T-79.511 Special Course on Cryptology / Zero Knowledge: A first look at the $\mathbf{PCP}(\log n, 1) = \mathbf{NP}$ theorem

Mika Kojo

October 19, 2001

1 Introduction

In 1970's the work by Cook and Levin on complexity of computation introduced the notion of the complexity class **NP**. Intuitively, a language $L \subset \{0,1\}^*$, for some universe $\{0,1\}^*$, is in **NP** iff there exists a polynomial time verifier $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ (a Turing machine) such that for any $x \in \{0,1\}^*$ and some positive integer $c \in \mathbb{Z}_+$;

$$x \in L \text{ iff } \exists y \in \{0,1\}^*, |y| = |x|^c, \text{ and } V(x,y) = 1.$$

In particular, in this survey we are interested in the "proof" string $y \in \{0,1\}^*$.

The interesting development throughout the 1980's to 1992 (say) was all about finding different ways to characterize languages in **NP**. To motivate the topic of this survey let us see what Cook-Levin proof of **NP** completeness for $\mathbf{3-SAT}$ implies to a certain approximation problem.

Let us recall the definition of $\mathbf{MAX-3SAT}$. Let $\phi \in \mathbf{3-CNF}$, now $\mathbf{MAX-3SAT}$ is the problem of finding x such that it maximizes the number of clauses satisfied in ϕ. We denote by $\mathbf{MAX-3SAT}(\phi)$ the proportion of clauses satisfied in ϕ by any assignment. It is a standard result that $\mathbf{MAX-3SAT}$ is **NP**-hard.

Now consider an approximation variant of $\mathbf{MAX-3SAT}$. Let $\mathbf{MAX-3SAT}(\epsilon)$ denote the decision problem; let $\phi \in \mathbf{3-CNF}$ and $\mathbf{MAX-3SAT}(\phi) = 1$ or $\mathbf{MAX-3SAT}(\phi) < 1-\epsilon$ then output 1 if ϕ is satisfiable or 0 otherwise. Now one asks whether $\mathbf{MAX-3SAT}(\epsilon)$ is **NP** hard.

The Cook and Levin idea for the **NP**-completeness of $\mathbf{3-SAT}$ was to find a generic way to present proofs. Namely, they define a tableau that represents the computation of the verifier that accepts. This is an elegant approach as long as we can check the proof in polynomial time and space. Indeed, with Turing machines this is simple. More precisely, the proof consists of an ordered list of workspaces of the Turing machine (or tapes) such that the first represents the pair (x,y), where x is the input and y is the "proof", then the last element is the state which accepts. Now the whole list takes only polynomial amount of space and due to local nature of Turing machine computation it may be checked in polynomial time. Indeed, it suffices to check that certain local "windows" behave correctly under the finite control of the Turing machine.

This construction can be easily written as a $\mathbf{3-SAT}$ expression (by utilizing the local nature of the checking) and hence it is seen that $\mathbf{3-SAT}$ is **NP**-complete. However, one may ask what does this tableau construction imply about $\mathbf{MAX-3SAT}(\epsilon)$. Let $\phi \in \mathbf{3-CNF}$ and generate rejecting, say, tableau proof for $\mathbf{3-SAT}$ for it. Take some other $\mathbf{3-SAT}$ tableau proof and combine these two trivially at some point. Namely, cut the tail of the first proof off and the beginning of the second proof and glue them together. Now you end up with a invalid proof that looks locally as a tableau which accepts the ϕ for $\mathbf{3-SAT}$. The problem is that the tableau may be mapped into a $\mathbf{3-SAT}$ formula which is approximable within any constant $\epsilon < 1$. Hence, this tableau approach is not capable of saying anything about the hardness of $\mathbf{MAX-3SAT}(\epsilon)$.

Above we saw that the problem is explicitly the local nature of the tableau proof. We would like to have a proof format that is locally checkable such as the tableau, but more robust in the sense that cut-and-paste proofs would not look like proofs any longer. Indeed, we may guess that such a format might allow prof of **NP**-hardness results in context of approximation questions.

We start with definitions (along the lines of [1]). Suppose we give each verifier a third argument, a random string $\tau \in \{0,1\}^*$ of finite length.

Definition 1.1 A verifier $V : \{0,1\}^{*^3} \to \{0,1\}$ is $(r(n), q(n))$-restricted if on each input of size n it uses at most $O(r(n))$ random bits for its computation and queries at most $O(q(n))$ bits of the proof.

Definition 1.2 A verifier $V : \{0,1\}^{*^3} \to \{0,1\}$ can probabilistically check proofs for language $L \subset \{0,1\}^*$ if;

(i) For every input $x \in L$ there exists a proof $y \in \{0,1\}^*$ s.t. $V(x,y,\tau) = 1$ for every random string τ;

(ii) For any input $x \notin L$, every proof $y \in \{0,1\}^*$ is rejected with probability at least $c < 1$, i.e.

$$\Pr_{\tau}[V(x,y,\tau) = 1] < c.$$

To make the definitions more intuitive it is better to think y as a promise of the proof (or an oracle) rather than the actual proof. The idea of this survey is to explain that the verifier V does not need to read all of the proof, but just query it at suitable points. Thus the proof is more like a deterministic function or a string having random access properties.

Definition 1.3 A language L is in $\mathbf{PCP}(r(n), q(n))$ if there is an $(r(n), q(n))$-restricted verifier V that can check proof for L.

It is immediate that $\mathbf{NP} = \mathbf{PCP}(0, \mathbf{poly}(n))$, for this is exactly the set of languages for which the proof can be checked in deterministic polynomial time (i.e. no randomness is used). However, perhaps remarkably a much more intriguing connection can be proven.

Theorem 1.4 (PCP theorem) $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

It is unfortunately not possible to give a full proof of this fact in this survey, but we will go through several essential ingredients. The references [1] (which this survey is mainly based on), [2], [7], and [4] give more or complementary details. Indeed, let us consider briefly what the **PCP** theorem actually says.

Trivially, any language in L in **NP** has a proof which can be checked by using $O(\log n)$ random bits and $O(1)$ "spot checks" or queries of the proof bits. This is remarkable in its own right, but perhaps more so when we elaborate the following result.

Theorem 1.5 $\mathbf{MAX-3SAT}(\epsilon)$ *is* **NP**-*hard for some* $\epsilon > 0$ *if and only if* $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

Proof.[1] Let us first prove that **NP**-hardness of $\mathbf{MAX-3SAT}(\epsilon)$ implies $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

Let L be an **NP** language and $x \in \{0,1\}^*$. Since $\mathbf{MAX-3SAT}(\epsilon)$ is **NP**-hard, we can reduce L to $\mathbf{MAX-3SAT}(\epsilon)$ in polynomial time. Now we show $L \in \mathbf{PCP}(\log n, 1)$. Let x be any input. The verifier first reduces x to an instance ϕ of $\mathbf{MAX-3SAT}(\epsilon)$ that is satisfiable iff $x \in L$. As a proof for x, the verifier expects a satisfying assignment for ϕ. Given such a proof it does a probabilistic check on it by randomly choosing a clause in ϕ and querying the proof for the values of the three variables involved in that clause. It accepts iff the clause is satisfied. Clearly, if $x \in L$, then a satisfying assignment for ϕ will cause the verifier to accept with probability 1. On the otherhand, if $x \notin L$ then every assignment satisfies less than a $1-\epsilon$ fraction of the clauses and so the verifier accepts with probability less than $1-\epsilon$. By repeating the check, say, $O(1/\epsilon)$ times, one gets a constant acceptance probability $1/2$.

We need to prove the other direction, that is, $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$ implies $\mathbf{MAX-3SAT}(\epsilon)$ is **NP**-hard (for some ϵ).

It is thus clear that $\mathbf{SAT} \in \mathbf{PCP}(\log n, 1)$. Let $\psi \in \mathbf{SAT}$. Given a random string r let $i_1(\psi, r), \ldots, i_c(\psi, r)$ denote the computation of the positions of the proof string queried by the verifier. For each r the computation of the verifier is determined by the value of the proof string at these positions. Now, we can encode the outcome of the computation by a constant-size $\mathbf{3-CNF}$ formula $\phi_{\psi,r}$. Namely, $\phi_{\psi,r}$ is true iff $V(\psi, y, r) = 1$. Let ϕ_ψ

[1] This proof follows almost verbatim [2].

be the concatenation of the $\phi_{\psi,r}$ for all r. Clearly, if ψ is satisfiable then there is a proof y such that ϕ_ψ is satisfiable. Conversely, if ψ is not satisfiable then for every proof y at least one clause in half of the $\phi_{\psi,r}$ is not satisfied. Thus a constant fraction of the clauses in ϕ_ψ are not satisfied. Since the formula ϕ_ψ can be constructed in polynomial time, this shows that **SAT** reduces to $\mathbf{MAX-3SAT}(\epsilon)$ for some $\epsilon > 0$. ∎

Referring to our previous consideration about $\mathbf{MAX-3SAT}$, by the Cook-Leving tableau construction, it now clear that **PCP** is what we were looking for. Namely, **PCP** theorem implies that robust proofs exist (and indeed, that they can be constructed).

The above theorem gives a very interesting result in the sense that we now know that for some $\epsilon > 0$ it is **NP**-hard to even approximate $\mathbf{MAX-3SAT}$. Naturally, cryptographers find these results most intriguing in the sense that they may have implications to public key cryptography[2], for example. However, even if no such connection would arise the way **PCP** theorem allows one to create robust proofs is an important complexity theoretical notion.

We will take upon parts of the proof of theorem 1.4 in the following sections. However, due to limitations in space and time we only sketch the essentials. See the already mentioned papers for further details. It is also useful to mention that the constant $O(1)$ in the **PCP** theorem can be made amazingly small. This again can be further studied in the mentioned references, and in their references.

We note, before stepping further, the following easy fact.

Theorem 1.6 $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$.

Proof. This is easy and is given explicitly in [1]. The proof of theorem 1.5 uses all the necessary ideas. Further details are omitted. ∎

ACKNOWLEDGMENTS. Thanks are due to Helger Lipmaa for suggesting this topic in the first place and for giving some references.

[2]It may be appropriate to mention here that the **PCP** theorem has implications to point lattice problems, such as finding approximate shortest vectors. However, Ajtai's more recent results go beyond what **PCP** based proofs have achieved.

2 Composition lemma

We need to prove that $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$. The main tool for the task is the composition lemma, mainly as an organizing tool for the proof. We will prove it here, and consider in subsequent sections how the rest of the proof may be dealt with. Unfortunately, the details are too numerous to be given explicitly.

Definition 2.1 The verifier V acts in three stages. First, it reads the random string, and decides which locations it examines from the proof. In the second stage the verifier reads symbols (not necessarily bits) from the proof string onto its local workspace. The third stage is the computation which decides whether or not to accept. The *decision time* is the time taken by the third stage.

Following Arora [1] we work with $\mathbf{3-SAT}$ also, although it may be that graph 3-coloring, for example, gives nicer exposition (see [4], [8]).

Definition 2.2 A verifier V for $\mathbf{3-SAT}$ is in *normal form* if it satisfies the following properties.

(i) The verifier has a certain alphabet A, which is not necessary binary;

(ii) The verifier can check assignments that are split into many parts. Namely, the verifier has a subroutine that can check proof of special form very efficiently. Let p be a positive integer. We define the behaviour of the subroutine in a bit more detail.

   (a) The subroutine defines a code C over the alphabet A, with minimal distance $1 > \delta > c$. There is a natural inclusion map $\sigma : \{0,1\}^{n/p} \to C$;

   (b) The subroutine expects the proof string to be of the form
   $$\sigma(a_1)|\sigma(a_2)|\dots|\sigma(a_p)|\pi,$$
   where π is a string that helps in showing that $a_1|\dots|a_p$ is a satisfying statement. It is important that π is used only to make the proof verification more efficient, it will not affect the correctness of the result (except in the case that it is misleading and thus verifier will reject).

Let $z_1|\dots|z_p|\pi$ be the proof. Suppose that $\sigma^{-1}(z_1)|\dots|\sigma^{-1}(z_p)$ is a satisfying statement, then there exists π such that V accepts with probability 1. If there exists i such that z_i is not uniquely decodable (i.e. not $\delta/3$, say, close to a codeword) then for all π the subroutine accepts with probability less than $1/2$. Finally, if all z_i's are codewords, but $\sigma^{-1}(N(z_1))|\dots|\sigma^{-1}(N(z_p))$ is not a satisfying assignment, where $N(z_i)$ is the nearest codeword to z_i, then for all π the probability that the subroutine accepts is less than $1/2$.

In the definition above we used some coding theory terminology, however, in this survey very little is needed. Namely, we call a code $C \subset A^n$ such that the natural (Hamming) metric gives large distances for two elements $x, y \in C$. The (relative) minimal distance δ indicates the "goodness" of the code in the sense that we can decode message $z \in A^n$ uniquely if there exists only one codeword $x \in C$ such that x and z have distance less than $\delta/2$. It is natural to say that z in such a case is $\delta/2$-close (to an unique codeword).

Definition 2.3 A verifier V in normal form is $(r(n), q(n), t(n))$-restricted if on input of size n it uses $O(r(n))$ random bits, reads $O(q(n))$ symbols from the proof string and has decision time $\mathbf{poly}(t(n))$. While checking assignments split into p parts it reads $O(p \cdot q(n))$ symbols.

It is important to remark that although we deal with *symbols*, instead of bits with normal form verifiers this causes no real problems. For we have $O(q(n)\log|A|)$ for the number of bits we read given the alphabet A. Furthermore, $O(q(n)\log|A|) \leq \mathbf{poly}(t(n))$.

Lemma 2.4 (Composition lemma) *Let V_1 and V_2 be normal-form verifiers for $\mathbf{3-SAT}$ that are $(R(n), Q(n), T(n))$-restricted and $(r(n), q(n), t(n))$-restricted respectively. Then tehre is normal form $\mathbf{3-SAT}$ verifier that is $(R(n) + r'(n), Q(n) \cdot q'(n), t'(n))$-restricted, where $r'(n) = r(\mathbf{poly}(T(n)))$, $q'(n) = q(\mathbf{poly}(T(n)))$, and $t'(n) = t(\mathbf{poly}(T(n)))$.*

A particular special case worth mention here is the case when $Q(n)$ and $q(n)$ are both $O(1)$. This implies that composition verifier is $(R(n) + r(\mathbf{poly}(T(n))), 1, t(\mathbf{poly}(T(n))))$-restricted. Furthermore, suppose that t is a function like log, growing very slowly, then $\log(\mathbf{poly}(T(n)) = O(\log(T(n))$, giving an exponential improvement over $T(n)$. Indeed, this illustrates the power of composition as applied in the **PCP** theorem.

Proof. (Proof sketch of lemma 2.4, see [1] for further details.)

Let us fix the size of the input n. We drop n from the functions when appropriate. The proof is an explicit construction of the verifier V_3 that is combination of the verifier V_1 and V_2. Indeed, we give the construction first and then explain it in some detail.

Suppose y is the proof accepted by V_1. Let $\gamma \in \{0,1\}^R$ be the random string for V_1, and let us fix it. Now clearly we have also fixed the locations $i_1, \dots, i_Q$ that V_1 will read off y. Let us write $y(i_1), \dots, y(i_Q)$ for these read symbols. We can now build $\phi_\gamma \in \mathbf{3-SAT}$ such that V_1 accepts iff $\exists \pi_\gamma$ s.t.

$$y(i_1)|\dots|y(i_Q)|\pi_\gamma,$$

is an satisfying assignment to ϕ_γ. As in [1] we write $y(i_{Q+1}) = \pi_\gamma$. Observe that we can now use V_2's subroutine to check this assignment very efficiently (by definition). We require that each symbol $y(i_i)$ is now encoded as V_2 requires.

Namely, we have now a verifier V_3. The proof string for V_3, y', is of format constructed above with a table of 2^R entries called Γ. The "program" for V_3 runs as follows. Pick a random $\gamma \in_U \{0,1\}^R$ and a random $\gamma' \in_U \{0,1\}^r$. We use V_1 to generate locations $i_1(\gamma), \dots, i_{Q+1}(\gamma)$ and the $\mathbf{3-SAT}$ formula ϕ_γ as above. Now run V_2 using the random string γ' and check that

$$y'(i_1(\gamma))|\dots|y'(i_{Q+1}(\gamma))|\Gamma(\gamma)$$

encodes a satisfying assignment for ϕ_γ. Then V_3 accepts iff V_2's subroutine accepts. This completes the construction.

From the above it is immediate that the complexity results hold. The correctness of the construction requires more effort, but for this we defer interested readers to [1]. ■

The lemma 2.4 gives us tools to "prove" the **PCP** theorem. Namely, we need to assume only two other theorems (with long proofs) for this results.

Theorem 2.5 $\mathbf{3-SAT}$ *has a* $(\log n, 1, \log n)$*-restricted normal form verifier.*

Theorem 2.6 $\mathbf{3-SAT}$ *has a normal-form verifier that is* $(n^3, 1, 1)$*-restricted, and uses alphabet* $\{0, 1\}$.

We do not formally prove these results, although some tools needed for their proofs are described in the following section (indeed, we will sketch the steps xhow to construct the normal-form verifier of theorem 2.5, and the verifier for theorem 2.6 follows through similar steps). However, the above theorems are just the right ones to prove the theorem 1.4.

Corollary 2.7 $\mathbf{3-SAT}$ *has a normal-form verifier that is* $(\log n, 1, \log\log n)$*-restricted.*

Proof. Indeed, use the theorem 2.5 to play the role of the verifiers to be composed in the lemma 2.4. ∎

Corollary 2.8 *There exists a normal-form verifier for* $\mathbf{3-SAT}$ *that is* $(logn, 1, 1)$*-restricted, and the alphabet is* $\{0, 1\}$.

Proof. Indeed, the verifier of corollary 2.7 and 2.6 compose giving the result. ∎

Proof. (Proof of theorem 1.4.) Indeed, the corollary 2.8 indicates that $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$. Furthermore, the verifier is in normal form. ∎

This section has left us with the trouble of finding proofs for the theorems 2.5 and 2.6. It is not possible to work through all the necessary definitions and details, however, we will indicate some steps towards understanding them.

3 Constructing normal-form verifiers

Throughout this section we skip the verification of split assignments, as these are quite easy to workout from the general case. Sufficiently detailed description of those steps may be found from [1].

The main idea here is to construct a $\mathbf{3-SAT}$ normal-form verifier that satisfies the theorem 2.5. However, this overview is far from complete in the sense, that proofs are omitted (hardest part being usually the case when our formula ϕ is not satisfiable).

Let us sketch the steps. First, we need to embed the initial satisfiability proof into some space where two proofs will be far away from each other. That is, the produced encoding of the proof should be robust in a sound way. Second, we need algorithms that first check that the given proof looks like a satisfiability proof and then is able to check that it is indeed such (given that it looks like one). We will now give further details.

Suppose $\phi \in \mathbf{3-CNF}$ throughout. Let $a = (a_1, \ldots, a_N)$ be a satisfying assignment to ϕ, namely $\phi(a) = 1$. Then we wish to encode a in a robust way, or in otherwords we will embed it into a larger space in such a way that two assignment have large (enough) minimal distance. Most of the details are ignored here and only the high-level construction is illustrated.

Define the integers h, m as $(h+1)^m = N$ and find a suitable $q = p^n$, where p is a prime number, such that $h < q$. The faction h/q should be large, but q is polynomially bounded in terms of h. So this gives us a isomorphism

$$\xi : \{0, \ldots, N\} \to \{0, \ldots, h\}^m,$$

and some injection

$$\zeta : \{0, \ldots, h\}^m \to \mathbb{F}_q^m.$$

We fix one such injection for the rest of this section. Now we may find some polynomial $f \in \mathbb{F}_q[X_1, \ldots, X_m]$ such that

$$f(x) = a_y, \text{ when } x = (\zeta \circ \xi)(y).$$

Such a polynomial is easy to find by multivariate polynomial interpolation and one sees that $\deg f \leq mh$. From now on we may assume that the proof for satisfiablity of ϕ is indeed some polynomial $f : \mathbb{F}_q^m \to \mathbb{F}_q$ and few additional tables. (In this survey we usually ignore details about the tables, although one must be careful about their sizes in the formal proof of **PCP** theorem.)

Let us write

$$\sigma : (\{0, \ldots, N\} \to \{0, 1\}) \to (\mathbb{F}_q^m \to \mathbb{F}_q),$$

for our map from satisfying assignments to polynomials. Thus we may consider it as a way of *coding* the satisfying assignments. Let $\delta \in [0,1]$ denote the (relative) minimal distance of two codewords in codσ. We may now describe an algorithm for checking whether some given $f : \mathbb{F}_q^m \to \mathbb{F}_q$ is actually δ-close to some satisfying assignment. This is called the *low-degree test*.

It is important to understand that we encode the function f not as a polynomial but a table of outputs. This gives a large table, but only polynomial is size. Notice, that this means that the encoding of f doesn't reveal whether or not it is a low-degree polynomial at all. Hence, we need a efficient algorithm which uses as few queries of f's values as possible. Recall, that we always assume that the proof contains the table for f and some additional tables.

It is a convention to speak about lines $\ell : \mathbb{F}_q \to \mathbb{F}_q^m, x \mapsto \omega_1 + x\omega_2$ in this context. Observe that ℓ may be identified by the points $\{\ell(x) : x \in \mathbb{F}_q\}$. We use this identification in the sense that lines which produce same sets are consider identical (i.e. we take the class of lines under this equivalence). With all these definitions the following steps define the low-degree test (or test for δ-closeness);

1. Pick $k = c/\delta$, for some fixed c, lines $\ell_1, \ldots, \ell_k$, and points $z_1 = \ell(y_1), \ldots, z_k = \ell(y_k)$, where $y_1, \ldots, y_k \in_U \mathbb{F}_q$;
2. Read univariate polynomials $p_{\ell_1}, \ldots, p_{\ell_k}$ from the proof string (i.e. from the additional tables), and the values of $f(z_1), \ldots, f(z_k)$ of the function f;
3. If for all $i = 1, \ldots, k$ we have $p_{\ell_i} = f(z_i)$ then *accept*; otherwise *reject*.

We have not defined what the polynomials p_{ℓ_i} should be. However, it is clear from the construction that these polynomials cannot be anything else but compositions with f, i.e. $p_{\ell_i} = f \circ \ell_i$ for the algorithm to work. It is important to observe that the algorithm doesn't trust that this would hold, and the polynomials p_{ℓ_i} are something else then with large probability the low-degree test rejects.

Full analysis of the low-degree test requires significant amount of work (for the hard case). However, details may be found from [1] and more recent details from [3].

Given that we may now assume that f is δ-close to some assignment we wish to compute few selected values of the satisfying assignment. We now have as an input some number of points $z_1, \ldots, z_s \in \mathbb{F}_q^m$ which indicate the points where the satisfying assignment, let us denote it by $\bar{f}$, is to be evaluate. The algorithm can be explained as follows;

1. Select $y_1, \ldots, y_{s+1}, z_{s+1} \in_U \mathbb{F}_q$ and find a polynomial map $\kappa : \mathbb{F}_q \to \mathbb{F}_q^m$ such that $\kappa(y_i) = z_i$ by interpolation;
2. Pick $y \in_U \mathbb{F}_q$ and let $z = \kappa(y)$;
3. Read polynomial T_κ and the value $f(z)$ of the proof string;
4. If $T_\kappa(y) = f(z)$ then output values $T_\kappa(y_1), \ldots, T_\kappa(y_s)$ as the values $\bar{f}(z_1), \ldots, \bar{f}(z_s)$; otherwise *reject*.

We state without a proof that if f is δ-close then the probability that this algorithms fails is $2\sqrt{\delta} + o(1)$.

The above algorithms are all coding theory that is needed in the proof of **PCP** theorem. However, slight and simplified variations are utilized over the field $\mathbb{F}_2$ when f is assumed to be linear. This case is required for the proof of theorem 2.6, but we ignore the details are refer to [1].

Infact, the algorithms presented this far may be used to prove the following theorem.

Theorem 3.1 *Let L be a language. For every normal-form verifier V for L that is $(r(n), q(n), \mathbf{poly}(t(n))$-restricted there is a normal-form verifier V' that is $(r(n)(1+\dfrac{\log t(n)}{logr(n)}), 1, t(n) + r(n))$-restricted.*

Proof.(Proof sketch, see [1] for further details.) Indeed, write the proof in the format described above. Now it is clear that the queries for V' are aggregated into $O(1)$ by construction. Rest can be seen with a more careful analysis of the construction. ∎

We have yet only considered encodings of the satisfying assignments, but we need to be able to check whether they really satisfy ϕ. For this it is necessary to arithmetize the $\mathbf{3-SAT}$ instance ϕ and find an algorithm that can check satisfiability efficiently.

Let $\chi_j : \{0, \ldots, h\}^m \times \{0, \ldots, h\}^m \to \{0,1\}$ be a family of functions such that $\chi_j(c, v) = 1$ if v is the jth variable of the clause v from ϕ and 0 otherwise. By abuse of notation we denote by χ_j the polynomial obtained by lifting it to $\mathbb{F}_q$. Let

$s_j : \{0,\dots,h\}^m \to \{0,1\}$ be a family of functions such that $s_j(c) = 1$ if the jth variable is unnegated in the clause c in ϕ and 0 otherwise. Again we abuse the notation and denote by s_j also its lifting to $\mathbb{F}_q$.

We may thus write the satisfiability of the clauses of ϕ as

$$\prod_{j=1}^{3} \chi_j(c, v_j)(s_j(c) - f(v_j)) = 0,$$

where $c, v_1, v_2, v_3 \in \{0,\dots,h\}^m$. We represent by $g_f : \mathbb{F}_q^m \to \mathbb{F}_q$ the polynomial that can be constructed from those equations. That is, if $g_f(x) = 0$ for all $x \in \{0,\dots,h\}^{4m} \subseteq \mathbb{F}_q^m$ then ϕ is satisfiable (and the proof is its satisfying assignment).

Our task is now to find a way to verify that f is the satisfying assignment for ϕ with small (less than some constant) error probability. To do so we randomize the problem using the following lemma.

Lemma 3.2 ("Zero-testers") *There exists a family of $c = q^{O(m)}$ polynomials $r_1,\dots,r_c \in \mathbb{F}_q[X_1,\dots,X_{4m}]$ such that if $f : \{0,\dots,h\}^{4h} \to \mathbb{F}_q$ is any function not identically zero then if r is chosen randomly from this family then*

$$\Pr[\sum_{y\in\{0,\dots,h\}^{4m}} r(y)f(y) = 0] \le \epsilon,$$

for some constant $\epsilon < 1$ and r_i's can be constructed in $q^{O(m)}$ time.

Proof.(Proof sketch, more details may be found in [1].) We put

$$r_i(y_1,\dots,y_{4m}) = \prod_{j=1}^{4m} I_{t_j}(y_j),$$

where

$$(t_1,\dots,t_{4m}) = i,$$

for some suitable encoding of the indices, and

$$I_{t_i}(0) = 1, \text{ and } I_{t_i}(\xi_j) = t_i^j,$$

for some $\xi_j \in \mathbb{F}_q$ and $j \in \{1,\dots,h\}$. This provides the construction and by Schwartz-Zippel lemma the bound. ■

The missing piece now is an algorithm that can prove the identity

$$\sum_{y\in\{0,\dots,h\}^{4m}} r(y)g_f(y) = 0.$$

Indeed, we need to do this efficiently, but with only a small margin for error. The algorithm for this is called *sum-check* procedure.

Let $b \in \mathbb{F}_q[X_1,\dots,X_\ell]$ (and infact we take $b = r \circ f$, but ignore this for now). Then we use the convenient notation

$$b_{a_1,\dots,a_{i-1}}(y_i) = \sum_{y_{i+1},\dots,y_\ell \in H} b(a_1,\dots,a_{i-1},y_i,\dots,y_\ell),$$

where $H \subseteq \mathbb{F}_q^\ell$. In particular, we have translated the b into a family of univariate polynomials. This is important as we need large number of them and helps to tabulate them along the given proof.

Suppose that we know the value

$$c := \sum_{y_1,\dots,y_\ell \in H} b(y_1,\dots,y_\ell),$$

and indeed $c = 0$ for us. Thus we may run the following procedure;

1. Put $v := c$;
2. Pick random $a_1,\dots,a_\ell \in \mathbb{F}_q$;
3. For $i = 1,\dots,\ell$;
   (a) Look up univariate polynomial $g_{a_1,\dots,a_{i-1}}$ from the proof string;
   (b) If $v \ne \sum_{y_i\in H} g_{a_1,\dots,a_{i-1}}(y_i)$ then *reject*; otherwise put $v := g_{a_1,\dots,a_{i-1}}(a_i)$.
4. Compute the value $b(a_1,\dots,a_\ell)$;
5. If $g_{a_1,\dots,a_{\ell-1}}(a_\ell) \ne b(a_1,\dots,a_\ell)$ then *reject*; otherwise *accept*.

To see that this procedure is correct in the case that ϕ is satisfiable requires only little inspection. The reader may view the procedure as follows; we pick a path $a_1,\dots,a_\ell$ through a particular tree and then check that at each point the tables given match up, then the initial value c must have been correct. Indeed, it is possible to see that the error probability is small and details may be found again from [1].

The construction given is infact sufficient to prove the theorem 2.5. To prove theorem 2.6 we need, as already mentioned, the linear functions over $\mathbb{F}_2$, but these require even less than the above.

Indeed, these algorithms suffice to construct normal-form verifiers for $\mathbf{3-SAT}$ that are $(\log n, 1)$-restricted. Hence, we have concluded our overview of the **PCP** theorem.

References

[1] Sanjeev Arora, *Probabilistic Checking of Proofs and Hardness of Approximation Problems*, Technical Report CS-TR-476-94, Princeton University, 1994.

[2] Sanjeev Arora, *Around the PCP Theorem*, Technical report SOSC-97.2, Schoold of Computer Science, McGill University.

[3] Sanjeev Arora and Madhu Sudan, Improved low-degree testing and its applications. Reprint.

[4] Laszlo Babai, Transparent Proofs and Limits to Approximation.

[5] Johan Hstad, Some optimal inapproximability results, September 24, 1999.

[6] Petteri Kaski, T-79.511 Special Course on Cryptology / Zero Knowledge: Rudiments of Interactive Proof Systems, September 26, 2001.

[7] Madhu Sudan, *Probabilistically Checkable Proofs*, Lecture notes, IAS/Park City Mathematics Series, Scribe: Venkatesan Guruswami.

[8] Daniel Spielman, *Computationally Efficient Error-Correcting Codes and Holographic Proofs*, Dissertation, June 1995, Massachusetts Instititute of Technology.

[9] Lauri Tarkkala, T-79.511 Special Course on Cryptology / Zero Knowledge: IP=PSPACE, October 1, 2001.