Interactive proofs were introduced as a generalization of the classical notion of a proof in the hope that more things can be proven in an interactive way (for instance, the non-isomorphism of two graphs). Today we look at a different kind called a *probabilistically checkable proof* (or *holographic proof*), which is seemingly less general than a classical proof. Again, we start from the definition of NP: A decision problem L is in NP if there is a polynomial-time verifier V and a polynomial p such that

$$\begin{aligned} &\text{if } x \in L, && \text{then there is a } \pi, |\pi| \leq p(|x|) \text{ such that } V(x,\pi) = 1, \text{ and} \\ &\text{if } x \notin L, && \text{then for all } y, |\pi| \leq p(|x|), V(x,\pi) = 0. \end{aligned}$$

Now instead of thinking of π as a witness or certificate for x, we will think of it as a *proof*. In interactive proofs, we thought of π as provided by another (computationally unbounded) party called the prover. Today we will forget about the prover and focus on π (the proof) itself. What we will focus on is how π is accessed by the verifier V.

1 Probabilistically checkable proofs

In the definition of NP, π is provided on some tape of the Turing Machine, which V reads from left to right. What happens if instead V has *random access* to π, that is, V can provide the value i and observe the ith bit of π? Up to a polynomial factor in the running time, nothing much will change because random access can be simulated easily by sequential access, and vice versa.

We now make the radical restriction that instead of reading the whole proof π, V only gets to look at a small portion of π. Say if π is m bits long, we allow V to only look at, say, $\log m$ bits within π. In the case when V is deterministic, this is not terribly interesting: If V gets to see only $\log m$ bits of the proof, then it doesn't need the proof to begin with, because it can try all possible sequences of $2^{O(\log m)}$ values in the locations of the proof it is supposed to read and see if any of those combinations would accept. So a deterministic verifier that has random access to $\log m$ locations inside the proof is no more powerful than a polynomial-time Turing Machine.

Things change substantially when the verifier gets access to randomness! Intuitively, the reason is the following: Even if for any fixing of the randomness, the verifier gets to see only a *constant* number of bits within π, it is entirely plausible that as the randomness changes different parts of π are exposed, so when considering all choices of the randomness the verifier gets to see the whole proof. But in any specific run of the verifier, only a tiny portion of the proof is revealed. This is where the name "holographic proofs" comes from. (Unfortunately for some reason the much drier "probablilstically checkable proof" is now the standard terminology.)

To model random access to a proof, we view the proof as an oracle. So reading the ith bit from the proof amounts to querying the oracle for its contents at location i and reading off the answer from the answer tape. It turns out that the following two quantities are particularly relevant in this context:

- **Randomness complexity:** The number of random bits (i.e., the portion of the randomness tape) the verifier uses on an input of length n
- **Query complexity:** The maximum number of locations in the proof the verifier queries on an input of length n (for any setting of the randomness tape)

We are now equipped to give the definition of a probabilistically checkable proof.

Definition 1. Let L be a decision problem. A *probabilistically checkable proof system (PCP)* for L is a polynomial-time randomized non-adaptive oracle Turing Machine V (called *the verifier*) such that

$$\begin{aligned} &\text{If } x \in L, \qquad \Pr[\text{there exists } \pi \in \{0,1\}^* \text{ s.t. } V^\pi(x) \text{ accepts}] = 1, \\ &\text{If } x \notin L, \qquad \Pr[\text{there exists } \pi \in \{0,1\}^* \text{ s.t. } V^\pi(x) \text{ accepts}] \leq 1/2. \end{aligned}$$

The class $\mathrm{PCP}(r(n), q(n))$ consists of those decision problems that have probabilistically checkable proofs with verifiers of randomness complexity $r(n)$ and query complexity $q(n)$.

By *non-adaptive* we mean that the verifier V makes all its queries into π at once; that is, for every i, the location of the ith query cannot depend on the answer to the previous queries.

Notice that there is no restriction of the length of π; this is because it is implicitly determined by the randomness complexity and query complexity. Over all possible choices of the randomness, the verifier can access at most $q(n) \cdot 2^{r(n)}$ bits in π, so this is the effective length of π.

The most useful setting of parameters is to set the randomness complexity $r(n)$ to $O(\log n)$ and the query complexity $q(n)$ to a constant. This gives rise to the class $\mathrm{PCP}(O(\log n), O(1))$, which is called just PCP. In this setting, the effective length of the proof becomes polynomial in the length of the input, and a randomized verifier can be simulated by a determnistic one that tries all possible choices of the randomness, and accepts iff all of them yield accepting computations. The new verifier works just like a regular NP verifier, thus $\mathrm{PCP} \subseteq \mathrm{NP}$.

What about the opposite direction? Let's look at an example, say SAT. A PCP for SAT would be a procedure that, on input ϕ, gets oracle access to a purported (polynomial-length) proof that ϕ is satisfiable, gets to examine this proof at a *constant* number of places, and based on this has to say whether ϕ is satisfiable or not. We expect it to be correct most of the time. Can this be possible?

One of the most impressive theorems in the theory of computation says exactly that:

Theorem 2 (The PCP Theorem)**.** $\mathrm{PCP} = \mathrm{NP}$.

What this says is that any standard NP proof can be rewritten as a holographic proof. Take your favorite proofs from mathematics. Some of them are hundreds of pages long. If we write this proof in a special PCP form, it is possible to verify its authenticity with confidence, say, 50% just by looking at a constant number of symbols inside the proof! In fact even this constant is very reasonable: examining 4 locations suffices. If we want to have confidence say 99% we just repeat the random verification process about 7 times.

The PCP Theorem was proved in 1993 by Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy, who built upon a breakthrough of Arora and Muli Safra from 1992. In 2005 Irit Dinur came up with a completely different proof.

2 Hardness of approximation

Probabilistically checkable proofs for NP are intimately related to *approximation algorithms* for optimization problems in NP. Let's take 3SAT as an example. Suppose you are given a satisfiable 3CNF ϕ. Since 3SAT is NP-complete, looking for a satisfying assignment of ϕ may be asking too much. But suppose you are willing to settle for less; say you are happy with an assignment that satisfies a $1-\varepsilon$ fraction of all the clauses. (Think of ε as some fixed constant like 1%. This looks like an easier task; is there a polynomial-time algorithm for it?

Here is another task which is seemingly even easier. Instead of solving the above *approximate search problem* which asks you to find a good assignment, now you are only interested in the following promise problem:

Promise problem ε-GAP-3SAT
Input: A boolean formula ϕ.
Yes instances: Some assignment satisfies all clauses of ϕ.
No instances: Every assignment satisfies fewer than a $(1-\varepsilon)$-fraction of all the clauses in ϕ.

Clearly if you can *find* an assignment that satisfies at least $1-\varepsilon$ fraction of the clauses of every satisfiable boolean formula, you can also distinguish between the yes-instances and no-instances of ε-GAP-3SAT. So ε-GAP-3SAT can be no harder than the above problem of approximating SAT. How hard is it?

Theorem 3 (Johan Håstad)**.** *For every* $\varepsilon < 1/8$*,* ε-GAP-3SAT *is* NP*-hard.*

So finding an *approximate* assignment for 3SAT is as hard as finding an exact assignment! The constant 1/8 in Håstad's theorem is optimal; Uri Zwick showed that 1/8-GAP-3SAT can be solved in polynomial time.

Proving Håstad's theorem requires several tools which we won't have time to cover. However, if we are willing to settle for a suboptimal value of ε (i.e., show that approximating 3SAT is hard for *some* constant ε), we will see that the statement follows easily (and is in fact equivalent to) the PCP theorem.

2.1 Constrant satisfaction problems

To explain the connection between probabilistically checkable proofs and hardness of approximation, it will be convenient to work with a generalization of qSAT (qCNF satisfiability) called qCSP, for "q-ary constraint satisfaction problem satisfiability". Let $x_1, \ldots, x_n$ be variables taking values in $\{0,1\}$. A *q-ary constraint satisfaction problem* (over alphabet $\{0,1\}$) is a collection of *constraints*

$\phi_i \colon \{0,1\}^q \to \{0,1\}$, each of which is associated with a sequence of q variables $x_{i_1}, \ldots, x_{i_q}$. So qSAT is a special case of qCSP where all the constraints are disjunctions of literals; in a qCSP, *any* constraint in q variables is allowed.

Just like we defined ε-GAP-3SAT, we have the following promise problem which captures the hardness of approximating q-ary constraint satisfaction problems:

Promise problem GAP-qCSP
Input: A $q$ary constraint satisfaction problem Ψ.
Yes instances: Some assignment satisfies all constraints of Ψ.
No instances: Every assignment satisfies fewer than a 1/2-fraction of all the constraints in Ψ.

Here we specialize ε to $1/2$ for convenience. Here is the connection between PCPs and CSPs:

$$\boxed{L \in \mathrm{PCP}(O(\log n), q) \qquad \Longleftrightarrow \qquad L \text{ reduces to GAP-}q\mathrm{CSP}}$$

This connection follows more or less directly from the definitions, but it plays a crucial role in many hardness of approximation proofs.

For the forward direction, assume L has a PCP of randomness complexity $O(\log n)$ and query complexity q. Then instance z (of length n) of L reduces to a CSP Ψ with variables$x_1, \ldots, x_n$ taking value in $\{0,1\}$, where x_i represents the contents of the ith location in the proof. For each setting of the randomness r of the verifier there is a constraint ψ_r whose variables $x_{r_1}, \ldots, x_{r_q}$ correspond to those places in the proof queried by the verifier. The constraint $\psi(x_{r_1}, \ldots, x_{r_q})$ takes the value 1 only when the verifier would accept if it would have read the bits $x_{r_1}, \ldots, x_{r_q}$ from the proof. Notice that there are $2^{O(\log n)} = \mathrm{poly}(n)$ such constraints and each of them can be written down in time polynomial in n: To write down the constraint we run the verifier (with a fixed setting of the randomness) on all possible values of the proof bits it depends on and see when it accepts and when it rejects.

If z is a "yes" instance of L, then there is some proof π that makes V accept regardless of the randomness. Then setting x_i to the value of the ith element in π will satisfy all the constraints of Ψ. If z is a "no" instance of L, then no matter what π is V rejects at least half the time. Therefore no matter what the variables x_i are set to, at least half of the constraints will not be satisfiable.

For the reverse direction, we are given a polynomial-time reduction from L to GAP-qCSP. Now consider the following PCP verifier V for L: On input z, run the reduction to produce a CSP Ψ over variables $x_1, \ldots, x_n$. We view the proof π as the sequence of values for $(x_1, \ldots, x_n)$. The verifier chooses a random constraint ψ of Ψ, queries the locations of the proof indexed by the variables in ψ, evaluates ψ on the answers, and accepts iff ψ evaluates to 1. Just as above, it is easy to see that if z is in L, then some assignment satisfies all the constraints of Ψ and the corresponding proof will be accepted with probability 1; if z is not in L, then no assignment satisfies more than half the constraints of Ψ, so no matter which proof V has access to it is bound to reject at least half the time.

2.2 Hardness of approximation for 3SAT

Once we have one problem that is hard to approximate – GAP-qCSP – we can get other ones via reductions.

For example, to show that ε-GAP-3SAT is hard, we can reduce from GAP-qCSP. Given an instance Ψ of GAP-qCSP, we can transform each of its constraints $\psi_i(x_{i_1}, \ldots, x_{i_q}$ into a 3CNF formula ϕ_i in variables $x_{i_1}, \ldots, x_{i_q}, y_{i1}, \ldots, y_{is}$, $s = O(2^q)$, such that $(x_{i_1}, \ldots, x_{i_q})$ is a satisfying assignment for ψ_i if and only if there exists a choice of $y_{i1}, \ldots, y_{is}$ so that $(x_{i_1}, \ldots, x_{i_q}, y_{i1}, \ldots, y_{is})$ satisfies ϕ_i.

We have already seen how to do this transformation: Since ψ_i is a function from $\{0,1\}^q$ to $\{0,1\}$, it can be computed by a circuit of size $4 \cdot 2^q$. We can transform this circuit into a 3CNF formula using the method we introduced when reducing CSAT to 3SAT. The resulting formula will have at most $32 \cdot 2^q$ clauses.

Now consider the 3CNF instance ϕ obtained by taking a conjunction in all the clauses in all such formulas ϕ_i. Clearly if Ψ has a satisfying assignment, it can be extended to a satisfying assignment for ϕ. We now argue that if no assigmnent satisfies more than half the constraints in Ψ, then no assignment can satisfy more than $1 - 1/64 \cdot 2^q$ fraction of the clauses in ϕ.

We prove this by contradiction. Assume there is an assignment that satisfies a $1 - \varepsilon$-fraction of constraints in ϕ, where $\varepsilon = 1/64 \cdot 2^q$. So at most an ε-fraction of the clauses in ϕ are not satisfied by this assignment z. Since every formula ϕ_i has $32 \cdot 2^q$ clauses, it follows that at most an $2^q \cdot \varepsilon \leq 1/2$ fraction of the formulas ϕ_i are not satisfied by the assignment. Now if a formula ϕ_i is satisfied by z, then the corresponding constraint ψ_i is also satisfied by the x-part of z. So more than half of the ψ_i are simultaneously satisfiable, and therefore Ψ is not a "no" instance of GAP-qCSP.

To summarize, we gave a polynomial-time reduction from a qCSP instance Ψ to a 3CNF *phi* such that

$$\begin{aligned} \text{if } \Psi \text{ is satisfiable,} \quad & \text{then } \phi \text{ is satisfiable} \\ \text{if } \Psi \text{ is } \leq 1/2\text{-satisfiable,} \quad & \text{then } \phi \text{ is } \leq 1 - 1/64 \cdot 2^q\text{-satisfiable.} \end{aligned}$$

Since q is a constant, so is $\varepsilon = 1/64 \cdot 2^q$. By the PCP theorem, we have a polynomial-time reduction from SAT to ε-GAP-3SAT, and so the latter one is NP-hard.

2.3 Hardness of approximation for independent set

An *independent set* in an undirected graph is a collection of vertices such that no pair is connected by an edge. Calculating the exact set of an independent set in a graph is NP-hard. Now we will see how it follows from the PCP theorem that the size of the maximum independent set in a graph is hard to approximate within any constant factor.

To do so we look at the following promise version of independent set: Given two constants $0 < s \leq c \leq 1$, we define

Promise problem GAP-(c, s)IS
Input: An undirected graph G on n vertices.

Yes instances: G has an independent set of size cn.
No instances: G has no independent set of size sn.

Suppose we had an algorithm A that, given a graph G whose maximum independent set has size k, finds an independent set of size εk for some constant $\varepsilon > 0$. Then GAP-(c, s)IS would be solvable in polynomial time whenever $s < \varepsilon c$: To solve it simply run A and accept if A finds an independent set of size sn or more. If G has an independent set of size cn, then A is guaranteed to find one of size $\varepsilon cn > sn$, so we will accept. If G has no independent set of size sn, A will never find such a set, so we will reject.

Theorem 4. *For every $\varepsilon > 0$ there exists a $c > 0$ such that the promise problem GAP-$(c, \varepsilon c)$IS is* NP*-hard.*

We first give the proof in the case $\varepsilon = 1/2$. The proof is by reduction from GAP-qCSP. Given a GAP-qCSP instance Ψ with m constraints and n variables, we create the following graph G: The vertices of G will be divided into clusters, and there will be one cluster for each constraint $\psi_i(x_{i_1}, \ldots, x_{i_q})$ of Ψ. Within each cluster, there will be a vertex for every assignment to $x_{i_1}, \ldots, x_{i_q}$ such that $\psi_i(x_{i_1}, \ldots, x_{i_q}) = 1$. Notice that there are m clusters and at most 2^q vertices per cluster.

We now describe the edges of G. The vertices of G represent partial assignments to the variables: The edges will represent inconsistencies among these assignments. Specifically, take two vertices. Each of these vertices represents an assignment to a subset consisting of q variables. Some of the variables may be shared. We connect the two vertices if *at least one of the shared variables* gets a different value (0 at one vertex, 1 at the other). In particular, all the vertices within a cluster are connected by a clique.

Suppose Ψ has a satisfying assignment x. Then G must have an independent set of size m: Within each cluster, choose the vertex that represents an assignment derived from x. (Since x is satisfying for ϕ_i, such a vertex will exist in cluster i). All these assignments are consistent, so the corresponding vertices form a clique.

Now suppose every assignment of Ψ satisfies at most half the clauses. We will argue that every independent set in G has size less than $m/2$. For contradiction, assume G has an independent set of size at least $m/2$. The vertices in this independent set must come from at least $m/2$ different clusters, each representing a clause. Since the partial assignments representing all these vertices are consistent, there must be a single assignment x which is consistent with all these partial assignments. This x will satisfy at least $m/2$ different clauses.

To summarize, we proved that:

$$\begin{aligned} \text{if } \Psi \text{ is satisfiable,} &\quad \text{then } G \text{ has an i.s. of size } m \\ \text{if } \Psi \text{ is } \leq 1/2\text{-satisfiable,} &\quad \text{then } G \text{ has no i.s. of size } m/2. \end{aligned}$$

This proves the theorem for $\varepsilon = 1/2$ (set c to be m divided by the total number of vertices).

To handle smaller values of ε, notice that there is nothing special about the soundness constant $1/2$ in the PCP theorem. By running the verifier k times independently at random (and accepting iff all runs accept), we can amplify the soundness constant from $1/2$ to $1/2^k$. We have to pay a price:

Both the randomness complexity and the query complexity go up by a factor of k. But as long as k is constant, this does not change the statement of the theorem. Choose k so that $1/2^k = \varepsilon$. Then the same argument shows that the theorem is true for this value of ε.

3 A weak PCP theorem

We now prove the following weak version of the PCP theorem:

Theorem 5. $\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), O(1))$.

While this theorem does not have as many applications (as the corresponding CSP instance is of exponential size), it will play a role in the proof of the actual PCP theorem which we will see next time.

To prove this theorem we need to design a PCP with the required parameters for some NP-complete problem. Instead of working with SAT like we usually do, it will be more convenient to choose another NP-complete problem.

But in order to avoid some technical complications, we will start by explaining our ideas using a simpler problem called LIN. In this problem, the input is a system of linear equations modulo 2, like

$$\begin{aligned} x_1 + x_2 + x_5 &= 1 \\ x_1 + x_3 + x_4 + x_7 &= 0 \\ &\vdots \\ x_2 + x_7 &= 0. \end{aligned}$$

The problem is to decide if this system of equations has a solution or not. In fact LIN is in P, so it is pointless to design proof systems for it. But let's forget this for a moment and see how we can design a PCP for LIN, because the ideas will be useful for proving the weak PCP theorem.

Let's start with a standard NP proof for LIN. An NP-proof can consist of the values for all the variables $x_1, \ldots, x_n$. The verifier then checks that these values satisfy all the equations. To read this proof, the verifier needs to observe the values of all n variables, so a total of n queries. How is it possible to do with less?

Here is the first idea. A system of m equations over n variables modulo 2 can be written as $Ax = b$, where A is an m by n matrix and b is a column vector of length n. (Here the vectors are column vectors.) Now let r be a *random* vector of length m. Consider the products $r^{\mathrm{T}}Ax$ and $r^{\mathrm{T}}b$, which are scalar values in $\{0, 1\}$. If $Ax = b$, then $r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$. An easy but important observation is that if *at least one* of the equations fails to hold, namely $Ax \neq b$, then $\Pr[r^{\mathrm{T}}Ax = r^{\mathrm{T}}b] = 1/2$. To see this, notice that $\Pr[r^{\mathrm{T}}Ax = r^{\mathrm{T}}b] = \Pr[r^{\mathrm{T}}(Ax - b) = 0]$. If $Ax - b$ is nonzero, then the product $r^{\mathrm{T}}(Ax - b)$ has equal chance of being 0 and 1, so the probability is exactly $1/2$.

Now notice that once we fix r, the expression $r^{\mathrm{T}}Ax$ is just some linear combination of the x_is; so checking that $r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$ amounts to verifying that some linear combination of the x_is adds to

the desired value $r^{\mathrm{T}}b$. In the worst case, however, this expression may involve all the variables x_i, which take n queries to read, so it seems like we have not made any progress towards verifying the satisfiability of the LIN system with a constant number of queries.

Now comes the second idea. A classical NP-proof for a LIN instance consists of the values for all the variables $x_1, \ldots, x_n$. In the PCP proof, we will ask not only for the values of the variables, but also for the value of every *linear combination* of them. In other words, for every $a \in \{0,1\}^n$, there will be a bit in the proof (indexed by the string a) which is supposed to give the value of the linear combination $\langle a, x\rangle = a_1x_1 + \cdots + a_nx_n$. Then, to find out the value $r^{\mathrm{T}}Ax$, it is sufficient to query this proof at the location $a = r^{\mathrm{T}}A$ (which is an n bit vector).

So here is the suggested PCP for LIN: On input a LIN system $Ax = b$, the verifier expects a proof $\pi \in \{0,1\}^{2^n}$, where for every $a \in \{0,1\}^n$, $\pi(a)$ is supposed to equal the value $\langle a, x\rangle$. The verifier chooses a random $r \in \{0,1\}^m$, computes $a = r^{\mathrm{T}}A$ and checks that $\pi(a) = r^{\mathrm{T}}b$.

If the system $Ax = b$ is satisfiable and x is a satisfying assignment, then the proof π where $\pi(a) = \langle a, x\rangle$ clearly makes the verifier accept with probability 1. Now we want to claim that if the system is not satisfiable, then the verifier rejects with probability at least $1/2$.

We can try to reason along the following lines: If $Ax \neq b$ for every x, then no matter what x is, we have that $r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$ with probability $1/2$. Since $\pi(a) = \langle a, x\rangle = r^{\mathrm{T}}Ax$, the verifier rejects with probability $1/2$. However, this reasoning is incomplete. What is the problem?

The issue is that the verifier cannot be sure that $\pi(a) = \langle a, x\rangle$ for every proof location a; the proof may easily claim, say, that $x_1 = 0$, $x_2 = 0$, but $x_1 + x_2 = 1$. If the verifier queries this proof at $x_1 + x_2$, he will not be aware of this inconsistency. So a proof can try to cheat the verifier in the sense that it tells it whatever it wants to hear in order to accept (i.e. $\pi(r^{\mathrm{T}}A)$ could always have the value $r^{\mathrm{T}}b$.)

To take care of this issue, the verifier wants to "force" the proof to be of a particular form: Namely, he wants to be sure that for every a, $\pi(a) = \langle a, x\rangle$ for some assignment x. But how can the verifier force an exponentially long proof to have this property for every a while making only a constant number of queries?

This is in fact impossible. What the verifier will be able to do instead is ensure that with high probability, $\pi(a) = \langle a, x\rangle$ for *most* (say 90%) values of a. Once this is established, it turns out that the verifier can recover the other (10%) of the values, and thus have (with high probability) *virtual access* to a correct proof of the form $\pi(a) = \langle a, x\rangle$.

3.1 The linearity test

Let us reformulate the problem we have to deal with: Given access to a proof π, which we think of as a string of length 2^n, we want to guarantee (with high probability) that π is of the form $\pi(a) = \langle a, x\rangle$ for some x while making only a constant number of (random) queries into π.

To do this it will be useful to think of π not as a *proof* but as a *function* of a: This is a function f that maps $\{0,1\}^n$ to $\{0,1\}$. We want to check that f is of the form $f(a) = \langle a, x\rangle$ for some x; in other words, f should be a *linear* function over $\mathbb{F}_2$, the field of two elements.

It should be clear that if we only make a constant number of queries into f, it should be impossible to distinguish (with constant probability) the case when f is linear from the case when f is not linear. We can take any linear function (say the all zero function), and make it non-linear by changing one random value of it. Then no test that makes a constant number of queries is likely to query this particular value, so it will fail to detect this non-linearity.

But in some sense taking a linear function and changing it in one place makes it only a little bit non-linear; it turns out that, for our purposes, we can view such a function as essentially linear. It will be sufficient for us to distinguish linear functions from those that are "far" from being linear.

Definition 6. A function $f\colon \{0,1\}^n \to \{0,1\}$ is *δ-close* to linear if there is a linear function $\ell(x) = \langle b, x\rangle$ such that f and ℓ differ in at most $\delta \cdot 2^n$ outputs. We say f is *δ-far* from linear if it is not δ-close to linear.

We are now ready to describe *linearity testing* as the following (oracle) promise problem. Given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$:

Yes instances: f is a linear function.
No instances: f is δ-far from every linear function.

Think of δ as some constant, like $1/100$.

Theorem 7. *Assume $\delta < 1/12$. There is a randomized procedure $T^?$ which (on input 1^n) makes 3 queries into its oracle and*

- *If $f\colon \{0,1\}^n \to \{0,1\}$ is linear, then T^f accepts with probability 1,*
- *If f is δ-far from linear, then T^f rejects with probability at least $\delta/2$.*

The idea behind the linearity test is extremely simple. A function $f\colon \{0,1\}^n \to \{0,1\}$ is linear if and only if

$$f(x) + f(y) = f(x+y)$$

for every pair $x, y \in \{0,1\}^n$. The linearity test will check that this relation is satisfied for a *random* pair x, y.

The linearity test: On input 1^n, choose $x, y \in \{0,1\}^n$ independently at random, query the oracle at locations x, y and $x+y$ and accept iff $f(x) + f(y) = f(x+y)$.

Clearly, if f is linear, the test accepts with probability 1. We need to show that if f is δ-far from linear, then the test rejects with probability at least $\delta/2$. We will prove the contrapositive, namely

> If the test rejects with probability less than $\delta/2$, then f is δ-close to linear.

So let's assume that the test rejects with probability $< \delta/2$. We define the function $\ell\colon \{0,1\}^n \to \{0,1\}$ at x to equal the majority (over y) of the values $f(x+y) + f(y)$, namely:

$$\ell(x) = \begin{cases} 1, & \text{if } \Pr_y[f(x+y) + f(y) = 1] \geq 1/2, \\ 0, & \text{otherwise.} \end{cases}$$

We will now prove that (1) ℓ is δ-close to f and (2) ℓ is linear.

The first part is straightforward. By assumption, we have that

$$\Pr_{x,y}[f(x) \neq f(x+y) + f(y)] < \delta/2$$

so by Markov's inequality,

$$\Pr_x\big[\Pr_y[f(x) \neq f(x+y) + f(y)] \geq 1/2\big] < \delta.$$

Now if $\Pr_y[f(x) \neq f(x+y) + f(y)] < 1/2$, then $\Pr_y[f(x) = f(x+y) + f(y)] \geq 1/2$ and therefore $\ell(x) = f(x)$, so ℓ and f match on a $1 - \delta$ fraction of outputs.

For the second part, by the same reasoning as above, we have that

$$\Pr_x\big[\Pr_y[f(x) \neq f(x+y) + f(y)] \geq 1/4\big] < 2\delta.$$

Let $S \subseteq \{0,1\}^n$ be the set of those x such that

$$\Pr_y[f(x) \neq f(x+y) + f(y)] < 1/4.$$

Fix a pair of inputs x, y from S. We have that

$$\begin{aligned}
\Pr_u[\ell(x) \neq f(x+u) + f(u)] < 1/4 \qquad & \text{since } \ell(x) = f(x) \text{ for } x \in S \\
\Pr_u[\ell(y) \neq f(y+u) + f(u)] < 1/4 \qquad & \text{since } \ell(y) = f(y) \text{ for } y \in S \\
\Pr_u[\ell(x+y) \neq f(x+u) + f(y+u)] < 1/2 \qquad & \text{by definition of } \ell
\end{aligned}$$

Taking a union bound, we have that if $x, y \in S$, then

$$\Pr_u[\ell(x) + \ell(y) \neq \ell(x+y)] < 1.$$

Since the quantity inside the probability is independent of u, it must hold that $\ell(x)+\ell(y) = \ell(x+y)$ for all $x, y \in S$.

Now let x, y be arbitrary. Then

$$\Pr_u[\ell(x) \neq \ell(x+u) + \ell(u)] < \Pr[x+u \notin S \text{ or } u \notin S] < 4\delta$$

since if both $x+u$ and u are in S, then $\ell(x+u) = \ell(x) + \ell(u)$. Similarly

$$\begin{aligned}
\Pr_u[\ell(y) \neq \ell(y+u) + \ell(u)] < 4\delta \\
\Pr_u[\ell(x+y) \neq \ell(x+u) + \ell(y+u)] < 4\delta
\end{aligned}$$

Taking a union bound again, we have that

$$\Pr_u[\ell(x) + \ell(y) \neq \ell(x+y)] < 12\delta \leq 1$$

and since the event is independent of u, equality must hold for all x, y, so ℓ is linear.

3.2 A weak PCP for LIN

We are now almost ready to describe our weak PCP for LIN. Let's recall what we did so far: Given a system of equations $Ax = b$, we expect a PCP proof of the form $\pi(a) = \langle a, x \rangle$ for all $a \in \{0,1\}^n$. The linearity test (repeated a constant number of times) allows us to verify, with probability at least say $9/10$, that π is $1/10$-*close* to a proof of this form. If we could tell that π was exactly of this form, then we can finish by choosing a random linear combination r of the equations in the system, query the proof at location $r^{\mathrm{T}}A$, and accept iff the queried value equals $r^{\mathrm{T}}b$. By our previous reasoning, if π encodes a satisfying assignment for the system then the verifier accepts with probability 1 and if it doesn't, it accepts with probability $1/2$.

There is one problem left to resolve: The linearity test does not allow us to conclude that π is *exactly* linear, but merely that it is $1/10$-close to linear.

However, it turns out that once we know that π is $1/20$-close to a linear function ℓ, we can simulate a query into ℓ by making *two* queries into π. The idea is simple: To find the value of $\ell(y)$, we choose a random u and output $\pi(y+u) + \pi(u)$. Since both $y+u$ and u are random, with probability $1 - 2/10 = 4/5$ we will have $\pi(y+u) = \ell(y+u)$ and $\pi(u) = \ell(u)$, and therefore $\pi(y+u) + \pi(u) = \ell(y+u) + \ell(u) = \ell(y)$. This procedure which extracts the value of $\ell(y)$ out of π (which is a faulty version of ℓ) is called *local correction*, as it allows us to correct the faults in y locally (i.e. with very few queries).

We can now give the full PCP for LIN: On input $Ax + b$ and given oracle access to a proof π, first run the linearity test a constant number of times to ensure that with probability at least $9/10$, π is $1/20$-close to a linear function ℓ. Then choose a random r and compute the value $a = r^{\mathrm{T}}A$. Now to find $\ell(a)$, choose a random u and query π at locations $a+u$ and u. Accept iff $\pi(a+u)+\pi(u) = r^{\mathrm{T}}b$.

If $Ax + b$ is satisfiable, then the proof $\pi(a) = \langle a, x \rangle$ will pass the linearity test with probability 1, and $\pi(a+u) + \pi(u) = \langle a, x \rangle$ with probability 1. Since $\langle a, x \rangle = r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$, the verifier accepts with probability one.

If $Ax + b$ is not satisfiable, let $\ell(a) = \langle a, x \rangle$ be the closest linear function to the proof π provided to the verifier. By the analysis of the linearity test, with probability $9/10$, π and ℓ are $1/10$-close. Assume this is the case. Then $r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$ with probability $1/2$. On the other hand, $\ell(a) = \pi(a+u) + \pi(u)$ with probability at least $4/5$. All these events hold with probability $1 - 1/10 - 1/2 - 1/5 \geq 1/10$; if this is the case, then

$$\pi(a+u) + \pi(u) = \ell(a) = \langle a, x \rangle = r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$$

so the verifier rejects.

To summarize, if $Ax = b$ is satisfiable, there exists π which makes the verifier accept with probability 1, and if not, then every π makes the verifier reject with probability at least $1/10$. We can repeat the PCP a constant number of times to amplify the rejection probability to $1/2$.

3.3 A weak PCP for all problems in NP

We now generalize the PCP for LIN into one that works for all of NP. To do so we need an NP-complete problem which looks like LIN. This problem is called QE (for quadratic equations modulo 2). An instance of QE is a system of homogeneous quadratic equations modulo 2 like

$$\begin{aligned} x_1x_3 + x_2x_2 + x_5x_7 &= 1 \\ x_1x_1 + x_1x_3 + x_4x_4 + x_2x_7 &= 0 \\ &\vdots \\ x_2x_2 + x_3x_3 &= 0. \end{aligned}$$

You can show that this problem is NP-complete by reduction from CSAT, in much the same way we proved that SAT is NP-complete.

Now QE looks a lot like LIN, except that the equations are modulo 2. Each of these equations is of the form

$$\sum_{i,j=1}^{n} a_{ij}x_ix_j = b$$

with $a_{ij}, b \in \{0,1\}$. This suggests the following PCP: Interpret the proof π as a string of length 2^{n^2}, with the hope that $\pi(a_{11}, \ldots, a_{nn})$ should equal the value of $\sum_{i,j=1}^{n} a_{ij}x_ix_j$ for some x. Then just like in the case for LIN, we can take a random linear combination of the equations and query π at the appropriate location to verify that x is indeed a satisfying assignment; if it is not, by exactly the same reasoning as before, we detect an inconsistency with probability $1/2$.

Again, the main problem is how to ensure that the proof π is of the correct form, namely

$$\pi(a_{11}, \ldots, a_{nn}) = \sum_{i,j=1}^{n} a_{ij}x_ix_j$$

for some $x \in \{0,1\}^n$. Notice that this π is a linear function of the a_{ij}s; so we can at least check that it is close to linear using the linearity test we designed.

However, checking linearity is not sufficient; we must also guarantee that the coefficients of this linear function are of the form x_ix_j for some x. To do so, it will help to have access to an auxiliary proof ρ of length 2^n whose expected form will be

$$\rho(b_1, \ldots, b_n) = b_1x_1 + \cdots + b_nx_n.$$

Given access to such a ρ, we can also verify that it is close to linear by running the linearity test. Once we know that π and ρ are close to linear, we can treat them both as linear by using local decoding, so let's assume that they are both linear. Now it remains to check that they are consistent (i.e., they encode the same x). How do we check consistency? To do this, notice that if π and ρ are indeed consistent, then it must be the case that for every $b, c \in \{0,1\}^n$:

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) = \pi(b_1c_1, b_1c_2, \ldots, b_nc_n)$$

which suggests the following consistency test: Choose b, c at random from $\{0,1\}^n$ and check that the above relation is satisfied.

Clearly if ρ and π are consistent linear functions this test will accept with probability 1. We now argue that if ρ and π are linear but inconsistent then the test will reject with probability at least $1/4$.

Any two linear functions ρ and π have the form

$$\rho(b_1, \ldots, b_n) = b_1 x_1 + \cdots + b_n x_n$$
$$\pi(a_{11}, \ldots, a_{nn}) = a_{11} y_{11} + a_{12} y_{12} + \cdots + a_{nn} y_{nn}$$

for some $x \in \{0,1\}^n, y \in \{0,1\}^{n^2}$. Now suppose that $y_{ij} \neq x_i x_j$ for at least one pair (i, j). Then we claim that with probability at least $1/4$,

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) \neq \pi(b_1 c_1, b_1 c_2, \ldots, b_n c_n).$$

To see this, we evaluate both sides:

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) = \sum_{i,j=1}^{n} b_i c_j x_i x_j$$
$$\pi(b_1 c_1, \ldots, b_n c_n) = \sum_{i,j=1}^{n} b_i c_j y_{ij}.$$

Let's think of the difference of the two: For fixed x and y, this is a quadratic polynomial in the variables b_i and c_i. Our assumption is that at least one of the terms $x_i x_j$ and y_{ij} are different, so this polynomial is not identically zero. But when we evaluate a non-zero quadratic polynomial over $\mathbb{F}_2$ at a random input, the probability of it vanishing is at most $1/4$. One way to see this is to write the polynomial in the form

$$\sum_{i=1}^{n} b_i \cdot (\text{some linear function of } c_1, \ldots, c_n).$$

Since the polynomial is non-zero, at least one of these linear functions is non-zero, so with probability $1/2$ it won't vanish after $c_1, \ldots, c_n$ are replaced by random values. After this replacement is done, we just get a non-zero linear function in $b_1, \ldots, b_n$, so with another $1/2$ probability this won't vanish after $b_1, \ldots, b_n$ are replaced by random values.

To summarize, we have the following PCP for QE. The proof is of the form (π, ρ), where π has length 2^{n^2} and ρ has length 2^n. We expect ρ and π to encode some satisfying assignment x to the system of equations. To check this is the case, we run the following tests:

1. **Linearity test:** Run the linearity test a constant number of times on π and ρ. This ensures that with probability 99%, both π and ρ are 1%-close to linear functions.
2. **Consistency test:** Choose random $b, c \in \{0,1\}^n$. We want to check that

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) = \pi(b_1 c_1, b_1 c_2, \ldots, b_n c_n)$$

but because ρ and π are only 1%-close to linear (and not linear), we do so via local decoding: So we choose extra random $u, v \in \{0,1\}^n, w \in \{0,1\}^{n^2}$ and check that

$$(\rho(b+u)+\rho(u))(\rho(c+v)+\rho(v)) = \pi(b \circ c + w) + \pi(w)$$

where $b \circ c = (b_1c_1, b_1c_2, \ldots, b_nc_n)$.

3. **Satisfiability test:** Choose random $r \in \{0,1\}^m$. Take a linear combination of the equations as indexed by r and let a denote the vector of coefficients. We want to check that $\pi(a) = r^{\mathrm{T}}b$, but because π is only 1%-close to linear we do so via local decoding: So we choose an extra random $w \in \{0,1\}^{n^2}$ and check that $\pi(a+w) + \pi(w) = r^{\mathrm{T}}b$.

4. If all test pass accept, otherwise reject.

By the previous discussion (modulo the exact calculation of probabilities), we have proved that if the system is satisfiable, then π and ρ that encode a satisfying assignment make the verifier accept with probability 1. If not, then no matter what π and ρ are, the verifier rejects with constant probability. Since the verifier makes only a constant number of queries, we obtain the desired PCP for NP.