Introduction to Complexity Theory*
Lecture 18: NP in PCP[poly,O(1)]

Notes taken by Tal Hassner and Yoad Lustig

May 30, 1999

Summary: The major result in this lecture is $\mathcal{NP} \subseteq \mathcal{PCP}(poly, O(1))$. In the course of the proof we introduce an $\mathcal{NPC}$ language "Quadratic Equations", and show it to be in $\mathcal{PCP}(poly, O(1))$, in two stages : first assuming properties of the proof (oracle) and then testing these properties. An intermediate result that might invoke independent interest is an efficient probabilistic algorithm that distinguishes between linear and far-from-linear functions.

1 Introduction

In this lecture we revisit $\mathcal{PCP}$ in hope of giving more flavor of the area by presenting the result $\mathcal{NP} \subseteq \mathcal{PCP}(poly, O(1))$. Recall that last semester we were shown without proof "the $\mathcal{PCP}$ theorem" $\mathcal{NP} = \mathcal{PCP}(log(n), O(1))$ (Corollary 12.3 in Lecture 12). (Clearly $\mathcal{PCP}(log(n), O(1)) \subseteq \mathcal{NP}$ as given only logarithmic randomness the verifier can only use its randomness to choose from a polynomial selection of queries to the oracle, therefor the answers to all the possible queries can be encoded in a polynomial size witness (for a more detailed proof see proposition 3.1 in lecture 12).

Two of the intermediate results on the way to proving the $\mathcal{PCP}$ theorem are $\mathcal{NP} \subseteq \mathcal{PCP}(poly, O(1))$ and $\mathcal{NP} \subseteq \mathcal{PCP}(log, polylog)$. In this lecture we will only see a proof of $\mathcal{NP} \subseteq \mathcal{PCP}(poly, O(1))$.

We recall the definition of a PCP proof system for a language L. A Probabilistically Checkable Proof system for a language L is a probabilistic polynomial-time oracle machine M (called verifier) satisfying:

1. Completeness : For every $x \in L$ there exists an oracle π_x such that : $\Pr[M^{\pi_x}(x) \text{ Accepts}] = 1$.
2. Soundness : For every $x \notin L$ and for every oracle π : $\Pr[M^{\pi}(x) \text{ Accepts}] \leq \frac{1}{2}$.

In addition to limiting the verifier to polynomial time we are interested in two additional complexity measures namely, the *randomness* M uses, and the *number of queries* M may perform. We denote by $\mathcal{PCP}(f(\cdot), g(\cdot))$ the class of languages for which there exists a PCP proof system utilizing at most $f(|x|)$ coin tosses and $g(|x|)$ queries for every input x.

We should note that an easy result is that $\mathcal{PCP}(0, poly) \subseteq \mathcal{NP}$. In $\mathcal{PCP}$ we allow the verifier two advantages over an $\mathcal{NP}$ verifier. The first is probabilistic soundness as opposed to perfect one, (which is not relevant in this case as the verifier is deterministic). The second major advantage is that the $\mathcal{PCP}$ "proof" (oracle) is not limited to polynomial length but is given as an oracle. One may look at the oracle as a proof of exponential length for which the verifier machine has random

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

access (reading a bit from the memory address *addr* equivalent to performing a query about *addr*). However the verifier is limited to polynomial time and therefor actually reads only polynomial number of bits. In case the verifier is completely deterministic the bits it will read are known in advance and therefor there is no need to write down the rest of the bits i.e. a polynomial witness (as in $\mathcal{NP}$) will suffice.

2 Quadratic Equations

Back to our goal of proving $\mathcal{NP} \subseteq \mathcal{PCP}(poly, O(1))$, it is easy to see that for any $L \in \mathcal{NPC}$ proving $L \in \mathcal{PCP}(poly, O(1))$ will suffice, since given any language L_1 in $\mathcal{NP}$ we can decide it by reducing every instance x to it's corresponding instance x_L and using our $\mathcal{PCP}(poly, O(1))$ proof system to decide whether x_L is in L (iff $x \in L_1$).
In this section we introduce an $\mathcal{NPC}$ language that we will find convenient to work with.

Definition 1 (Quadratic Equations): *The language* Quadratic Equations *denoted QE consists of all satisfiable sets of quadratic equations over $GF(2)$.*

Since in GF(2) $x = x^2$ for all x we assume with out loss of generality all the summands are either of degree 2 or constants.
QE formulated as a problem :

Problem: QE
Input: A sequence of quadratic equations $\{\sum_{i,j=1}^{n} c_{i,j}^{(k)} x_i x_j = c^{(k)}\}_{k=1}^{m}$.
Task: Is there an assignment to $x_1 \ldots x_n$ satisfying all equations ?

Clearly QE is in $\mathcal{NP}$ as a satisfying assignment is an easily verifiable witness. To see that it is $\mathcal{NP} - hard$ we will see a reduction from $3 - SAT$.

Given an instance of $3 - SAT$, $\bigwedge_{i=1}^{m} (l_{i1} \vee l_{i2} \vee l_{i3})$ where each l_{ij} is a literal i.e. either an atomic proposition p or it's negation $\neg p$. We will associate with each clause $C_i = (l_{i1} \vee l_{i2} \vee l_{i3})$ (for example $(p_4 \vee \neg p_7 \vee p_9)$) a cubic equation in the following way:

With each atomic proposition p we associate a variable x_p. Now, looking at C_i if l_{ij} is an atomic proposition p then we set the corresponding factor y_{ij} to be $(1 - x_p)$ otherwise l_{ij} is the negation of an atomic proposition $\neg p$ in which case we set the corresponding factor y_{ij} to be x_p . Clearly, the factor y_{ij} equals 0 iff the literal l_{ij} is true, and the expression $y_{i1} y_{i2} y_{i3}$ equals zero iff the disjunction $l_{i1} \vee l_{i2} \vee l_{i3}$ is true (In our example the clause $(p_4 \vee \neg p_7 \vee p_9)$ gets transformed to the equation "$(1 - x_4) x_7 (1 - x_9) = 0$"). Therefor the set of equations $\{y_{i1} y_{i2} y_{i3} = 0\}_{i=1}^{m}$ is satisfiable iff our $3 - CNF$ is satisfiable.

We have reduced $3 - SAT$ to a satisfiability of a set of cubic equations but still have to reduce it to quadratic ones (formally we have to open parenthesis to get our equations to the normal form this can be easily done).

What we need is a way to transform a set of cubic equations into a set of quadratic ones (such that one set is satisfiable iff the other is). The latter can be done as follows:
For each pair of variables x_i, x_j in the cubic system introduce a new variable z_{ij} and a new equation $z_{ij} = x_i x_j$, now go through all the equations and whenever we find a summand of degree 3 $(x_i x_j x_k)$ replace it by a summand of degree 2 $(z_{ij} x_k)$. Our new equation system is composed of two parts, the first part introduces the new variables $(\{z_{ij} = x_i x_j\}_{i,j=1}^{n}$ quadratic in the size of the original

input), and the second part is the transformed set of equations all of degree 2 (linear in the size of the input)
In our example:
$(1-x_4)x_7(1-x_9)=0 \iff x_7 - x_4x_7 - x_7x_9 + x_4x_7x_9 = 0$ may be replaced by:
$x_4x_7 = z_{4\,7}$ and $x_7 - x_4x_7 - x_7x_9 + z_{4\,7}x_9 = 0$.
The last technical step is to go over the equation set and replace every summand of degree 1 (i.e. x_i) by its square (i.e. x_i^2). Since in GF(2) for all a it holds that $a = a^2$ this is purely a technical transformation that does not change in any way the satisfiability of the equations.
Clearly the new set of equations is satisfiable iff the original set is.

Since the entire procedure of reducing a $3-CNF$ to a set of quadratic equations can be done in polynomial time we have reduced $3-SAT$ to QE.

Note that the trick used to reduce the degree of summands in the polynoms can be iterated to reduce the degree of higher degree summands, however the new equations of the kind $z_{ij} = x_ix_j$ are of degree 2, and therefor such a trick cannot be used to reduce the degree of equations to degree 1. In fact degree 1 equations are linear equations for which we know an efficient procedure of deciding satisfiability (Gaussian Elimination). Thus there can be no way of reducing a set of quadratic equations to a set of linear ones in polynomial time, unless $\mathcal{P} = \mathcal{NP}$.

3 The main strategy and a tactical maneuver

The task standing before us is finding a $\mathcal{PCP}(poly, O(1))$ proof system for QE. Intuitively this means finding a way in which someone can present a proof for satisfiability of an equation set (the oracle), which might be very long but one may verify its correctness (at least with high probability) taking only a constant number of "glimpses" at the proof regardless of the size of the equation system.

The existence of such a proof system seems counter-intuitive at first since in the proof systems we are familiar with, any mistake however small in any place of the proof cause the entire proof to be invalid. (However existence of such proof systems is exactly what the "PCP Theorem" asserts.)

In this section we try to develop an intuition of how such a proof system can exist by outlining the main ideas of the proof that $QE \in \mathcal{PCP}(poly, O(1))$. We will deal with the "toy example" of proving that one linear expression can get the value 0 over GF(2) (of course since such a question is decidable in polynomial time there is a trivial proof system in which the oracle gives no information at all (always answers 1), but we will not make use of that triviality).

To develop an intuition we adopt a convention that the proof of validity for an equation system (the oracle in the $\mathcal{PCP}$ setting) is written by an adversary who tries to cheat us into accepting non-satisfiable equations. The goal of our system is to overcome such tries to deceive us (while still enabling the existence of proofs for satisfiable proof systems).

Suppose first that we can restrict our adversary to writing only proofs of certain kinds, i.e. having special properties. For example we may restrict the proofs to encode assignments in the following way :
Fix some encoding of linear expressions in the variables $x_1, \ldots, x_n$ into natural numbers (denote by C(ex) the natural number encoding the expression 'ex'). A reasonable encoding for the linear expression $\sum_{i=1}^{n} \lambda_i x_i$ would be simply the number whose binary expansion is the sequence of bits $\lambda_1\lambda_2\ldots\lambda_n$).

Suppose the adversary is restricted to writing proofs in which he encodes assignments. To encode an assignment $x_1 = a_0, \ldots, x_n = a_n$, the adversary evaluates all the linear expressions over

$x_1, \ldots, , x_n$ with that specific assignment, and writes the value $ex(a_0, \ldots, a_n)$ at the place $C(ex)$ in the proof. In the $\mathcal{PCP}$ setting this means the oracle π answers $ex(a_0, \ldots, a_n)$ on the query $C(ex)$. For example $\pi(C(x_1 + x_3))$ would have the value $a_1 + a_3$. If we can trust the adversary to comply to such a restriction, all we have to do given an expression ex, is to calculate $C(ex)$ and query the oracle π.

The problem of course is that we cannot restrict the adversary. What we can do is check whether the proof given to us is of the kind we assume it is. To do that we will use properties of such proofs. For example in our case we may try to use the linearity of adding linear expressions: for every two expressions e_1, e_2 it holds that $(e_1 + e_2)(a_0, \ldots, a_n) = e_1(a_0, \ldots, a_n) + e_2(a_0, \ldots, a_n)$. Therefor if the proof is of the kind we assume it to be, then the corresponding values in the proof will also respect addition, i.e. the oracle π must satisfy $\pi(C(e_1 + e_2)) = \pi(C(e_1)) + \pi(C(e_2))$.

In general we look for a characteristic of a special 'kind' of proofs. We want that assuming a proof is of that 'kind', one would be able to check its validity using only $O(1)$ queries, and that checking whether a proof is of that special 'kind' will also be feasible using $O(1)$ queries.

The main strategy is to divide the verification of a proof to two parts :

1. Check whether the proof is of a 'good kind' (the oracle answers have a special property), otherwise reject.

2. Assuming that the proof is of the 'good kind', determine its validity, (for example is the proof encoding a satisfying assignment to the equation system)

Up to this point we have developed the general strategy which we intend to use in our proof. Our characteristic of a 'good' proof would be that it encodes an assignment to the variables in a special way. In Section 4 we will define exactly when is a proof of the 'good kind', and see how 'good' proofs' validity can be checked using $O(1)$ queries. In Section 5 we will see how to check whether a proof is of the 'good kind' in $O(1)$ queries.

The bad news is that our strategy as stated above is infeasible if taken literally. We search for a characteristic of proofs that will enable us to distinguish between 'good' proofs and 'bad' proofs in $O(1)$ queries, but suppose we take a 'good' proof and switch just one bit. The probability that such a 'flawed' proof can be distinguished from the original proof in $O(1)$ queries is negligible. (The probability of even querying about that specific bit is the number of queries over the size of the proof, but the size of the proof is all the queries one may pose to an oracle which is exponential in our case).

On the other hand this problem should not be critical, if the adversary changes only one bit, it seems he doesn't have a very good chance of deceiving us, since we probably won't read that bit anyway and therefor it would look to us as if we were given a proof of the 'good' kind.

It seems that even if distinguishing between 'good' proofs and 'bad' ones is infeasible, it may suffice to distinguish between 'good' proofs and proofs which are 'very different' from 'good' proofs. We should try to find a "kind" of proofs for which we can verify that a given proof "looks similar" to. A proof will be "similar" to "good" if with high probability sampling from it will yield the same results as sampling from some really "good proof".

Since we only sample the proof given to us we can only check that the proof behaves "on average" as a proof with the property we want. This poses a problem in the other part of the proof checking process. When we think of it, the adversary does not really have to cheat in many places to deceive us, just in the few critical ones (e.g. the location $C(ex)$ where ex is the expression we try to evaluate). (Remember our adversary is a fictitious all powerful entity - it knows all about our algorithm. Our only edge is that the adversary must decide on a proof before we choose our random moves).

For example in our "toy system" we decide whether the equation is satisfiable based on the single bit $C(ex)$ (which one can calculate from the expression ex) so the adversary only has to change that bit to deceive us. In general during the verification process we do not ask random queries but ones that have an intent, usually this means our queries have a structure i.e. they belong to a small subset of the possible queries. So the adversary can gain a lot by cheating only on that small subset while "on the average" the proof will still look as of the good kind.

Here comes our tactical maneuver: What we would like is to enjoy both worlds, ask random queries and yet get answers to the queries that interest us (which are from a small subset). It turns out that sometimes this can be done, what we need is another property of the proofs. Suppose that we want to evaluate a linear function L on a point x. We can choose a random shift r and evaluate $L(x+r)$ and $L(r)$ we can now evaluate $L(x) = L(x+r) - L(r)$. If r is uniformly distributed so is $x+r$ (also they are dependent), so we have reduced evaluating L at a specific point (i.e. x) into evaluating L at two random points (i.e. r and $x+r$).

In general we want to calculate the value of a function f in a point x by applying f only to random points, we need some method of calculating $f(x)$ from our point x the shift r and the value of the function on the random points $f(x+r)$ and $f(r)$. If on a random point the chance of a mistake is small enough (i.e. p) then the chance of a mistake in one of the two points $x+r, r$ is at most $2p$ which means we get the value on x with probability greater or equal then $1-2p$. If one can apply such a trick to "good" proofs , we can ask the oracle only random queries, that way neutralizing an adversary attempt to cheat us at "important places". This is called *self-correction* since we can "correct" the value of a function (at an important point) using the same function.

4 Testing satisfiability assuming a nice oracle

This section corresponds to the second part of our strategy i.e. we are going to check satisfiability while assuming the oracle answers are nice enough.

The property we are going to assume is as follows:
The oracle π fixes some assignment $x_1 = a_1, \ldots, x_n = a_n$ and when given a sequence of coefficients $\{b_{ij}\}_{i,j=1}^{n}$ answers $\sum_{i,j=1}^{n} b_{ij} a_i a_j$. We assume that the testing phase (to be described in Section 6) ensures us that the oracle π has this property "on the average" i.e. on at most a fraction of 0.01 of the queries we might get an arbitrary answer. The idea is that the oracle encodes a satisfying assignment (if the equations are satisfiable) and that our task is verifying that the assignment encoded by the oracle is indeed satisfying.

Note that we may get the assignment a_i to any specific variable x_i by setting $b_{ii} = 1$ and all the other coefficients $b_{kl} = 0$; But we cannot just find the entire assignment (and evaluate the equations) since that would take a linear number of queries.

We have to check whether the set $\{\sum_{i,j=1}^{n} c_{i,j}^{(k)} x_i x_j = c^{(k)}\}_{k=1}^{m}$ is satisfiable using only a constant number of queries, the tools at our disposal allow us to evaluate any quadratic equation with the assignment the oracle encodes (also we might have to use our self correction trick to ensure the oracle does not cheat on the questions we are likely to ask. The self correction will work since the assignment is fixed and therefor $\sum_{i,j=1}^{n} b_{ij} a_i a_j$ is just a linear function of the b_{ij}s).

The naive approach of checking every equation in our set will not work for the obvious reason that the number of queries using this approach equals the number of equations which might be linear in the size of the input. We must find a way to "check many equations at once". Our trick will be random summations i.e. we will toss a coin for each equation and sum the equations for which the result of the toss is 1. If all the equations are satisfied clearly the sum of the equations will be satisfied (we sum both sides of the equations).

Random summation test

1 Choose a random subset S of the equations $\{\sum_{i,j=1}^{n} c_{i,j}^{(k)} x_i x_j = c^{(k)}\}_{k=1}^{m}$

2 Sum the linear expressions at the left hand side of the equations in S and denote $ex_{rsum} = \sum_{k \in S} (\sum_{i,j=1}^{n} c_{i,j}^{(k)} x_i x_j)$.
After rearranging the summands present ex_{rsum} in normal form.

3 Sum the constants at the right hand side of the equations in S and denote $c_{rsum} = \sum_{k \in S} c^{(k)}$.
(Note that if an assignment $a_1, \ldots, a_n$ is satisfying then $ex_{rsum}(a_1, \ldots, a_n) = c_{rsum}$).

4 Query about ex_{rsum} using self correction technique and compare the answer to c_{rsum}. Accept if they are equal and reject otherwise.

What is the probability of discovering that not all the equations were satisfied ? If only one equation is not satisfied we would discover that with probability $\frac{1}{2}$ since if we include that equation in the sum (set S), the left hand side of the sum is not equal to the right hand side no matter which other equations we sum. But what happens if we have more then one unsatisfied equation?
In this case two wrongs make a right since if we sum two unsatisfied equations the left hand side will be equal to the right hand side (this is the wonderful world of GF(2)).

To see that in any case the probability of discovering the existence of an unsatisfied equation is $\frac{1}{2}$, consider the last toss of coin for an unsatisfied equation (i.e. for all the other unsatisfied equations it was already decided whether to include them in the sum or not). If the sum up to now is satisfied then with probability $\frac{1}{2}$ we will include that equation in the sum and therefor have an unsatisfied sum at the end, and if the sum up to now is not satisfied then with probability $\frac{1}{2}$ we will not include that last equation in the sum and remain with an unsatisfied sum at the end.

We have seen that if an assignment does not satisfy all the equations, then with probability $\frac{1}{2}$ it fails the random sum test. All that is left to completely analyze the test (assuming the oracle was tested and found nice) is a little book keeping.

First we must note that the fact the oracle passed the "oracle test" and found nice, does not assure us that all its answers are reliable, there is 0.01 probability of getting an arbitrary answer. In fact since our queries have a structure (they are all partial sums of the original set of equations), we must use self correction which means asking two queries for every query we really want to ask (of the random shift and of the shifted point of interest). During a random summation test we need to ask only one query (of the random sum), using self correction that becomes two queries (of the shifted point and of the random shift) and therefor we have a probability of 0.02 that our result will be arbitrary. Assuming the answers we got were not arbitrary, the probability of detecting an unsatisfying assignment is 0.5 . Therefor the overall probability of detecting a bad assignment is $0.98 * 0.5 = 0.49$ making the probability of failure 0.51. Applying the test twice and rejecting if one of the trials fail we get failure probability of $0.51^2 < 0.3$.

The result of our "book keeping" is that if we can test that the oracle has our property with failure probability smaller then 0.2, then the probability of the entire procedure to fail is smaller than 0.5, and we satisfy the $\mathcal{PCP}$ satisfiability requirement. (In our case the completeness requirement is easy, just note that an oracle encoding a satisfying assignment always pass).

5 Distinguishing a nice oracle from a very ugly one

In this section our objective is to present a method of testing whether the oracle behaves "typically well". We shall say that an oracle π behaves "well" if it encodes an assignment in the way defined above, and that π behaves "typicly well" if it agrees on more then 99% of the queries with a function that encodes an assignment that way. Our test should detect an oracle that does not behave "typicly well" with probability greater then 0.8 .

As described in the strategy our approach is to search some characterizing properties of the "good oracles". What does a "good oracle" look like ? Assume π is a good oracle corresponding to an assignment $x_1 = a_1, \ldots, x_n = a_n$. We may look at the oracle as a function $\pi : \{0,1\}^{n^2} \to \{0,1\}$ denoting its argument $\underline{b} = (b_{11}, \ldots, b_{1n}, b_{21}, \ldots, b_{nn})^\top$. Then, since $a_1 \ldots, a_n$ are fixed, π is a linear function of $\underline{b}$ i.e. $\pi(\underline{b}) = \sum_{i,j=1}^{n} \lambda_{ij} b_{ij}$. Furthermore π's coefficients λ_{ij} have a special structure - there exists a sequence of constants $\{a_i\}_{i=1}^n$ s.t. $\lambda_{ij} = a_i a_j$. It turns out that these properties characterize the "good oracles", since if π^* is a linear function for which there exists a sequence of constants $\{a_i^*\}_{i=1}^n$ s.t. π^*'s coefficients are $\lambda_{ij}^* = a_i^* a_j^*$, then π^* is encoding the assignment $x_1 = a_1^*, \ldots, x_n = a_n^*$.

Our "oracle test" is composed of two stages: testing the linearity of π and testing that the linearity coefficients λ_{ij}'s have the correct structure.

5.1 Tests of linearity

In order to devise a test and prove its correctness we begin with some formal definitions.

Definition 2 (linearity of a function)
A function $f : \{0,1\}^m \to \{0,1\}$ is called linear *if there exist constants $a_1^f, \ldots, a_m^f$ s.t. for all $\underline{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top \in \{0,1\}^m$ it holds that $f(\underline{\sigma}) = \sum_{i=1}^{m} a_i^f \sigma_i$.*

Claim 5.1 (alternative definition of linearity)
A function $f : \{0,1\}^m \to \{0,1\}$ is linear *iff for every two vectors $\underline{\sigma^1}, \underline{\sigma^2} \in \{0,1\}^m$ and every $\lambda_1, \lambda_2 \in \{0,1\}$ it holds that: $\lambda_1 f(\underline{\sigma^1}) + \lambda_2 f(\underline{\sigma^2}) = f(\lambda_1 \underline{\sigma^1} + \lambda_2 \underline{\sigma^2})$.*

Proof: Suppose first that f is linear by the definition i.e. there exists constants $a_1^f, \ldots, a_m^f$ s.t. for all $\underline{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top$, $f(\underline{\sigma}) = \sum_{i=1}^{m} a_i^f \sigma_i$. Then for every $\underline{\sigma^1}, \underline{\sigma^2}$ it holds that $\lambda_1 f(\underline{\sigma^1}) + \lambda_2 f(\underline{\sigma^2}) = \sum_{i=1}^{m} a_i^f \lambda_1 \sigma_i^1 + \sum_{i=1}^{m} a_i^f \lambda_2 \sigma_i^2 = \sum_{i=1}^{m} a_i^f (\lambda_1 \sigma_i^1 + \lambda_2 \sigma_i^2) = f(\lambda_1 \underline{\sigma^1} + \lambda_2 \underline{\sigma^2})$.

Suppose now that the claim holds i.e. for every two vectors $\sigma^1, \sigma^2 \in \{0,1\}^m$ $\lambda_1 f(\sigma^1) + \lambda_2 f(\sigma^2) = f(\lambda_1 \underline{\sigma^1} + \lambda_2 \underline{\sigma^2})$. Denote by a_i^f the value $f(\underline{e^i})$ where $\underline{e^i}$ is the i^{th} element in the standard basis i.e. all $\underline{e^i}$'s coordinates but the i^{th} are 0 and the i^{th} coordinate is 1.
Every $\underline{\sigma} \in \{0,1\}^m$ can be expressed as $\underline{\sigma} = \sum_{i=1}^{m} \sigma_i \underline{e^i}$.

Then, $f(\underline{\sigma}) = f(\sum_{i=1}^{m} \sigma_i \underline{e^i})$ and by the claim we get :

$f(\sum_{i=1}^{m} \sigma_i \underline{e^i}) \;=\; \sum_{i=1}^{m} \sigma_i f(\underline{e^i}) \;=\; \sum_{i=1}^{m} \sigma_i a_i^f$ ■

Definition 3 (distance from linearity)

Two functions $f, g : \{0,1\}^m \to \{0,1\}$ *are said to be* at distance at least δ *(or* δ far*) if :*
$\Pr_{\sigma \in_R \{0,1\}^m}[f(\sigma) \neq g(\sigma)] \geq \delta$

Two functions $f, g : \{0,1\}^m \to \{0,1\}$ *are said to be* at distance at most δ *(or* δ close*) if :*
$\Pr_{\sigma \in_R \{0,1\}^m}[f(\sigma) \neq g(\sigma)] \leq \delta$

Two functions $f, g : \{0,1\}^m \to \{0,1\}$ *are said to be* at distance δ *if :*
$\Pr_{\sigma \in_R \{0,1\}^m}[f(\sigma) \neq g(\sigma)] = \delta$

A function $f : \{0,1\}^m \to \{0,1\}$ *is said to be* at distance at most δ from linear *if there exists some linear function* $L : \{0,1\}^m \to \{0,1\}$ *s.t.* f *is at distance at most* δ *from* L.
In a similar fashion we define distance at least ϵ from linear *and* distance ϵ from linear.

Notice that since there are only finitely many linear functions $L : \{0,1\}^m \to \{0,1\}$ for every function f there is a closest linear function (not necessarily unique) and the distance from linearity is well defined.

We define now a verification algorithm $A^{(\cdot)}(\cdot)$ that accepts as input a distance parameter ϵ and oracle access to a function f (therefor we actually run $A^f(\epsilon)$), and tries to distinguish between f's which are at distance at least ϵ from linear and linear functions. A iterates a "basic procedure" $T^{(\cdot)}$ that detects functions which are "bad" (ϵ-far from linear) with small constant probability. Using enough iterations ensures the detection of "bad" f's with the desired probability.

<u>Basic procedure T^f:</u>

1 Select at random $\underline{a}, \underline{b} \in_R \{0,1\}^n$.

2 Check whether $f(\underline{a}) + f(\underline{b}) = f(\underline{a} + \underline{b})$, if not reject.

<u>Linearity Tester $A^f(\epsilon)$:</u>

1 Repeat for $\lceil \frac{36}{\epsilon} \rceil$ times the basic procedure $T^{(f)}$

2 Reject if $T^{(f)}$ rejects even once, accept otherwise.

Theorem 4 Linearity testing : *Let* $p_{rej} = \frac{\epsilon}{4}$*. Then if* f *is linear* T^f *always accepts, and if* f *is at distance at least* ϵ *from linear then* $\Pr[T^f rejects] \geq p_{rej}$.

Proof: Clearly if f is linear T will accept. If f is not linear we deal separately with functions close to linear and with functions relatively far from linear.

If f is at distance greater or equal $\frac{3}{8}$ then the probability of rejection is bounded from below by $\frac{1}{8}$. As the proof is rather long and technical it is given in appendix B.

For f close to linear, denote by δ the distance of f from linearity and let L be a linear function at distance δ from f.

Denote by $G = \{\underline{a} | f(\underline{a}) = L(\underline{a})\}$ the set of "good" inputs on which the functions agree. Clearly $|G| = (1-\delta)2^m$. We shall try to bound from below the probability that an iteration of A rejects.

Since the value of a linear function of every two from the three points $\underline{a}, \underline{b}, \underline{a}+\underline{b}$ fixes the value of third, it is easy to see that the algorithm will reject if two of these points are in G and the third is not in G. Therefor the probability that one iteration rejects is greater or equal to $\Pr[\underline{a} \in G, \underline{b} \in G, (\underline{a}+\underline{b}) \notin G] + \Pr[\underline{a} \in G, \underline{b} \notin G, (\underline{a}+\underline{b}) \in G] + \Pr[\underline{a} \notin G, \underline{b} \in G, \underline{a}+\underline{b} \in G]$.

The three events are clearly disjoint. What might be less obvious is that they are symmetric. It is easy to see $\Pr[\underline{a} \notin G, \underline{b} \in G, (\underline{a}+\underline{b}) \in G] = \Pr[\underline{a} \in G, \underline{b} \notin G, (\underline{a}+\underline{b}) \in G]$. Notice also that instead of choosing $\underline{b}$ at random we may choose $(\underline{a}+\underline{b})$ at random and then $\underline{a} + (\underline{a}+\underline{b}) = \underline{b}$, so the third event is also symmetric and therefor $\Pr[\underline{a} \in G, \underline{b} \in G, (\underline{a}+\underline{b}) \notin G] = \Pr[\underline{a} \notin G, \underline{b} \in G, (\underline{a}+\underline{b}) \in G]$.

Thus the probability of rejection is greater or equal to $3\Pr[\underline{a} \notin G, \underline{b} \in G, (\underline{a}+\underline{b}) \in G]$.
The latter can be presented as $3\Pr[\underline{a} \notin G] \cdot \Pr[\underline{b} \in G, (\underline{a}+\underline{b}) \in G | \underline{a} \notin G]$. By definition of δ it holds that $\Pr[\underline{a} \notin G] = \delta$. It is also clear that
$\Pr[\underline{b} \in G, (\underline{a}+\underline{b}) \in G | \underline{a} \notin G] = 1 - \Pr[\underline{b} \notin G \text{ or } (\underline{a}+\underline{b}) \notin G | \underline{a} \notin G]$.
Therefor the probability of rejection is greater or equal to $3\delta(1 - \Pr[\underline{b} \notin G \text{ or } (\underline{a}+\underline{b}) \notin G | \underline{a} \notin G])$. Using the symmetry of $\underline{b}$ and $\underline{a}+\underline{b}$ explained above and union bound the probability of rejection is greater or equal to $3\delta(1 - 2\Pr[\underline{b} \notin G | \underline{a} \notin G])$. Since $\underline{a}$ and $\underline{b}$ were chosen independently the probability of rejection is greater or equal to $3\delta(1 - 2\Pr[\underline{b} \notin G]) = 3\delta(1-2\delta)$.

Notice That analysis above is good for small δ i.e. for functions which are quite close to linear functions. However the function $3\delta(1-\delta)$ drops to 0 at $\frac{1}{2}$, therefor the analysis is not good enough for functions which are quite far from linear functions. The obvious reason for the failure of the analysis is that if the function is far enough from a linear function then the probability that two of the three points will fall inside the "good" set of points (whose f value is identical to their value under the closest linear function) is small.

Combining both cases, of functions relatively close to linear and functions relatively far from linear we get the desired result. That is, let δ be the distance of f from linear. Note that $\delta \le \frac{1}{2}$, (since for every function f the expected distance of f from a random linear function is $\frac{1}{2}$). In case $\delta > \frac{3}{8}$ we have a rejection probability of at least $\frac{1}{8} \ge \frac{\delta}{4}$. Otherwise, $\delta \le \frac{3}{8}$, and we have a rejection probability of at least $3\delta(1-2\delta) > 3\delta(1 - 2\cdot\frac{3}{8}) = \frac{3}{4}\delta$. ■

Corollary 5 *If f is linear then the verification algorithm $A^{(f)}(\epsilon)$ always accepts it. If f is ϵ far from linear then $A^{(f)}(\epsilon)$ rejects it with probability larger then 0.99*

Proof: If f is linear it always passes the basic procedure T. Suppose f is ϵ far from linear then by Theorem 4 the probability that one iteration of the basic procedure T^f rejects is bigger then $\frac{\epsilon}{4}$. Therefor the probability that f passes all the iterations A invokes is smaller then $(1 - \frac{3}{4}\delta)^{36\cdot\frac{1}{\epsilon}}$. By the inequality $1 - x \le e^{-x}$ the probability that A accepts is smaller then $e^{-\frac{3}{4}\epsilon\cdot\frac{36}{\epsilon}} = e^{-9} < 0.01$.

(To prove the $1 - x \le e^{-x}$ inequality notice equality holds for 0, and differentiate both sides to see that the right hand side drops slower then the left hand side). ■

5.2 Assuming linear π testing π's coefficients structure

If π passes the linearity tester we assume that there exists a linear function $L_\pi : \{0,1\}^{n^2} \to \{0,1\}$ at distance smaller or equal 0.01 from π. For the rest of the discussion L_π stands for some such fixed linear function.

For π to be nice it is not enough that it is close to a linear L_π, but L_π must also be of the special structure that encodes an assignment. We saw that this means that there exists a sequence $\{a_i\}_{i=1}^n$ s.t. $L_\pi(\underline{b}) = \sum_{i,j=1}^{n} a_i a_j b_{ij}$ in other words L_π's linearity coefficients $\{\lambda_{ij}\}_{i,j=1}^n$ have the special structure which is $\lambda_{ij} = a_i a_j$.

L_π is a linear function from $\{0,1\}^{n^2}$ to $\{0,1\}$ and therefor its natural representation is as a row vector of length n^2 ($1 \times n^2$ matrix). However if we rearrange L_π's n^2 coefficients to an $n \times n$ matrix form the constraint on L_π can be phrased in a very elegant form, namely there exists a vector $\underline{a} = (a_1, \ldots, a_n)^\top$ s.t. $(\lambda_{ij}) = \underline{a}\,\underline{a}^\top$. (Notice this is not a scalar product but an outer product and the result is an $n \times n$ matrix). Notice that $(\underline{a}\,\underline{a}^\top)_{ij} = a_i a_j$ which is exactly what we want of λ_{ij}.

So what has to be checked is that (λ_{ij}) really has this structure i.e. there exists a vector $\underline{a}$ s.t. $(\lambda_{ij}) = \underline{a}\,\underline{a}^\top$. How can this be done ?

Notice that if (λ_{ij}) has this special structure then the vector $\underline{a}$ is exactly the diagonal of the matrix (λ_{ij}). This is the case since $\lambda_{ii} = a_i a_i = a_i^2$ however in GF(2) $x^2 = x$ for every x, leading us to $\lambda_{ii} = a_i$.

The last observation means that we can find out every coefficient a_i in $\underline{a}$ simply by querying with $b_{ii} = 1$ and the rest of the $b_{kl} = 0$ (we will have to use self correction as always). This seems to lead us to a reasonable test. First obtain $\underline{a}$ by querying its coefficients a_i, then construct $\underline{a}\,\underline{a}^\top$. What has to be done now is to check whether (λ_{ij}) is indeed equal to the matrix we have constructed $\underline{a}\,\underline{a}^\top$. A natural approach for checking this equality will be sampling: we have access to (λ_{ij}) as a function since we can query π, we can try to simulate $\underline{a}\,\underline{a}^\top$ as a function and compare their results on random inputs. The problem with this natural approach is that querying about every coefficient a_i would cost a linear number of queries and we can only afford a constant number of queries.

It seems we will have to get by without explicitly constructing the entire $\underline{a}$. As the "next best thing" we may try to "capture" $\underline{a}$ by a random summation of its coefficients. A random sum of $\underline{a}$ coefficients defined by a random string $\underline{r} \in_R \{0,1\}^n$ is the sum of the coefficients a_i for which $r_i = 1$ i.e. $\sum_{i=1}^{n} r_i a_i$. This is the scalar product of $\underline{a}$ with $\underline{r}$ denoted $< \underline{a}, \underline{r} >$. The random sum is of course a function of the random string $\underline{r}$ so we introduce the notation $A(\cdot) =< \underline{a}, \cdot >$ where $A(\cdot)$ is the random sum as a function of $\underline{r}$.

Just as we can find any coefficient a_i of $\underline{a}$ by querying π with the appropriate bit on the diagonal b_{ii} turned on, we can find the result of a sum of coefficients. To get the result of a sum of the coefficients $i_1, \ldots, i_k$ all we have to do is query with $b_{i_1 i_1} = 1, \ldots, b_{i_k i_k} = 1$ and all the other bits 0. The result will be $\sum_{l=1}^{k} x_{i_l}^2 = \sum_{l=1}^{k} x_{i_l}$ which is what we want. (As always we have to use self correction).

How is all this going to help us to compare the matrices $\underline{a}\,\underline{a}^\top$ to (λ_{ij}) ?
Most of us immediately identify an $n \times n$ matrix M with its corresponding linear function from $\{0,1\}^n$ to $\{0,1\}^n$. However such a matrix can also stand for a bilinear function
$f_M : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, (where $f_M(\underline{x}, \underline{y}) = \underline{x}^\top M \underline{y}$).

For matrices of the form $\underline{a}\,\underline{a}^\top$ the operation becomes very simple :
$f_{(\underline{a}\,\underline{a}^\top)}(\underline{x}, \underline{y}) = \underline{x}^\top(\underline{a}\,\underline{a}^\top)\underline{y} = (\underline{x}^\top \underline{a})(\underline{a}^\top \underline{y}) =< \underline{x}, \underline{a} >< \underline{a}, \underline{y} >= A(\underline{x})A(\underline{y})$
(The same result can also be developed in the more technical way of opening the summations).

Our access to $A(\cdot)$ enables us to evaluate the bilinear function represented by $\underline{a}\,\underline{a}^\top$. We can also evaluate the bilinear function represented by (λ_{ij}) since :

$f_{(\lambda_{ij})}(\underline{x}, \underline{y}) = x^\top(\lambda_{ij})y = \sum_{i,j=1}^{n} \lambda_{ij} x_i y_j$.

So in order to evaluate $f_{(\lambda_{ij})}(\underline{x}, \underline{y})$ we only have to feed $b_{ij} = x_i y_j$ into π. Once again we will use our self correcting technique, this time the structure of the queries does not stand out as in the the case of $A(\cdot)$. Nonetheless, the distribution of queries is not uniform (for example there is a skew towards 0s as it is enough that one of the coordinates x_i or y_j is 0 so that b_{ij} will be 0).

It seems reasonable to test whether $(\lambda_{ij}) = \underline{a}\,\underline{a}^\top$ by testing whether the bilinear functions represented by the two matrices are the same, and do that by sampling. The idea is to sample random vectors $\underline{x}, \underline{y} \in \{0,1\}^n$ and check if the functions agree on their value.

Structure test for (λ_{ij}):

1. Select at random $\underline{x}, \underline{y} \in_R \{0,1\}^n$.
2. Evaluate $A(\underline{x})$ and $A(\underline{y})$ by querying with coefficients corresponding to a matrix U where $\underline{x}$ (resp. $\underline{y}$) is the diagonal i.e. $U_{ii} = x_i$ and the rest of the coefficients are 0s. (The queries should be presented using self correction).
3. Compute $f_{(\underline{a}\,\underline{a}^\top)}(\underline{x}, \underline{y}) = A(\underline{x}) \cdot A(\underline{y})$
4. Query $f_{(\lambda_{ij})}(\underline{x}, \underline{y})$ by querying π with $\{x_i y_j\}_{i,j=1}^{n}$ as the query bits. (Again self correction must be used).
5. Accept if $f_{(\underline{a}\,\underline{a}^\top)}(\underline{x}, \underline{y}) = f_{(\lambda_{ij})}(\underline{x}, \underline{y})$, Reject otherwise.

We are left to prove that if the matrices (λ_{ij}) and $\underline{a}\,\underline{a}^\top$ differ then if we sample we will get different results of the bilinear functions with reasonably high probability.

Given two different matrices M, N we want to bound from below the probability that for two random vectors $\underline{x}, \underline{y}$ the bilinear functions $f_M(\underline{x}, \underline{y})$ and $f_N(\underline{x}, \underline{y})$ agree.

The question is when $\underline{x}^\top M \underline{y} = \underline{x}^\top N \underline{y}$?

Clearly this is equivalent to $\underline{x}^\top (M - N)\underline{y} = 0$.

Suppose we choose $\underline{y}$ first, if $(M-N)\underline{y} = \underline{0}$ then whatever $\underline{x}$ we choose the result $\underline{x}^\top \underline{0}$ will always be 0 and does not depend on $\underline{x}$. If on the other hand $(M-N)\underline{y} \neq \underline{0}$ then it might be the case that $\underline{x}^\top((M-N)\underline{y}) = 0$ depending on the choice of $\underline{x}$. We will analyze the probabilities for these two events separately.

To analyze the probability of choosing a $\underline{y}$ s.t. $(M-N)\underline{y} = \underline{0}$, denote the column dimension of $(M-N)$ by d. There exist d linearly independent columns $i_1, \ldots, i_d$ of $(M-N)$ assume without loss of generality these are the last d columns $n-d+1, \ldots, n$. Also assume the choice of $\underline{y}$ is made by tossing coins for its coordinates' values one by one and that the coins for the last coordinates $n-d+1, \ldots, n$ are tossed last. Lets look at the situation just before the last d coins are tossed (the rest of the coins have already been chosen).

The value of $(M-N)\underline{y}$ will be the sum of columns corresponding to coordinates of value 1, however at our stage not all the coordinates values have been chosen. At this stage we may look of at the last d columns' "contribution" to the final sum $((M-N)\underline{y})$ as a random variable. Denote by $(M-N)_{\downarrow j}$ the j^{th} column in $(M-N)$ and by $\underline{v_{rand}}$ the random variable $\sum_{k=n-d+1}^{n} y_k \cdot (M-N)_{\downarrow k}$. (The sum of columns corresponding to coordinates of value 1 out of the last d coordinates). The rest of the coordinates "contribution" has already been set. Denote by $\underline{v_{set}}$ the vector $\sum_{k=1}^{n-d} y_k \cdot (M-N)_{\downarrow k}$ (The contribution of the rest of the coordinates).

Clearly $(M-N)\underline{y} = \underline{v_{set}} + \underline{v_{rand}}$ i.e. $(M-N)\underline{y} = \underline{0}$ iff $\underline{v_{set}} = -\underline{v_{rand}}$. The question is can this happen and at what probability ? First note that this can happen - otherwise $\underline{v_{set}}$ is independent of columns $i_1, \ldots, i_d$ which means the columns dimension is bigger then d. Second notice that since columns $i_1, \ldots, i_d$ are independent there is only one linear combination of them that equals $\underline{v_{set}}$. This means that there is only one result of the coin tosses for coordinates $i_1, \ldots, i_d$ that will make $\underline{v_{rand}}$ equal $\underline{v_{set}}$, i.e. the probability is $2^{-d} = 2^{-dim(M-N)}$.

Assuming $(M-N)\underline{y} \neq \underline{0}$ what is the probability that $\underline{x}^\top((M-N)\underline{y}) = 0$? The question is given a fixed vector that is not $\underline{0}$ what is the probability that its inner product with a randomly chosen vector is 0 ? By exactly the same reasoning used above (and in the random sum argument in Section 4), the probability is $\frac{1}{2}$. (Toss coins for the random vector coordinates one by one and look at the coin tossed for a coordinate which is not 0 in $(M-N)\underline{y}$, there is always a result of the coin flip that would bring the inner product to 0).

Overall the two vectors have the same image if either $(M-N)\underline{y} = 0$ (with probability bounded by $\frac{1}{2}$) or if $\underline{x}^\top((M-N)\underline{y}) = 0$ (with probability $\frac{1}{2}$). Therefor the probability that the two vectors differ is at least $\frac{1}{4}$.

To finish the probability "book keeping" we must take into account the probability of getting wrong answers from the oracle. The test itself has a failure probability of 0.75 . The test uses three queries ($A(\underline{x}), A(\underline{y})$ and $f_{(\lambda_{ij})}(\underline{x}, \underline{y})$) but using self correction each of those becomes two queries. We get a probability of 0.06 of getting an arbitrary answer leading us to probability of 0.81 for the test to fail. Repeating the test 10 independent times we get a failure probability of $0.81^{10} \leq \frac{1}{8}$ and the probability of failure in either the linearity testing or structure test is bounded by $0.125 + 0.01 \leq 0.15$.

The goal of detecting "bad oracles" with probability greater than 0.8 is achieved.

5.3 Gluing it all together

Basicly our verification of a proof consists of three stages:

1. linearity testing : a phase in which if the oracle π is 0.01 far from linear we have a chance of 0.99 to catch it (and reject).
   To implement this stage we use the linearity tester $A^\pi(0.01)$. (Distance parameter set to 0.01).

2. structure test : assuming the oracle π is close to linear we test that it encodes an assignment using the "structure test". To boost up probability we repeat the test 10 independent times. At the end of this stage we detect bad oracles with probability greater than 0.8

3. satisfiability test : assuming "good oracle" we use the "random summation" test to verify that the assignment encoded by the oracle is indeed satisfying. (This test is repeated twice).

6 Appendix A: Linear functions are far apart

In this section we intend to prove that linear functions are far apart. In addition to the natural interest such a result may invoke, we hope the result may shed some light as to why linear functions are good candidates for $\mathcal{PCP}$ proof systems.

When looking for a $\mathcal{PCP}$ proof system it is clear that the proofs must be robust in the sense that given a good proof π a small change effecting a constant number of bits must yield another good proof π_1, this is so since the probability of the verifier's to detect such a small change is negligible. This means that π_1 must carry "the same information" as π (at least with respect to the information the verifier can "read" from it). This formulation has a very strong scent of error correcting codes, we do know that error correcting codes have this property. In the case of a $\mathcal{PCP}$ proof system for a language in $\mathcal{NP}$ we even have a natural candidate for the information to be coded namely the witness.

One should note that our discussion is just a plausibility argument as there are major differences between a $\mathcal{PCP}$ verifier and an error correcting decoder. On one hand the $\mathcal{PCP}$ verifier does not have to decode at all. The verifier's task is to be convinced that there exists an information word with some desired properties (the witness). On the other hand the verifier is significantly weaker (computationally) then an efficient error correcting decoder, since it may only look at part of the code word (proof).

If the reader is convinced that the error correcting approach is a good one to begin with, note that in addition to being an error correcting code linear functions have another desired property : we can use self correction in a natural way since $L(x) = L(x+r) - L(r)$.

Proposition 6.1 *If $f, g : \{0,1\}^m \to \{0,1\}$ are both linear and $f \neq g$ then the distance of f from g is exactly $\frac{1}{2}$*

Proof: Note that $(f-g)$ is also linear, clearly $f(\underline{x}) \neq g(\underline{x})$ iff $(f-g)(\underline{x}) \neq 0$. All we have left to prove is that for every linear function $h \neq 0$ it holds that $\Pr_{\underline{x}}[h(\underline{x}) \neq 0] = \frac{1}{2}$.
Denote $h(\underline{x}) = \sum_{i=1}^{m} a_i^h x_i$. Since $h \neq 0$ there exists an a_i^h that does not equal 0. Assume the bits of $\underline{x}$ are chosen one by one (x_1 is chosen before x_2 and so on), denote by l the last i for which a_i^h is not 0 (i.e. $a_l = 1$ and for all $j > l$ $a_j = 0$). Clearly $h(\underline{x}) = (\sum_{i=1}^{l-1} a_i^h x_i) + x_l$. If $\sum_{i=1}^{l-1} a_i^h x_i = 0$ then with probability $\frac{1}{2}$ we choose $x_l = 1$ and we get $h(\underline{x}) = 1$. If on the other hand $\sum_{i=1}^{l-1} a_i^h x_i = 1$ then with probability $\frac{1}{2}$ we choose $x_l = 0$ and again we get $h(\underline{x}) = 1$. ■

Corollary 6 *If a function $f : \{0,1\}^m \to \{0,1\}$ is at distance less than $\frac{1}{4}$ from linear then there exists a* unique *linear function L_f s.t. f is at distance less then $\frac{1}{4}$ from L_f.*

Proof: Otherwise by triangle inequality we would get two linear functions at distance smaller than $\frac{1}{2}$ ■

7 Appendix B: The linearity test for functions far from linear

Recall that our objective is to bound the probability of failure of one iteration of the basic procedure T in case of functions far from linear. The result we need is that given a function $f : \{0,1\}^m \to \{0,1\}$ which is at distance at least $\frac{3}{8}$ from linear the probability that $f(x+y) \neq f(x) + f(y)$ for randomly chosen x, y is bigger than some constant c. Denote by η the probability of failing one iteration of T (i.e. $\eta \stackrel{\text{def}}{=} \Pr_{x,y}[f(x+y) \neq f(x) + f(y)]$).

Intuitively if the probability of choosing x and y s.t. $f(x) + f(y) = f(x+y)$, is very big than there must be some linear function L which agree with f on "a lot" of points so f cannot be "too far" from linear. The problem is that it is not clear how to find this L or even those points on which L and f agree.

Since a linear function behavior on the entire space $\{0,1\}^n$ is fixed by its behavior on any n independent vectors, we should somehow see how "most of the points would like f to behave".

Formalizing this intuition is not as hard as it may seem. Given a point a we would like to find how "most of the points want f to behave on a", the natural way to define that, is that the point x "would like" $f(a)$ to take a value that would make $f(a) = f(x+a) - f(x)$ true.

Definition 7 ($L_f(\cdot)$):
For every $a \in \{0,1\}^n$ define $L_f(a)$ to be either 0 or 1 s.t. $\Pr_{x \in_R \{0,1\}^n}[L_f(a) = f(x+a) - f(x)] \geq \frac{1}{2}$ *(In cases were $L_f(a)$ might be either 0 or 1 (probability $\frac{1}{2}$) define $L_f(a)$ arbitrarily.)*

What we would like to see now is that L_f is indeed linear, and that L_f is reasonably close to f (depending on η the probability to fail the basic procedure T). We would indeed prove those claims but before embarking on that task we have to prove a technical result.

By definition $\Pr_{x,a}[f(x) + f(x+a) = L_f(a)] \geq 1 - \eta$ therefor by an averaging argument $L_f(a)$ behaves nice "on the average a" (i.e. for most a's $\Pr_x[f(x)+f(x+a) = L_f(a)] \geq 1-2\eta$). However there might have been a few bad points b on which $\Pr_x[f(x) + f(x+b) = L_f(b)] = \frac{1}{2}$. We would like to show L_f behaves nicely on all the points.

Claim 7.1 *For all points $a \in \{0,1\}^n$ it holds that:* $\Pr_{x \in_R \{0,1\}^n}[f(x) + f(x+a) = L_f(a)] \geq 1 - 2\eta$.

Proof: Look at two points x, y chosen at random, what is the probability that "they vote the same for $L_f(a)$" i.e. $f(x) + f(x+a) = f(y) + f(y+a)$?
Clearly if we denote $p = \Pr_x[f(x) + f(x+a) = L_f(a)]$ then
$\Pr_{x,y}[f(x) + f(x+a) = f(y) + f(y+a)] = p^2 + (1-p)^2$ (x and y might both go with $L_f(a)$ or against it). From Another perspective:

$$
\begin{aligned}
&\Pr_{x,y}[f(x) + f(x+a) = f(y) + f(y+a)] = \\
&= \Pr_{x,y}[f(x) + f(x+a) + f(y) + f(y+a) = 0] = \\
&= \Pr_{x,y}[f(x) + f(y+a) + f(x+y+a) + f(y) + f(x+a) + f(x+y+a) = 0] \leq \\
&\leq \Pr_{x,y}[(f(x) + f(y+a) + f(x+y+a) = 0) \wedge (f(y) + f(x+a) + f(x+y+a) = 0)] = \\
&= 1 - \Pr_{x,y}[(f(x) + f(y+a) + f(x+y+a) \neq 0) \vee (f(y) + f(x+a) + f(x+y+a) \neq 0)] \leq \\
&\leq 1 - 2\eta
\end{aligned}
$$

The last inequality is true since all $x, y, a+x, a+y$ are uniformly distributed (also dependent), from the definition of η and using the union bound. We got that $p^2 + (1-p)^2 \geq 1 - 2\eta$. Simple manipulation brings us to :
$1 - 2p + 2p^2 \geq 1 - 2\eta \iff p(p-1) \geq -\eta \iff p(1-p) \leq \eta$.
Note that since $L_f(a)$ value (0 or 1) was defined to maximize $p = \Pr[f(x) + f(x+a) = L_f(a)]$, p is always bigger or equal to $\frac{1}{2}$. So $\frac{1}{2}(1-p) \leq p(1-p) \leq \eta$ and from that $p \geq 1 - 2\eta$ follows. ∎

Claim 7.2 *If the failure probability η is smaller then $\frac{1}{6}$ then the function $L_f(\cdot)$ is linear.*

Proof: Given any $a, b \in \{0,1\}^n$ we have to prove $L_f(a+b) = L_f(a) + L_f(b)$. The strategy here is simple, we will prove (by the probabilistic method) the existence of "good intermediate" points in the sense that L_f behaves nicely on these points and this forces L_f to be linear.
Suppose there exists x, y s.t. the following events happen simultaneously :

E1 : $L_f(a+b) = f(a+b+x+y) + f(x+y)$.
E2 : $L_f(b) = f(a+b+x+y) + f(a+x+y)$.
E3 : $L_f(a) = f(a+x+y) + f(x+y)$.

Then :

$$L_f(a+b) = f(a+b+x+y) + f(x+y) = L_f(b) + f(a+x+y) + f(x+y) = L_f(b) + L_f(a)$$

.

To prove the existence of these points choose x, y at random. By Claim 7.1 the probability that each of the events E1, E2, E3 will not happen is smaller or equal 2η, so the probability that one of these will not happen is smaller or equal to 6η. Since $\eta < \frac{1}{6}$ the probability that some of those events do not happen is smaller then 1, i.e. there exists x and y for which the events E1, E2, E3 happen and therefor L_f is linear with regard to a, b. ■

Claim 7.3 *The function $L_f(\cdot)$ is at most at distance 3η from f.*

Proof: Basicly we show that since L_f is defined as "how f should behave if it wants to be linear" then if $f(a) \neq L_f(a)$ then "f is not linear on a" since f is close to linear it must be close to L_f.

Denote by p the probability that f agrees with L_f i.e. $\Pr_x[f(x) = L_f(x)] = p$. (Notice that the distance between f and L_f equals $1-p$ by definition).

Choose two vectors x, y and denote by E the event "$f(x+y) = L_f(x+y)$" and by F the event "$f(x) + f(y) = f(x+y)$".
$\Pr_{x,y}[F \wedge E^c] = \Pr_{x,y}[(f(x)+f(y) = f(x+y)) \wedge (f(x+y) \neq L_f(x+y))] =$
$= \Pr_{x,y}[(f(x+y+y)+f(y) = f(x+y)) \wedge (f(x+y) \neq L_f(x+y))]$

Clearly whenever $f(x+y+y)+f(y) = f(x+y)$ and $f(x+y) \neq L_f(x+y)$ it holds that $f(x+y+y)+f(y) \neq L_f(x+y)$ therefor :
$\Pr_{x,y}[F \wedge E^c] \leq \Pr_{x,y}[f((x+y)+y)+f(y) \neq L_f(x+y)] \leq 2\eta$
Where the last inequality is by claim 7.1

Now notice $\Pr[F] = (1-\eta)$ and $\Pr[E] = p$ by definition.
Looking at $\Pr[F]$ from another perspective:

$$\Pr[F] = \Pr[F \wedge E] + \Pr[F \wedge E^c] \leq \Pr[E] + \Pr[F \wedge E^c] \leq p + 2\eta$$

Comparing the two evaluations of $\Pr[F]$ we get $1-\eta \leq p+2\eta$ i.e. $p \geq 1-3\eta$. Since the distance between f and L_f is $1-p$ our result is that f's distance from L_f is smaller or equal to 3η ■

To conclude if η, the probability to fail T, is smaller then $\frac{1}{8}$ then f is at distance at most $\frac{3}{8}$ from linear and this means :

Corollary 8 *If f is at distance bigger then $\frac{3}{8}$ from linear then the probability to fail one iteration of the basic procedure T is bigger or equal to $\frac{1}{8}$.*