomial and so has a short description.) The verifier checks that the polynomial is of low degree, and that it corresponds to the current value being claimed (i.e., $p(0) + p(1) \equiv v$). Next, the verifier randomly instantiates the variable, yielding a new value to be claimed for the resulting expression (i.e., $v \leftarrow p(r)$, for uniformly chosen $r \in \mathrm{GF}(q)$). The verifier sends the uniformly chosen instantiation to the prover. (At the end of the last iteration, the verifier has a fully specified expression and can easily check it against the claimed value.)

Completeness of the above: When the claim holds, the prover has no problem supplying the correct polynomials, and this will lead the verifier to always accept.

Soundness of the above: It suffices to bound the probability that for a particular iteration the initial claim is false whereas the ending claim is correct. Both claims refer to the current summation expression being equal to the current value, where 'current' means either at the beginning of the iteration or at its end. Let $T(\cdot)$ be the actual polynomial representing the expression when stripping the current variable, and let $p(\cdot)$ be any potential answer by the prover. We may assume that $p(0) + p(1) \equiv v$ and that p is of low-degree (as otherwise the verifier will reject). Using our hypothesis (that the initial claim is false), we know that $T(0) + T(1) \not\equiv v$. Thus, p and T are different low-degree polynomials and so they may agree on very few points. In case the verifier instantiation does not happen to be one of these few points, the ending claim is false too.

Open Problem 1 (alternative proof of co$\mathcal{NP} \subseteq \mathcal{IP}$): *Polynomials play a fundamental role in the above construction and this trend has even deepened in subsequent works on PCP. It does not seem possible to abstract that role, which seems to be very annoying. I consider it important to obtain an alternative proof of* co$\mathcal{NP} \subseteq \mathcal{IP}$*; a proof in which all the underlying ideas can be presented at an abstract level.*

2.4 The Power of Interactive Proofs

$$\boxed{\mathcal{IP} = \mathcal{PSPACE}}$$

Interactive Proofs for PSPACE: Recall that PSPACE languages can be expressed by Quantified Boolean Formulae. The number of quantifiers is polynomial in the input, but there are both existential and universal quantifiers, and furthermore these quantifiers may alternate. Considering the arithmetization of these formulae, we face two problems: Firstly, the value of the formulae is only bounded by a double exponential function (in the length of the input), and secondly when stripping out summations, the expression may be a polynomial of high degree (due to the universal quantifiers which are replaced by products). The first problem is easy to deal with by using the Chinese Reminder Theorem (i.e., if two integers in $[0, M]$ are different then they must be different modulo most of the primes up-to poly$(\log M)$). The second problem is resolved by "refreshing" variables after each universal quantifier (e.g, $\exists x \forall y \exists z \phi(x, y, z)$ is transformed into $\exists x \forall y \exists x'(x = x') \wedge \exists y \phi(x', y, z)$).

IP in PSPACE: We show that for every interactive proof there exists an optimal prover strategy, and furthermore that this strategy can be computed in polynomial-space. This follows by looking at the tree of all possible executions.

The IP Hierarchy: Let $\mathcal{IP}(r(\cdot))$ denote the class of languages having an interactive proof in which at most $r()$ messages are exchanges. Then, $\mathcal{IP}(0) = \text{co}\mathcal{RP} \subseteq \mathcal{BPP}$. The class $\mathcal{IP}(1)$ is a randomized version of $\mathcal{NP}$; witnesses are verified via a probabilistic polynomial-time pprocedure, rather than a deterministic one. The class $\mathcal{IP}(2)$ seems fundamentally different; the verification procedure here is truly interactive. Still, this class seems close to $\mathcal{NP}$; specifically, it is contained in the polynomial-time hierarchy (which seems 'low' when contrasted with $\mathcal{PSPACE} = \mathcal{IP}(\text{poly})$). Interestingly, $\mathcal{IP}(2r(\cdot)) = \mathcal{IP}(r(\cdot))$, and so in particular $\mathcal{IP}(O(1)) = \mathcal{IP}(2)$. (Note that "$\mathcal{IP}(2r(\cdot)) = \mathcal{IP}(r(\cdot))$" can be applied successively a constant number of times, but not more.)

Open Problem 2 (the structure of the $\mathcal{IP}(\cdot)$ hierarchy):

Suppose that $L \in \mathcal{IP}(r)$. What can be said about $\overline{L}$?

Currently, we only know to argue as follows: $\mathcal{IP}(r) \subseteq \mathcal{IP}(\text{poly}) \subseteq \mathcal{PSPACE}$ and so $\overline{L} \in \mathcal{PSPACE}$ and is in $\mathcal{IP}(\text{poly})$. This seems ridiculous: we do not use the extra information on $\mathcal{IP}(r)$. On the other hand, we don't expect $\overline{L}$ to be in $\mathcal{IP}(g(r))$, for any function g, since this will put $\text{co}\mathcal{NP} \subseteq \text{co}\mathcal{IP}(1)$ in $\mathcal{IP}(2)$. So another parameter may be relevant here; how about the lengths of the messages exchanged in the interaction. Indeed, if L has an interactive proof in which the total message length is m then $\overline{L}$ has an interactive proof in which the total message length is $O(m^3)$. (This just follows by the known $\mathcal{PSPACE} \subseteq \mathcal{IP}$ construction.) I consider it important to obtain a better result! In general, it would be interesting to get a better understanding of the $\mathcal{IP}(\cdot)$ Hierarchy.

2.5 How Powerful Should the Prover be?

The Cryptographic Angle: Interactive proofs occur inside "cryptographic" protocols and so the prover is merely a probabilistic polynomial-time machine; yet it may have access to an auxiliary input (given to it or generated by it in the past). Such provers are relatively weak (i.e., they can only prove languages in $\mathcal{IP}(1)$); yet, they may be of interest for other reasons (e.g., see zero-knowledge).

The Complexity Theoretic Angle: It make sense to try to relate the complexity of proving a statement (to another party) to the complexity of deciding whether the statement holds. This gives rise to two related approaches:

1. The prover is a probabilistic polynomial-time oracle machine with access to the language. This approach can be thought of as extending the notion of self-reducibility (of NP-languages): These languages have an NP-proof system in which the prover is a polynomial-time machine with oracle access to the language. Indeed, alike NP-complete languages, the IP-complete languages also have such a "relatively efficient" prover. (Recall that an optimal prover strategy can be implemented in polynomial-space, and thus by a polynomial-time machine having oracle access to a PSPACE-complete language.)

2. The prover runs in time which is polynomial in the complexity of the languages.

Open Problem 3 *Further investigate the power of the various notions, and in particular the one extending self-reducibility on NP languages. Better understanding of the latter is also long due. A specific challenge: provide an NP-proof system for Quadratic Non-Residucity (QNR), using a probabilistic polynomial-time prover with access to QNR language.*

2.6 Computationally-Sound Proofs

Such proofs systems are fundamentally different from the above discussion (which did not effect the soundness of the proof systems): Here we consider relations of the soundness conditions – false proofs may exist (even with high probability) but are hard to find. Variants may correspond to the above approaches; specifically, the following has been investigated:

Argument Systems: One only considers prover strategies implementable by (possibly non-uniform) polynomial-size circuits (eq., probabilistic polynomial-time machines with auxiliary inputs). Under some reasonable assumptions there exist argument systems for $\mathcal{NP}$ having poly-logarithmic communication complexity. Analogous interactive proofs cannot exists unless $\mathcal{NP}$ is contained in Quasi-Polynomial Time (i.e., $\mathcal{NP} \subseteq \mathrm{Dtime}(\exp(\mathrm{poly}(\log n)))$).

CS Proofs: One only considers prover strategies implementable in time polynomial in the complexity of the language. In an non-interactive version one asks for "certificates a la NP-type" which are only computationally sound. In a model allowing both prover and verifier access to a random oracle, one can convert interactive proofs (alike CS proofs) into non-interactive ones. As a heuristics, it is also suggested to replace the random oracle by use of "random public functions" (a fuzzy notion, not to be confused with pseudorandom functions).

Open Problem 4 *Try to provide firm grounds for the heuristics of making proof systems non-interactive by use of "random public functions": I advise not to try to define the latter notion (in a general form), but rather devise some ad-hoc method, using some specific but widely believed complexity assumptions (e.g., hardness of deciding Quadratic Residucity modulo a composite number), for this specific application.*

3 Zero-Knowledge Proofs

Zero-knowledge proofs are central to cryptography. Furthermore, zero-knowledge proofs are very intriguing 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.

The zero-knowledge paradigm: Whatever can be efficiently computed after interacting with the prover on some common input, can be efficiently computed from this input alone (without interacting with anyone). That is, the interaction with the prover can be efficiently *simulated* in solitude.

A Technical Note: I have deviated from other presentation in which the simulator works in average (probabilistic) polynomial-time and require that it works in strict probabilistic polynomial-time. Yet, I allow the simulator to halt without output with probability at most $\frac{1}{2}$. Clearly this implies an average polynomial-time simulator, but the converse is not known. In particular, some known positive results regarding perfect zero-knowledge (with average polynomial-time simulators) are not known to hold under the above more strict notion.

3.1 Perfect Zero-Knowledge

The Definition: A simulator can produce *exactly* the same distribution as occurring in an interaction with the prover. Furthermore, in the general definition this is required with respect to any probabilistic polynomial-time verifier strategy (not necessarily the one specified for the verifier). Thus, the zero-knowledge property protects the prover from any attempt to obtain anything from it (beyond conviction in the validity of the assertion).

Zero-Knowledge NP-proofs: Extending the NP-framework to interactive proof is essential for the non-triviality of zero-knowledge. It is easy to see that zero-knowledge NP-proofs exist only for languages in $\mathcal{RP}$. (Actually, that's a good exercise.)

A perfect zero-knowledge proof for Graph Isomorphism: The prover sends the verifier a random isomorphic copy of the first input graph. The verifier challenges the prover by asking the prover to present an isomorphism (of graph sent) to either the first input graph or to the second input graph. The verifier's choice is made at random.
The fact that this interactive proof system is zero-knowledge is more subtle than it seems; for example, (many) parallel repetitions of the proof system are unlikely to be zero-knowledge.

3.2 (Computational) Zero-Knowledge

This definition is obtained by substituting the requirement that the simulation is identical to the real interaction, by the requirement that the two are *computational indistinguishable.*

Computational Indistinguishability is a fundamental concept of independent interest. Two ensembles are considered indistinguishable by an algorithm A if A's behavior is almost invariant of whether its input is taken from the first ensemble or from the second one. We interpret "behavior" as a binary verdict and require that the probability that A outputs 1 in both cases is the same up-to a negligible difference (i.e., smaller than $1/p(n)$, for any positive polynomial $p(\cdot)$ and all sufficiently long input lengths (denoted by n)). Two ensembles are computational indistinguishable if they are indistinguishable by all probabilistic polynomial-time algorithms.

A zero-knowledge proof for NP – abstract (boxes) setting: It suffices to construct such a proof system for 3-Colorability (3COL). (To obtain a proof system for other NP-languages use the fact that the (standard) reduction of $\mathcal{NP}$ to 3COL is polynomial-time invertible.)
The prover uses a fixed 3-coloring of the input graph and proceeds as follows. First, it uniformly selects a relabeling of the colors (i.e., one of the 6 possible ones) and puts the resulting color of each vertex in a locked box (marked with the vertex name). All boxes are sent to the verifier who response with a uniformly chosen edge, asking to open the boxes corresponding to the endpoint of this edge. The prover sends over the corresponding keys, and the verifier opens the two boxes and accepts iff he sees two different legal colors.

A zero-knowledge proof for NP – real setting: The locked boxes need to be implemented digitally. This is done by a *commitment scheme*, a cryptographic primitive designed to implement such locked boxes. Loosely speaking, a commitment scheme is a two-party protocol which proceeds in two phases so that at the end of the first phase (called the commit phase) the first party (called sender) is committed to a single value (which is the only value he can later reveal in the second phase), whereas at this point the other party gains no knowledge on the committed value. Commitment schemes exist if (and actually iff) one-way functions exist. Thus, the mildest of all cryptographic assumptions suffices for constructing zero-knowledge proofs for $\mathcal{NP}$ (and actually for all of $\mathcal{IP}$). Furthermore, zero-knowledge proofs for languages which are "hard on the average" imply the existence of one-way functions; thus, the above construction essentially utilizes the minimal possible assumption.

one-way functions imply

$$\mathcal{IP} = \mathcal{ZKIP}$$

3.3 Concluding Remarks

The prover's strategy in the above zero-knowledge proof for NP can be implemented by a probabilistic polynomial-time machine which is given (as auxiliary input) an NP-witness for the input. (This is clear for 3COL, and for other NP-languages one needs to use the fact that the relevant reductions are coupled with efficient witness transformations.) The efficient implementation of the prover strategy is essential to the applications below.

Applications to Cryptography: Zero-knowledge proofs are a powerful tool for the design of cryptographic protocols, in which one typically wants to guarantee proper behavior of a party without asking him to reveal all his secrets. Note that proper behavior is typically a polynomial-time computation based on the party's secrets as well as on some known data. Thus, the claim that the party behaves consistently with its secrets and the known data can be casted as an NP-statement, and the above result can be utilized. More generally, using additional ideas, one can provide a secure protocol for any functional behavior. These general results have to be considered as plausibility arguments; you would not like to apply these general constructions to specific practical problems, yet you should know that these specific problems are solvable.

Open Problems do exists, but seem more specialized in nature. For example, it would be interesting to figure out and utilize the minimal possible assumption required for constructing "zero-knowledge protocols for NP" in various models like constant-round interactive proofs, the "non-interactive" model, and perfect zero-knowledge arguments.

Further Reading: see chapter on Zero-Knowledge in my "fragments of a book" on *Foundations of Cryptography* (available from URL `http://theory.lcs.mit.edu/~oded/frag.html`).

Solution to Exercise regarding Zero-Knowledge NP-proofs: An NP-proof system for a language L yields an NP-relation for L (defined using the verifier). On input $x \in L$ a perfect zero-knowledge simulator either halts without output or outputs an accepting conversation (i.e., an NP-witness for x).

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 probe 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 The Definition

The Basic Model: A probabilistically checkable proof system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a 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. Completeness and soundness are defined similarly to the way they were defined for interactive proofs: for valid assertions there exist proofs making the verifier always accepts, whereas no oracle can make the verifier accept false assertions with probability above $\frac{1}{2}$. (We've specified the error probability since we intend to be very precise regarding some complexity measures.)

Additional complexity measures of fundamental importance are the *randomness* and *query* complexities. Specifically, $\mathcal{PCP}(r(\cdot), q(\cdot))$ denotes the set of languages 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. (As usual in complexity theory, unless stated otherwise, the oracle answers are always binary (i.e., either 0 or 1).)

Observed that the "effective" oracle length is at most $2^r \cdot q$ (i.e., locations which may be accessed on some random choices). In particular, the effective length of oracles in a $\mathcal{PCP}(\log, \cdot)$ system is polynomial.

PCP <u>augments</u> the traditional notion of a proof: An oracle which always makes the pcp-verifier accept constitutes a proof in the standard mathematical sense. However a pcp system 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

The PCP Characterization Theorem states that

$$\boxed{\mathcal{PCP}(\log, O(1)) = \mathcal{NP}}$$

Thus, probabilistically checkable proofs in which the verifier tosses only logarithmically many coins and makes only a constant number of queries exist for every NP-language. It follows that NP-proofs can be transformed into NP-proofs which 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. Furthermore, an original NP-proof can be transformed into an NP-proof allowing such trade-off in polynomial-time. (The latter is an artifact of the proof of the PCP Theorem.)

The Proof of the PCP Characterization Theorem is one of the most complicated proofs in the Theory of Computation. Its main ingredients are:

1. A $\mathsf{pcp}(\log, \mathrm{poly}(\log))$ proof system for $\mathcal{NP}$. Furthermore, this proof system has additional properties which enable proof composition as in item (3) below.
2. A $\mathsf{pcp}(\mathrm{poly}, O(1))$ proof system for $\mathcal{NP}$. This proof system also has additional properties enabling proof composition as in item (3).
3. The $\mathsf{proof\ composition}$ paradigm: Suppose you have a $\mathsf{pcp}(r(\cdot), O(\ell(\cdot)))$ system for $\mathcal{NP}$ in which a constant number of queries are made (non-adaptively) to an 2^{ℓ}-valued oracle and the verifier's decision regarding the answers may be implemented by a $\mathrm{poly}(\ell)$-size circuit. Further suppose that you have a $\mathsf{pcp}(r'(\cdot), q(\cdot))$-like system for $\mathcal{P}$ in which the input is given in encoded form via an additional oracle so that the system accepts input-oracles which encode inputs in the language and reject any input-oracle which is "far" from the encoding of any input in the language. In this latter system access to the input-oracle is accounted in the query complexity. Furthermore, suppose that the latter system may handle inputs which result from concatenation of a constant number of sub-inputs each encoded in a separate sub-input oracle. Then, $\mathcal{NP}$ has a $\mathsf{pcp}(2(r(\cdot) + r'(s(\cdot))), 2q(s(\cdot)))$, where $s(n) \stackrel{\mathrm{def}}{=} \mathrm{poly}(\ell(n))$. [The extra factor of 2 is an artifact of the need to amplify each of the two pcp systems so that the total error probability sums up to at most 1/2.]

In particular, the proof system of item (1) is composed with itself [using $r = r' = \log$, $\ell = q = \mathrm{poly}(\log)$, and $s(n) = \mathrm{poly}(\log(n))$] yielding a $\mathsf{pcp}(\log, \mathrm{poly}(\log\log))$ system for $\mathcal{NP}$, which is then composed with the system of item (2) [using $r = \log$, $\ell = \mathrm{poly}(\log\log)$, $r' = \mathrm{poly}$, $q = O(1)$, and $s(n) = \mathrm{poly}(\log\log(n))$] yielding the desired $\mathsf{pcp}(\log, O(1))$ system for $\mathcal{NP}$.

The $\mathsf{pcp}(\log, \mathrm{poly}(\log))$ system for $\mathcal{NP}$: We start with a different arithmetization of CNF formulae (than the one used for constructing an interactive proof for co$\mathcal{NP}$). Logarithmically many variables are used to represent (in binary) the names of variables and clauses in the input formula, and an oracle from variables to Boolean values is supposed to represent a satisfying assignment. An arithmetic expression involving a logarithmic number of summations is used to represent the value of the formula under the truth assignment represented by the oracle. This expression is a low-degree polynomial in the new variables and has a cubic dependency on the assignment-oracle. Small-biased probability spaces are used to generate a polynomial number of such expressions so that if the formula is satisfiable then all these expressions evaluate to zero, and otherwise at most half of them evaluate to zero. Using a summation test (as in the interactive proof for co$\mathcal{NP}$) and a low-degree test, this yields a $\mathsf{pcp}(t(\cdot), t(\cdot))$ system for $\mathcal{NP}$, where $t(n) \stackrel{\mathrm{def}}{=} O(\log(n) \cdot \log\log(n))$. [We use a finite field of $\mathrm{poly}(\log(n))$ elements, and so we need $(\log n) \cdot O(\log\log n)$ random bits for the summation test.] To obtain the desired pcp system, one uses $\frac{O(\log n)}{\log\log n}$-long sequences over $\{1, ..., \log n\}$ to represent variable/clause names (rather than logarithmically-long binary sequences). [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 test.] All this is relatively easy compared to what is needed in order to transform the pcp system so that only a constant number of queries are made to a (multi-valued) oracle. This is obtained via (randomness-efficient) "parallelization" of pcp systems, which in turn depends heavily on efficient low-degree tests.

Open Problem 5 *As a first step towards the simplification of the proof of the PCP Characterization, one may want to provide an alternative "parallelization" procedure which does not rely on polynomials or any other algebraic creatures. A first step towards this partial goal was taken by Safra and myself* (see TR96-047 of `http://www.eccc.uni-trier.de/eccc`)*: We have constructed an efficient low-degree test which utilizes a simple/inefficient low-degree test which is parallelized using a new "combinatorial consistency lemma".*

The $\mathsf{pcp}(\text{poly}, O(1))$ **system for** $\mathcal{NP}$**:** It suffices to prove the satisfiability of a systems of quadratic equations over GF(2) (as this problem in NPC). The oracle is supposed to hold the values of all quadratic expressions under a satisfying assignment to the variables. We distinguish two tables in the oracle: One corresponding to the (2^n) linear expressions and the other to the (2^{n^2} pure) bilinear expressions. Each table is tested for self-consistency (via a linearity test) and the two tables are tested to be consistent (via a matrix-equality test which utilizes "self-correction"). Each of these tests utilizes a constant number of Boolean queries, and randomness which is logarithmic in the size of the corresponding table.

4.3 PCP and Approximation

PCP-Characterizations of $\mathcal{NP}$ plays a central role in recent developments concerning the difficulty of approximation problems. To demonstrate this relationship, we first note that the PCP Characterization Theorem can be rephrased without mentioning the class $\mathcal{PCP}$ altogether. Instead, a new type of polynomial-time reductions, which we call *amplifying*, emerges.

Amplifying reductions: There exists a constant $\epsilon > 0$, and a polynomial-time reduction f, of 3SAT to itself so that f maps non-satisfiable 3CNF formulae to 3CNF formulae for which every truth assignment satisfies at most a $1 - \epsilon$ fraction of the clauses. I call the reduction f *amplifying*. Its existence follows from the PCP Characterization Theorem by considering the guaranteed pcp system for 3SAT, associating the bits of the oracle with Boolean variables and introducing 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).

Amplifying reductions and Non-Approximability: The above amplifying reduction of 3SAT implies that it is NP-Hard to distinguish satisfiable 3CNF formulae from 3CNF formulae for which every truth assignment satisfies less than a $1-\epsilon$ fraction of its clauses. Thus, Max-3SAT is NP-Hard to approximate to within a $1 - \epsilon$ factor.

Stronger Non-Approximability Results were obtained via alternative PCP Characterizations of NP. For example, the NP-Hardness of approximating Max-Clique to within $N^{1-\epsilon}$, $\forall\epsilon > 0$, was obtained via $\mathcal{NP} = \overline{\mathcal{FPCP}}(\log, \epsilon)$, where the second parameter in $\overline{\mathcal{FPCP}}$ measures the "amortized free-bit" complexity of the pcp system.

Open Problems regarding various parameters in PCP Characterizations of NP will probably remain also after the turbulence currently created by works in progress by Hastad and by Raz & Safra.