1 $NP \subset PCP(\log n, (\log n)^c)$

Lecture 3: March 26 and April 2, 2003

The PCP theorem states that $NP \subset PCP(\log n, (\log n)^c)$. We build gradually towards its proof. As an intermediate step we show that $NP \subset PCP(\log n, (\log n)^c)$ for some constant $c > 1$. The presentation here is taken in most parts from [4] (which is heavily based on [2], where $NP \subset PCP(\log n \log\log n, \log n \log\log n)$ was proved. We save a $\log\log n$ factor in the number of random bits (at a cost of a polylogarithmic factor in the number of query bits) by making the following change to the FGLSS protocol: we use low degree extensions rather than multilinear extensions. These low degree extensions were used in [1].

So as to save myself some rewriting, I kept the original notation of the FGLSS paper. Hence the verifier will be denoted by M, and it has access to an "oracle" O (the PCP witness). The whole PCP system together is referred to as a probabilistic oracle-machine. In this notation, we prove:

Theorem 1 *Any language $L \in NP$ is accepted by a probabilistic oracle-machine M such that that uses $O(\log n)$ random bits and queries $O((\log n)^c)$ bit locations, where $c > 1$ is some universal constant. M takes polynomial time to decide upon the query locations and to write down a list of tests to be made on the replies to the queries. Evaluating these tests takes polylogarithmic time.*

We now proceed to describe the ingredients of our protocol.

1.1 Polynomials

We recall here some simple facts regarding polynomials and multinomials over any (finite) field $\mathcal{F}$.

Lemma 2 *For every set of $d+1$ (point,value) pairs $\{(a_i, b_i) : 1 \leq i \leq d+1\}$ (where $a_i, b_i \in \mathcal{F}$ and the a_is are distinct), there is a unique polynomial $p(x)$ of degree d such that $p(a_i) = b_i$ for every i.*

Proof: Let

$$L_i(x) = \Pi_{j \neq i} \frac{x - a_j}{a_i - a_j}$$

be the polynomial that is 1 at a_i and 0 at all a_j for $j \neq i$. Set

$$p(x) = \sum b_i L_i(x).$$

Uniqueness follows the fact that over a field, every two degree d polynomials either agree on at most d points or agree on all points. □

Lemma 3 *Let $|\mathcal{F}| > d$. For every set of $(d+1)^m$ (point,value) pairs $\{(a_i, b_i)\}$, where $a_i \in [0,d]^m$, $b_i \in \mathcal{F}$, and the a_is are distinct, there is a unique multinomial $p(x_1, \ldots, x_m)$ of degree d in every variable such that $p(a_i) = b_i$ for every i.*

Proof: For $a_i = (a_i^1, \ldots, a_i^m)$, let

$$L_i(x) = \Pi_{k=1}^{m} \Pi_{j \neq a_i^k} \frac{x - j}{a_i^k - j}$$

be the degree d multinomial that is 1 at a_i and 0 at all other points in $[0,h]^m$. Set

$$p(x) = \sum b_i L_i(x).$$

Uniqueness follows from the uniqueness for univariate degree d polynomials, and is omitted. □

1.2 Arithmetization

If suffices to show a procedure for deciding the **NP**-complete language 3-SAT. Let f be a given 3CNF formula with n clauses and n variables. Let h, m be integers satisfying $(h+1)^m = n$, with $h \simeq \log n$ and $m \simeq \log n / \log\log n$. (We assume for simplicity that in f the number of variables is equal to the number of clauses, and that n is such that h, m as above exist. These assumptions are made without loss of generality, because f can always be padded by dummy clauses and variables.) Let c denote a clause number and $\vec{v}$ denote a variable name. To prepare for the later degree-h representation we represent the variable names as strings in $[0,h]^m$.

For $(j = 1,2,3)$ and for clause c, let $\chi_{j,c}$: $[0,h]^m \to \{0,1\}$ be an indicator function satisfying:

$$\chi_{j,c}(\vec{v}) = \begin{cases} 1 & \text{if } \vec{v} \text{ is the } j\text{th variable of } c \\ 0 & \text{otherwise} \end{cases}$$

As an example, assume that $h = 2$, $m = 2$, $n = (h+1)^m = 9$, and that the first variable of the fourth clause is x_5 (represented in ternary as 12). Then $\chi_{1,4}(\vec{v})$ is the multinomial

$$\frac{v_1 - 0}{1 - 0} \cdot \frac{v_1 - 2}{1 - 2} \cdot \frac{v_2 - 0}{2 - 0} \cdot \frac{v_2 - 1}{2 - 1}$$

which evaluates to 1 if $v_1 = 1$, $v_2 = 2$, and evaluates to 0 otherwise. Note that $\chi_{1,4}$ is of degree h in each of its variables, and this is the general rule for each of the $\chi_{j,c}$.

For j and c as before, let $s_{j,c}$ be a shorthand notation satisfying:

$$s_{j,c} = \begin{cases} 1 & \text{if the } j\text{th variable of } c \text{ is positive} \\ 0 & \text{if complemented} \end{cases}$$

A truth assignment A is a function from variable names to Boolean values

$$A: [0,h]^m \to \{0,1\}.$$

The following expression over any field $\mathcal{F}$ (CC standing for clause check)

$$CC(A,c) = \sum_{\vec{v}_1,\vec{v}_2,\vec{v}_3 \in [0,h]^m} \prod_{j=1}^{3} \chi_{j,c}(\vec{v}_j) \cdot (s_{j,c} - A(\vec{v}_j))$$

evaluates to 0 iff A satisfies clause c of f.

In order to verify that A is a satisfying assignment, one needs to check that for every clause c, $CC(A,c) = 0$. An oracle machine can do this by requesting the complete description of A from the oracle. However, this uses n answer bits. We develop a different procedure which requires only a polylogarithmic number of answer bits.

Let $\vec{v} = v_1, .., v_m$. We extend the arithmetization in such a way that v_i will take arbitrary values from $\mathcal{F}$. This requires to extend the domain of the functions A and $\chi_{j,c}$ from $[0,h]^m$ to $\mathcal{F}^m$.

Definition 1 *Let $\mathcal{F}$ be any field. Given a function $f: [0,h]^m \to \mathcal{F}$, the* degree-$h$ *of f over $\mathcal{F}$, denoted $\hat{f}: \mathcal{F}^m \to \mathcal{F}$, is such that:*

1. *$\forall \vec{y} = y_1, ..., y_m$, where $y_i \in [0,h]$: $\hat{f}(\vec{y}) = f(\vec{y})$*
2. *$\hat{f}$ can be expressed as a multinomial of degree h in all variables $y_1, ..., y_m$.*

The procedure of constructing the degree-h extension of a multi-variate Boolean function f was given in Lemma 3.

M can easily compute degree-h arithmetic expressions for each $\chi_{j,c}$, and can easily determine each $s_{j,c}$. The oracle can hold the degree-h representation of A. This can be viewed as a large binary table, as each character of $\mathcal{F}$ can be encoded using $\log|\mathcal{F}|$ bits. Each $CC(A,c)$ is a multinomial of degree $2h$ (because when j ranges over 3 values, it ranges over distinct sets of variables, and for each value of j we multiply two degree-h multinomials). Since M does not know the value of A on each $\vec{v} \in [0,h]^m$, M cannot check that the value of each $CC(A,c)$ is 0. Hence we shall use an approach that has two major components:

1. M verifies that the extended A that the oracle holds is (almost) a degree-h multinomial, and thus the extended CC is a multinomial of degree $2h$.
2. Assuming that CC is a multinomial of degree $2h$, M can now use an [5] type protocol to verify, with high probability, that for all c,, $CC(A,c) = 0$. In this part, the oracle provides M with the value of the degree-h extension of A on a small number of arguments which M chooses.

We show how to do part 1 in subsection 1.4, and part 2 in subsection 1.3

1.3 Verifying Simultaneously that All Clauses Evaluate to Zero

We now show how to verify with high probability that for all c, $CC(A, c) = 0$. Clearly, this would imply that A satisfies f. All arithmetic in this and the next section is done over the finite field $\mathcal{F}$, where $\mathcal{F}$ is some "large enough" finite field ($|\mathcal{F}| > h^3m^2$ suffices).

We first show how to verify that the arithmetic expression of a single clause evaluates to 0. We then show how all clauses can be tested simultaneously.

1.3.1 The Sum-Check Protocol

Given a multinomial φ of degree d (in each of its variables) and a constant c, we want to check that

$$\sum_{\vec{y}\in[0,h]^m} \varphi(\vec{y}) = c \qquad (*)$$

This can be done inefficiently by evaluating φ on the $(h+1)^m$ different points $\vec{y} \in [0, h]^m$. However, [5] developed a procedure for checking $(*)$ which requires the evaluation of φ only on a single random point $\vec{y}$ which belongs to the extended field $\mathcal{F}^m$. [5] used this protocol because the number of terms in the summation was too large for the performing a direct computation. In contrast, we use the protocol because M does not have an explicit expression for φ (M does not know A), and the amount of communication required to obtain such an expression is prohibitingly large. In [2], this protocol was used for both the above reasons.

We describe an adaptation of the [5] protocol to our purpose. Since φ is a multinomial of degree d, the function

$$g(y_1) = \sum_{y_2,\ldots,y_m\in[0,h]} \varphi(\vec{y})$$

is a polynomial of degree d.

To prove $(*)$, the oracle sends M a polynomial g' of degree d, and claims that $g' = g$. If $(*)$ is false, then there are two possibilities:

1. $\sum_{j=0}^{h} g'(j) \neq c$ (which M can determine easily),
2. $g' \neq g$, and since both g and g' are polynomials of degree d, they agree on at most d values. To check this condition, M picks at random a $k \in \mathcal{F}$ and checks whether $g'(k) = \sum_{y_2,\ldots,y_m\in[0,h],y_1=k} \varphi(\vec{y})$. Note, that if $g' \neq g$ then with probability at least $1 - \frac{d}{|F|}$ (taken over the choices of k), $g'(k) \neq g(k) = \sum_{y_2,\ldots,y_m\in\{0,1\},y_1=k} \varphi(\vec{y})$. Note also, that this condition is of the same form as (*), except for only $m-1$ variables and will be checked recursively.

M continues this procedure for $m-1$ more rounds. In round i, M picks a random value for y_{i-1}, sends to the oracle the values of all variables instantiated so far (which are y_1 ... y_{i-1}), and requests a d degree polynomial in the variable y_i to test the polynomial that results from the previous instantiations. When all variables are finally instantiated, M checks that for $\vec{y}$ that was formed, $\phi(\vec{y})$ indeed achieves the value claimed. For $|\mathcal{F}| > \frac{md}{\epsilon}$, the probability of falsely accepting is bounded by the – the sum over m rounds of possible error probability $d/|\mathcal{F}|$ in every round, which is at most ϵ. Note that, the number of random bits used by M is $O(m \log |\mathcal{F}|)$ and that the number of answer bits received by M is $O(md \log |\mathcal{F}|)$. An important observation is that the test requires that M checks the value of φ only at one single point when $\vec{y}$ is fully instantiated.

The above protocol can be used then to check that a single clause $CC(A,c) = 0$ (i.e., the role of $\sum_{\vec{y} \in [0,h]^m} \varphi(\vec{y})$ is played by $CC(A,c) = \sum_{\vec{v}_1,\vec{v}_2,\vec{v}_3 \in [0,h]^{3m}} \prod_{j=1}^{3} \chi_{j,c}(\vec{v}_j) \cdot (s_{j,c} - A(\vec{v}_j))$ where $\vec{y} = \vec{v}_1\vec{v}_2\vec{v}_3$). Observe that the last step of the sum-check protocol requires M to compute $\prod_{j=1}^{3} \chi_{j,c}(\vec{v}_j) \cdot (s_{j,c} - A(\vec{v}_j))$, where $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ have been are instantiated, but M cannot compute this on its own as M does not have a description of the function A. Instead, M requests the values of $A(\vec{v}_1)$, $A(\vec{v}_2)$ and $A(\vec{v}_3)$ from the oracle and uses values received for the computation of $\prod_{j=1}^{3} \chi_{j,c}(\vec{v}_j) \cdot (s_{j,c} - A(\vec{v}_j))$. We remark that the reliability of the sum-check test does not change too much even if a small constant fraction of the values of A is incorrect.

1.3.2 Constructing a Single Expression From Many Clauses

The sum-check protocol can be used to test the value of a single expression. However, in our case, M wants to test the value of a polynomial number of expressions. Checking each one of them separately requires too much communication. Thus we describe a way of checking them simultaneously. This is done by constructing one single expression $E(A)$ whose value simultaneously reflects the values of all the 2^m expressions $CC(A,c)$. This single expression is still of degree $2h$ in each of its variables and so the sum-check protocol can be used to check its value. (i.e, $\sum_{\vec{y} \in [0,h]^m} = E(A)$).

A naive attempt is to construct $E(A)$ as $\sum_c CC(A,c)$. If for each clause c, $CC(A,c)$ evaluates to 0, we know that also $E(A)$ would evaluate to 0. Unfortunately, if there exists some c such that $CC(A,c) \neq 0$ over $\mathcal{F}$, it still may be true that $E(A) = 0$ over $\mathcal{F}$.

In a more sophisticated approach M chooses independently and uniformly weights $w_c \in \mathcal{F}$ for each clause c, sends all w_c values to the oracle, and requests the oracle to convince it that $E(A) = \sum_c CC(A,c) \cdot w_c = 0$ over $\mathcal{F}$. If $\forall c$, $CC(A,c) = 0$, then also $E(A) = 0$. If $\exists c$ such that $CC(A,C) \neq 0$, then $prob_{w_c}(E(A) = 0) = 1/|\mathcal{F}|$. This randomized reduction would be good enough for our purpose, if not for one problem: M uses polynomially many random bits in generating the independent w_c, defeating our attempt to keep their number small ($O(\log n)$).

We now describe a solution that assigns the weights pseudo-randomly, and uses only $m \log |\mathcal{F}|$ truly random bits.

Lemma 4 *For $1 \leq i \leq m$, let c_i denote the i^{th} character of the $(h+1)$-ary representation of c, where c is the index of a clause. Let $R = (r_0, r_1, ..., r_{m-1})$, where each r_i is an element chosen independently at random from $\mathcal{F}$. Consider the family of expressions:*

$$E_R(A) = \sum_{c \in [0,h]^m} CC(A,c) \cdot \prod_{i=1}^{m} r_i^{c_i}$$

over all possible choices of R.

1. *If $\forall c$:$CC(A,c) = 0$, then $\forall R : E_R(A) = 0$.*
2. *If $\exists c$:$CC(A,c) \neq 0$, then with probability at least $1 - hm/|\mathcal{F}|$ (over the choices of R): $E_R(A) \neq 0$.*

Proof: View the $n = (h+1)^m$ expressions $CC(A,c)$, $c \in [0,h]^m$ as the coefficients of an m-variable degree-h multinomial over $\mathcal{F}$ where the variables are $r_0, ..., r_{m-1}$. Then, the expression $E_R(A)$ corresponds to evaluating this multinomial on a point $R \in \mathcal{F}^m$. It is well known that unless all coefficients are 0, a multilinear function is nonzero on at least a $(1 - h/|\mathcal{F}|)^m$-fraction of its points (easy proof by induction on m). The proof of the lemma follows. $\square$

Remark: Observe that in effect we use a linear error detecting code of large Hamming distance over the alphabet $\mathcal{F}$. The message (sequence of values of the individual expressions) of length $(h+1)^m$ is encoded by a code word of length $|\mathcal{F}|^m$, and the verifier checks whether a random character from this code word is zero. Other codes can be used instead of our construction.

Let us summarize the procedure for verifying simultaneously that for all clauses c, $CC(A,c) = 0$. Machine M chooses at random $R \in \mathcal{F}^m$, and applies the sum-check procedure to expression $E_R(A)$. The sum-check procedure can be applied as expression $E_R(A)$ (for fixed R and summing over c) is a multinomial of degree $2h$ in each variable with $3m$ variables $\vec{v_1}\vec{v_2}\vec{v_3}$ in $\mathcal{F}$, and at the final stage of the sum-check procedure applied to $E_R(A)$, machine M queries the oracle for the value of the extended A at the three independently distributed locations chosen during the checksum protocol.

1.4 The Degree-h Test

First we introduce some notation; the *distance* between two functions $f_1, f_2 : \mathcal{F}^m \to \mathcal{F}$,

$$\Delta(f_1, f_2) = \frac{|\{\vec{y} | f_1(\vec{y}) \neq f_2(\vec{y})\}|}{|\{\text{all } \vec{y}\}|}$$

i.e., the fraction of $\mathcal{F}^m$ on which f_1 and f_2 disagree. Let DH be the set of degree-h multinomials defined on $\mathcal{F}^m$. We define the *minimal distance* between a function $f : \mathcal{F}^m \to \mathcal{F}$ and a degree-h multinomial

$$\Delta_{DH}(f) = \min_{f' \in DH} \Delta(f, f').$$

We call a set of $|\mathcal{F}|$ points $\left\{y_1, \ldots, y_{|\mathcal{F}|}\right\} \subseteq \mathcal{F}^m$ an *aligned line* in direction i if they differ only in the i^{th} coordinate.

It is easy to see that a function is a degree-h multinomial if and only if it is a degree h univariate polynomial over every alligned line.

1.4.1 The Test Itself

We devise a multilinearity test with the following properties:

1. If f is a degree-h multinomial, the test always accepts.
2. If $\Delta_{DH}(f) \geq 0.1$, the test rejects with probability at least $1/2$.

DEGREE-h TEST (random sampling version)

1. For large enough constant k, randomly choose km *sample tests*, where a sample test is an aligned line and one random point on the line. Within each line, ask the oracle for the value of f on the first $h+1$ points, and of the random point.
2. Accept iff for every aligned line, the respective $h+2$ points agree with some degree h univariate polynomial.

The proof that the above test indeed works is rather complicated, and is postponed to a later lecture. In essence, it shows that when $|\mathcal{F}|$ is large enough relative to h (e.g., $|\mathcal{F}| > h^3 m^2$) and $\Delta_{DH}(f) \geq 0.1$, then the fraction of rejecting sample tests out of all sample tests is $\Omega(1/m)$.

1.4.2 Recycling random bits

The above test is wasteful in the number of random bits that the verifier uses. Generating each of the $O(m)$ samples requires $O(m \log |F|)$ random bits. Thus instead of generating each of the samples independently, we want an easy to compute sampling procedure which uses only $O(m \log |F|)$ random bits, generates $O(m)$ samples, and "hits" a rejecting sample test with probability at least $1/2$.

Problems of this type can be handled by the method of *two-point sampling* [3]. The basic idea behind two point sampling techniques is that pairwise independent sample points (or sample tests, in our case) share many of the properties of mutually independent sample points especially with respect to hitting sets of small density with constant probability, but require much less random bits to generate. One possible implementation is as follows.

Identify the set of sample points (all sample tests) $\mathcal{T}$ with a field $\mathcal{K}$. Consider the family

$$HASH = \{g_{a,b} | a, b \in \mathcal{K}\},$$

where $g_{a,b}$ is defined as the function $ax + b$ computed over the field $\mathcal{K}$.

Pick any set $K \subseteq \mathcal{K}$ such that $|K| = 2km$. The set system $\{K_{a,b} \mid a, b \in \mathcal{K}\}$ is defined by $K_{a,b} = \bigcup_{x \in K} g_{a,b}(x)$ for $a, b \in \mathcal{K}$. The desired properties of this system follow from the following properties of the hash functions:

1. (Uniformity) For any pair $x, y \in \mathcal{K}$ the probability for a randomly chosen hash function $g_{a,b}$ that $g_{a,b}(x) = y$ is exactly $1/|\mathcal{K}|$.
2. (Pairwise Independence) For any $x_1 \neq x_2 \in \mathcal{K}$ and for any $y_1, y_2 \in \mathcal{K}$ the probability that for a randomly chosen hash function $g_{a,b}$ the equations $g_{a,b}(x_1) = y_1$ and $g_{a,b}(x_2) = y_2$ hold is $1/|\mathcal{K}|^2$.

Consider a subset $H \subseteq \mathcal{K}$, and denote $|H|/|\mathcal{K}|$ by p. (In our case, H corresponds to the set of rejecting sample tests, and $p = 1/km$.) For $x \in \mathcal{K}$ let V_x be the indicator function of the event that a random $g_{a,b}$ maps x into H. $E(V_x) = p$, because of the uniformity property. Consider

$$V = \sum_{x \in K} V_x.$$

The probability that a random $K_{a,b}$ intersects with H is exactly $Prob(V > 0)$. We have $E(V) = 2kpm$ and $\sigma^2(V) = 2kp(1-p)m$ (because of the pairwise independence of the variables V_x). From the Chebyshev inequality:

$$Prob(V \leq 0) \leq \frac{\sigma^2(V)}{(E(V))^2} = \frac{1-p}{2kpm} < \frac{1}{2}.$$

For other suggested implementations of two-point sampling, the reader is referred to [3].

We are now ready to present our efficient multilinearity test.

PAIR-WISE INDEPENDENT DEGREE-h TEST (pair-wise independent sampling version) 1. Randomly choose two initialization points for the two point sampling procedure. Deterministically generate $2km$ pairwise independent sample tests. Ask the oracle for the value of f on the $h+2$ points of every sample tests. 2. Accept iff all of the sample tests correspond to degree-h univariate polynomails.

Lemma 5 *Let $|\mathcal{F}| \geq h^3m^2$ and let $f : \mathcal{F}^m \rightarrow \mathcal{F}$ be an arbitrary function.*

1. *If f is of degree h, the pair-wise independent degree-h test always accepts.*
2. *If $\Delta_{DH}(f) \geq 0.1$, the pair-wise independent degree-h test rejects with probability at least 1/2.*

1.5 Putting the Pieces Together

We are now ready to prove Theorem 1.

Proof: Let f be a 3CNF formula of length n. Set $h \simeq \log n$, $m \simeq \log\log n$, such that $(h+1)^m = n$. Let $\mathcal{F}$ be the smallest finite field of order $|\mathcal{F}| > h^3m^2$. (We remark that since h, m are relatively small - $O(\log n)$, the smallest prime that is larger than h^3m^2 can be found in deterministic polynomial time.) All computations in the following protocol are made over $\mathcal{F}$.

1. Arithmetize f and obtain degree-h representations as described in Subsection 1.2. In particular, let the (truthful) oracle hold the degree-h representation of a satisfying assignment A.
2. Perform on A the degree-h test (pair-wise independent sampling version) as described in Subsection 1.4.2. I.e query the oracle for the value of the extended A on points selected by the degree-h test. If the test fails, reject. Else, continue.
3. Select a random R and construct the expression $E_R(A)$, as described in Subsection 1.3.2. Prefix this R to any message sent in Step 4 of the protocol. (In Step 4 the oracle has to know which expression M has in mind.)

4. Perform the sum-check protocol (as described in Subsection 1.3.1) on the expression $E_R(A)$. Reject if at any stage the oracle's replies violate the consistency check described as possibility 1 in Subsection 1.3.1. As described in the sum-check protocol, this step entails queries to the oracle for degree-$2h$ polynomials corresponding to various expressions.

5. When Step 4 ends, M is left with three random arguments for the degree-h extension of A. One by one, request the values of A on these arguments from the oracle. Plug these values in their respective locations in the instantiated $E_R(A)$, and compute its value. Accept iff this value agrees with the value computed from the last polynomial sent by the oracle.

It is easy to see that if f is satisfiable, then M accepts the proof of the truthful oracle. If f is not satisfiable, then for any oracle, we distinguish between two cases. Denote by g the function obtained by enumerating all the oracle's answers when M requests values of the degree-h extension of A (these questions correspond to Steps 2 and 5 in the protocol).

1. $\Delta_{DH}(g) \geq 1/10$. In this case, by lemma 5, Step 2 rejects with probability at least $1/2$.

2. $\Delta_{DH}(g) < 1/10$. We show that Steps 3-5 reject with probability at least $1/2$.

   Assume, to the contrary, that oracle O is using function g, and the probability that M^O accepts Steps 3-5 above is at least $1/2$. Let g_h be the degree-h function that satisfies $\Delta(g, g_h) < 1/10$. We construct a new oracle O' that uses function g_h, in its responses in steps 2 and 5, and otherwise in step 4 answers in a manner identical to O. Namely, on any query of M that does not request explicitly the value of the degree-h extension of A at a point, the answer of O' is identical to the answer of O, but in order to answer queries on the value of the degree-h extension of A at a point, O' use the degree-h function g_h, rather than the function g that O uses.

   The oracle is queried on exactly three point of the degree-h extension of A at step 5, and each of these points is chosen uniformly at random. Since $\Delta(g, g_h) < 1/10$, the probability that O' is asked for the value of a point on which g and g_h disagree is at most $3/10$. Hence the probability that $M^{O'}$ accepts Steps 3-5 is at least $1/2 - 3/10 = 2/10$.

   Now we compute an upper bound on the probability that $M^{O'}$ accepts Steps 3-5. We first condition over the random choice of R. For f that is not satisfiable, the probability that $E_R(h)$ is identically 0 is at most $hm/|\mathcal{F}| < 1/100$ (by lemma 4). In this case we may assume that $M^{O'}$ accepts. For the case that $E_R(h)$ is not identically 0, we use the fact that g_h is of degree h, implying that $E_R(g)$ is of degree $2h$. By the analysis of the sum-check protocol, the probability that $M^{O'}$ accepts Steps 4 and 5 is at most $\frac{2h \cdot 3m}{|\mathcal{F}|} \leq 1/100$ when $|cF|$ is sufficiently large. Hence the probability that $M^{O'}$ accepts Steps 3-5 is at most $1/100 + 1/100 < 2/100$ which is in contradiction to our earlier argument that $M^{O'}$ accepts with probability at least $2/10$.

The total number of random bits that the verifier uses in all stages of the protocol is $O(m \log |\mathcal{F}|)$. As $|\mathcal{F}|$ is polynomial in h, this gives $O(m \log h) = O(\log n)$ random bits. The total number of bits that the oracle sends is: $O(mh \log |\mathcal{F}|)$ in Step 2, $O(mh \log |\mathcal{F}|)$ in Step 4, and $3|\mathcal{F}|$ in Step 5, totaling $O(\log^2 n)$.

To complete the proof of Theorem 1, we consider the running time of the verifier. The verifier first picks at random $O(m)$ pairwise independent sample tests for the degree h test, the value of R for the expression $E_R(A)$, and the three points $\vec{v}_1$, $\vec{v}_2$, $\vec{v}_3$, for the sum-check protocol. Then, before reading any characters from the oracle, M can simplify the expression $E_R(A)$ evaluated at $(\vec{v}_1, \vec{v}_2, \vec{v}_3)$, until M gets a multilinear expression in three unknowns, $A(\vec{v}_1)$, $A(\vec{v}_2)$, $A(\vec{v}_3)$, that has to be equal to a certain value that is also yet unknown (the outcome of the sum-check protocol). As a result, each test can be represented as a logical circuit of size polylogarithmic in n, and the number of tests is logarithmic in n. Now M reads the relevant characters from O (where each character can be encoded as a string of $\log |\mathcal{F}|$ bits) and substitutes them in the tests. Evaluating all tests then takes time that is polylogarithmic in n. $\square$

1.6 The role of the degree-h test

It is clear the the degree-h test plays a role in our proof of correctness for the protocol, but how can the oracle can cheat if the degree test is omitted? I do not know a satisfactory answer to this question. Here I give a partial answer that shows things that may go wrong in other sum-check protocols.

As a simplified example, assume that one is performing a sum-check protocol for an expression

$$E(A) = \sum_{\vec{x} \in [0,h]^m} p(\vec{x}) + q(\vec{x})A(\vec{x})$$

where p and q are multinomials of degree h, and A is supposedly a multinomial of degree h held by an oracle. Hence $E(A)$ is a multinomial of degree $2h$. Assume that the oracle is claiming that $E(A) = 0$, and that this claim cannot be true. Then a sum-check protocol over a field $\mathcal{F}$ will cause rejection with probability at least $1 - 2hm/|\mathcal{F}|$, if indeed A is a multinomial of degree h. However, if A is not of degree h (which may the case if no degree-h test is performed), then the oracle can easily cheat. Consider the last stage of the protocol where $\vec{x}$ has been completely instantiated to a random value $\vec{r}$, and it is left to check that $p(\vec{r}) + q(\vec{r})A(\vec{r}) = c$, for some $c \in \mathcal{F}$. As q is a nonzero multinomial, almost surely $q(\vec{r}) \neq 0$. Then

$$A(\vec{r}) = \frac{c - p(\vec{r})}{q(\vec{r})}$$

causes acceptence. For every $\vec{r}$, the oracle can compute in advance what the respective value of c in the sum-check protocol will be (in particular, the oracle can always reply with the zero univariate polynomial in the sum check protocol, forcing $c = 0$ always), and then prepare the table A with the $A(\vec{r})$ that would cause acceptance.

The example above is too simple to capture all properties of the sum-check protocol of subsection 1.3. A major difference is that in subsection 1.3, A is queried in three locations at the end protocol. Viewing it differently, for every specific $\vec{r}$, there are roughly $3|\mathcal{F}|^{2m}$ different executions of the sum-check protocol at the end of which $A(\vec{r})$ is queried. It may well be that for any particular value that the oracle chooses for $A(\vec{r})$, only a small fraction of these executions are accepting.

So how can the oracle cheat? Here is an approach that works if the original 3CNF is of a certain form. A 3CNF may have 8 "types" of clauses, where each type specifies for each of the three variables whether it is positive or negated. Assume that the input 3CNF contains at most $k < 8$ types. (In fact, every 3CNF can be arranged so that it contains at most 4 different types, by ordering the positive variables before the negative ones in every clause. Moreover, 3SAT is NP-complete even over clauses of only two types, either all negative or all positive.) Then the oracle can always reply with the zero polynomial, forcing the final test to be a test of whether the weighted sum over all clauses c of

$$C(A, c) = \Pi_{j=1}^{3} \chi_{j,c}(\vec{r}_j) \cdot (s_{j,c} - A(\vec{r}_j))$$

is 0. Now the idea is for A to prepare a completely random 0/1 table A. Hence $(A(\vec{r}_1), A(\vec{r}_2), A(\vec{r}_j))$ is a random vector $\vec{b} \in \{0,1\}^3$. Now the crucial point is that $C(A, c) = 0$ unless the type of clause c is the bitwise complement of $\vec{b}$. Hence if $\vec{b}$ is the complement of a type missing in the original 3CNF, all the expressions $C(A, c)$ become 0 and the protocol is accepted. Hence the acceptance probability is at least $1 - k/8$, regardless of the size of the finite field $\mathcal{F}$.

Can the oracle cheat with probability substantially higher than $1 - k/8$? In particular, can the oracle cheat with substantial probability even if the 3CNF formula has all eight types of clauses? A negative answer to this question may lead to a substantial simplification in the proof of the PCP theorem.

References

[1] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, 21–31, 1991.

[2] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[3] B. Chor and O. Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5:96–106, 1989.

[4] U. Feige, S. Goldwasser, L. Lovasz, S. Safra and M. Szegedy. “Interactive Proofs and the Hardness of Approximating Cliques”. *Journal of the ACM, Vol. 43, No. 2, March 1996, pp. 268–292.*

[5] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *JACM*, 39 (1992), 859–868.