CSE200: Complexity theory
Interactive proofs

Shachar Lovett

March 11, 2020

1 What are interactive proofs?

We defined NP as the class of languages, for which a witness (or a proof) can be verified efficiently (in polynomial time). For example, consider 3-SAT. The input is a 3-CNF, and the witness that it is satisfiable is an assignment to the variables that satisfies it. We think of a "prover" trying to prove that a given input is in the language (in our example, that a 3-CNF formula is in the language), and a "verifier" that wants to make sure that the prover is not cheating. The prover is all powerful, while the verifier has limited resources (in our case, it is limited to run in polynomial time).

More generally, we can think of a more involved interaction between the verifier and the prover, where the verifier can ask questions, to which the prover needs to supply answers that will satisfy the verifier. Consider an input x given to the verifier and the prover. An interaction is a sequence of messages that the verifier and prover exchange:

- The verifier sends a message m_1 which depends on x.
- The prover sends a message m_2 which depends on x, m_1.
- The verifier sends a message m_3 which depends on x, m_1, m_2.
- The prover sends a message m_4 which depends on x, m_1, m_2, m_3.
- ...

At the end, the verifier needs to decide whether to accept the input x or not, based on the interaction with the prover.

1.1 Deterministic interactive proofs

As a first step, lets consider deterministic interactive proofs, where both the verifier and the prover are deterministic. Recall that the verifier runs in polynomial time, and hence given an input x, the computation of the messages that she sends is limited to run in polynomial time,

as well as the number of messages. We say that a language L is decided by a deterministic interactive proof if for any input $x \in \{0,1\}^*$ we have:

- If $x \in L$ then there is a prover strategy that makes the verifier accept.
- If $x \notin L$ then for any prover strategy, the verifier rejects.

We have seen one example for a deterministic interactive proofs - the class NP. This is a special case, where the prover sends their proof (or witness) and then the verifier verifies it. It turns out that even if we allow more rounds of interaction, deterministic interactive proofs are equivalent to NP.

Lemma 1.1. *Assume that a language L is decided by a deterministic interactive proof. Then $L \in NP$.*

Proof. Let $x \in \{0,1\}^*$. Assume that there is a prover that would make the verifier accept. Let $m_1, m_2, \ldots, m_t$ be the sequence of messages exchanged between the verifier and this prover. Note that the length of each message is $|m_i| \leq \text{poly}(|x|)$ and the number of messages is $t \leq \text{poly}(|x|)$, and so the entire interaction has polynomial length. To make this into an NP language, the witness is going to be the full sequence of messages, which the verifier can now verify on their own, without any further interaction. To be concrete, the verifier would verify for each message m_i in the witness that they send, that this is the message that they would send given the history of the exchange up to this point. □

So, we need to allow randomness if we hope to get more power from interactive proofs.

1.2 Randomized interactive proofs

We now allow the verifier to be a randomized polynomial-time TM. We can allow the prover to be randomized also, but as the prover is all powerful, it turns out that we can assume without loss of generality that the prover is deterministic. We say that a language L is decided by a randomized interactive proof if for any input $x \in \{0,1\}^*$ we have:

- If $x \in L$ then there is a prover strategy, for which the verifier accept with probability $\geq 2/3$.
- If $x \notin L$ then for any prover strategy, the verifier rejects with probability $\geq 2/3$.

The class of all languages that can be verified by a randomized interactive proof (which is typically simply called an interactive proof) is called IP. Sometimes it makes sense to limit the number of messages. We denote by IP[k] the class of languages which can be decided by a k-round interactive proof, where we can allow k to depend on the input length. Note that IP = IP[poly].

Lund, Fortnow, Karloff and Nisan [2] proved that PH $\subset$ IP, which shows that interactive proofs can be very powerful. Shortly later, Shamir [3] proved that in fact IP = PSPACE, which gave a complete characterization of interactive proofs.

Theorem 1.2. $IP = PSPACE$.

We will show some aspects of this proof later. As a warmup, let us prove that $IP \subset$ PSPACE.

Lemma 1.3. $IP \subset PSPACE$.

Proof. Fix $L \in IP$. We will show that $L \in PSPACE$. Let V be the verifier for the interactive proof. Recall that V is a randomized poly-time TM, that computes the next messages to send, and at the end decides whether to accept the transcript or not. It will be convenient to view the randomness as given externally. So we have two randomized TM:

- **Next message:** $M_{\text{next}}(x, r, m_1, \ldots, m_{i-1})$ - given the input x, randomness r and previous messages $m_1, \ldots, m_{i-1}$, it computes in deterministic poly-time the next verifier message m_i.
- **Decider:** $M_{\text{decide}}(x, r, m_1, \ldots, m_t)$ - given the input x, randomness r and all messages $m_1, \ldots, m_t$, it computes in deterministic poly-time whether to accept the input x or not.

Let us assume for simplicity that the odd messages $m_1, m_3, \ldots$ are sent by the verifier, and the even ones $m_2, m_4, \ldots$ are sent by the prover. Recall that each message has length $\text{poly}(n)$, and the number of messages is also $t = \text{poly}(n)$. The length of the randomness is also polynomial $|r| = \text{poly}(n)$.

Let $\tau = (m_1, \ldots, m_i)$ be a possible prefix over messages in a possible interactive protocol. We define its *value*, denoted $\text{value}(x, \tau)$, as the maximal probability that a prover can convince a verifier to accept, conditioned on the first i messages being $m_1, \ldots, m_i$. Let also $\text{value}(x)$ be the maximal probability that a prover can convince the verifier to accept, where $\text{value}(x) = \text{value}(x, \tau = ())$. Then we have by definition:

- If $x \in L$ then $\text{value}(x) \geq 2/3$.
- If $x \notin L$ then $\text{value}(x) \leq 1/3$.

In particular, if we can compute $\text{value}(x)$ then we can decide if $x \in L$ or not.

We will compute $\text{value}(x, \tau)$ from the bottom up. If $\tau = (m_1, \ldots, m_t)$ is a complete transcript, then

$$\text{value}(x, \tau) = \Pr_r \left[M_{\text{decide}}(x, r, \tau) \text{ accepts} \right]$$

We can compute this in PSPACE by enumerating all possible values for r, for which one running $M_{\text{decide}}(x, r, \tau)$, and the summing up the number of iterations where the TM accepted.

Consider now prefix of messages $\tau = (m_1, \ldots, m_{i-1})$ for some $i < t$. Given a possible next message m_i we denote by $\tau \circ m_i$ the prefix of messages $(m_1, \ldots, m_i)$. If the next message m_i is sent by the verifier then we can use M_{next} to compute it, given the information we have and average over the possible choices for the random bits:

$$\text{value}(x, \tau) = \mathbb{E}_r \text{value}(x, \tau \circ m_i = M_{\text{next}}(x, r, \tau)).$$

If the next message m_i is send by the prover, then we need to maximize over all options for what the prover may sent:

$$\text{value}(x, \tau) = \max_{m_i} \text{value}(x, \tau \circ m_i).$$

We do not want to compute all possible options for value(x, τ) for all possible values of τ, because this would require exponential space, as there are exponentially many possible messages. Instead, we would compute it using recursion (or equivalently, a DFS-like process). Note that as the depth of the recursion is t, we only use polynomial space. □

2 Examples

In this section we give some examples for interactive protocols for languages outside NP.

2.1 Graph non-isomorphism

Let G_1, G_2 be two undirected graphs on n nodes. They are said to be *isomorphic* if they are the same graph, up to relabeling the vertices. Let us assume that the nodes of G_1, G_2 are $[n]$. Given a permutation π on $[n]$ and a graph $G = ([n], E)$, define $\pi(G)$ to be the graph obtained by permuting the nodes according to π. That is, its vertices are $[n]$ and its edges are $\{(\pi(i), \pi(j)) : (i, j) \in E)$. Then G_1, G_2 are isomorphic if there exists a permutation π such that $\pi(G_1) = \pi(G_2)$.

The Graph Isomorphism problem is to decide whether two graphs G_1, G_2 are isomorphic. It is in NP, as the witness is a permutation π such that $\pi(G_1) = G_2$. The Graph Non-Isomorphism problem is the complement problem, where we want to accept pairs of graphs which are not isomorphic. It is clearly in coNP, and based on our current knowledge it is not in NP. We will describe a simple 2-round randomized interactive protocol which can prove that G_1, G_2 are non-isomorphic.

Algorithm 1: Graph-Non-Isomorphism(G_1, G_2)

Input: Graphs G_1, G_2 on n nodes
Output: Are the graphs non-isomorphic?

1 The verifier chooses random $a \in \{0, 1\}$ and random permutation π on $[n]$
2 The verifier sends $\pi(G_a)$ to the prover
3 The prover sends back $b \in \{0, 1\}$
4 The verifier accepts if $a = b$

The next lemma shows why this interactive protocol works.

Lemma 2.1. *Graph non isomorphism is in IP*[2]. *Specifically:*

- *If G_1, G_2 are not isomorphic, then there is a prover strategy, such that the verifier accepts with probability* 1.

- *If G_1, G_2 are isomorphic, then for any prover strategy, the verifier accepts with probability* $1/2$.

Proof. Assume first that G_1, G_2 are not isomorphic. Then given $H = \pi(G_a)$, the prover can find out if H is isomorphic to G_1 or to G_2 (by trying out all possible permutations), and hence return b such that $a = b$ holds with probability 1.

Next, assume that G_1, G_2 are isomorphic. The main point is that the distribution of $\pi(G_1)$ and $\pi(G_2)$ are exactly the same, and hence the prover has no information on whether $a = 0$ or $a = 1$. Thus in this case, for any prover strategy we have $\Pr[a = b] = 1/2$. □

2.2 Quadratic non-residue

A number $1 \leq x \leq n$ is a quadratic residue modulo n if there exists y such that $x = y^2 \pmod{n}$. Clearly checking if x is a quadratic residue is in NP, as the prover can give y. However, we do not know how to show non-quadratic reside in NP. There is a randomized interactive protocol which does that. We need the following definition: $\mathbb{Z}_n^* = \{z \in [n] : \gcd(z, n) = 1\}$ is the set of elements modulo n that have a inverse modulo n.

Algorithm 2: Quadratic-Non-Residue(x, n)

Input: Modulo n, number $x \in [n]$
Output: Is x a quadratic non-residue modulo n?

1 The verifier randomly chooses $z \in \mathbb{Z}_n^*$ a bit $a \in \{0, 1\}$
2 The verifier sends $z^2 x^a$ to the prover
3 The prover sends $b \in \{0, 1\}$ to the verifier
4 The verifier accepts if $a = b$

Note that the verifier needs to sample a random $z \in \mathbb{Z}_n^*$. A simple way to do it is via rejection sampling: sample a random $z \in [n]$, compute $\gcd(z, n)$, and if it not 1 then to try again. It can be shown that the expected number of iterations needed is $O(\log n)$. So the verifier indeed runs in polynomial time (polynomial in $\log n$, which is the input length).

Lemma 2.2. *Quadratic Non-Residue is in $IP[2]$. Concretely:*

- *If x is a quadratic non-residue, then there is a prover strategy, such that the verifier accepts with probability* 1.

- *If x is a quadratic residue, then for any prover strategy, the verifier accepts with probability* $1/2$.

Proof. Define $S_a = \{z^2 x^a : z \in \mathbb{Z}_n^*\}$. If x is a quadratic non-residue, then S_0, S_1 are disjoint, and hence the prover can tell if $a = 0$ or $a = 1$ was chosen. Thus it can make the verifier accept with probability one.

If x is a quadratic residue than $S_0 = S_1$ and hence the prover has no information on what value of a was chosen. Thus for any strategy the probability the verifier accepts is $1/2$. □

3 Arthur-Merlin protocols

It seems that the interactive protocols for Graph Non-Isomorphism or Quadratic Residue depended crucially on the verifier using *private randomness*. That is, keeping its random choices secret, and then using it to test the prover. A natural question to ask is what happens if we disallow that. This is formalized in the notion of *Arthur-Merlin games*. These are a restricted type of interactive protocols, where the verifiers messages are simply completely random messages. We define AM to be the class of languages decided by Arthur-Merlin protocols, and AM[k] to be the class of languages decided by k-round Arthur-Merlin protocols.

The two protocols we described so far (for graph non-isomorphism and quadratic non-residue) are not AM protocols. However, Goldwasser and Sipser [1] proved that any IP protocol can be transformed into an AM protocol.

Theorem 3.1. *AM = IP. Specifically, AM[k] $\subset$ IP[k] $\subset$ AM[$k+2$] holds for any k.*

We will prove a special case, that will illustrate the main ideas but avoid some of the technicalities.

Theorem 3.2. *Graph non-isomorphism is in AM*[2].

Let G_1, G_2 be graphs. For simplicity, we will assume that G_1, G_2 have no trivial automorphisms. That is, there is no permutation π such that $\pi(G_i) = G_i$ except for the identity. Define

$$S = \{H : H \text{ isomorphic to } G_1 \text{ or to } G_2\}.$$

Then:

- If G_1, G_2 are isomorphic then $|S| = n!$.
- If G_1, G_2 are not isomorphic then $|S| = 2 \cdot n!$.

So the goal of the prover is to prove that $|S| \geq 2 \cdot n!$. We note here that in the general case, where G_1 or G_2 may have nontrivial automorphisms, the same idea works with

$$S = \{(H, \pi) : H \text{ is isomorphic to } G_1 \text{ or to } G_2, \quad \pi \in \mathrm{Aut}(H)\}.$$

But to keep the presentation simple, we assume from now that G_1, G_2 have no trivial automorphisms. We will construct a "size checking" protocol that will allow the prove to prove that $|S| \geq 2 \cdot n!$.

3.1 Size checking protocol for P: basic idea

Let $S \subset \{0,1\}^n$ be a set, where we need to decide whether $|S| \leq K$ or $|S| \geq 2K$ where K is known. Let us assume first that $S \in P$. That is, the verifier can efficiently test given an input x if $x \in S$ or not (this does not hold in our case, but we will extend this protocol later to allow $S \in$ NP later). Given this assumption, we will construct an AM[2] protocol for estimating the size of S. More precisely, for some $\alpha \geq 1/8$,

- If $|S| \geq 2K$, then there exists a prover for which the verifier accepts with probability $\geq (3/2)\alpha$.
- If $|S| \leq K$, then for any prover, the verifier accepts with probability $\leq \alpha$.

This is not quite what we need, but we will see how to amplify the errors. First, we need the definition of pairwise independent hash functions.

Definition 3.3 (Pairwise independent hash functions)**.** *Let $n \geq k$. A family $H_{n,k}$ of functions $h : \{0,1\}^n \to \{0,1\}^k$ is called pairwise independent if the following holds. Fix any distinct $x, x' \in \{0,1\}^n$ and any $y, y' \in \{0,1\}^k$. Then for a uniformly chosen $h \in H_{n,k}$ it holds that*

$$\Pr_{h \in H_{n,k}}[h(x) = y, h(x') = y'] = 2^{-2k}.$$

There are explicit constructions of pairwise independent hash functions of size $\text{poly}(n)$.

Claim 3.4. *For any $n \geq k$, there exists an explicit pairwise independent family $H_{n,k}$ of size n^2.*

Proof. We first construct the family for $k = n$. Let $\mathbb{F}_{2^n}$ be the finite field of size 2^n. Define

$$H_{n,n} = \{h(x) = ax + b : a, b \in \mathbb{F}_{2^n}\}.$$

We claim that $H_{n,n}$ is pairwise independent. To see this, fix $x \neq x'$ and y, y' in $\mathbb{F}_{2^n}$. The number of $a, b \in \mathbb{F}_{2^n}$ such that $ax + b = y$ and $ax' + b = y'$ is exactly one, as this is a nonsingular system of linear equations. So

$$\Pr_{h \in H_{n,n}}[h(x) = y, h(x') = y'] = 2^{-2n}.$$

For general $k \leq n$, we can start from the above construction of $H_{n,n}$, and then truncate the output of each function h to the first k bits. □

We will use pairwise independent hash functions to test if a set is small or large.

Claim 3.5. *Let $n \geq k$. Let $S \subset \{0,1\}^n$ be a set of size $|S| \leq 2^k$, and set $p = |S|/2^k$. Let $H_{n,k}$ be a family of pairwise independent hash functions. Sample a random $y \in \{0,1\}^k$ and $h \in H_{n,k}$ and define*

$$q(S) = \Pr_{h \in H_{n,k}, y \in \{0,1\}^k}[\exists x \in S, h(x) = y].$$

Then

$$p - p^2/2 \leq q(S) \leq p.$$

Proof. We have

$$\Pr_{h,y}[\exists x \in S, h(x) = y] = \mathbb{E}_{h,y}[y \in h(S)] = 2^{-k}\mathbb{E}[|h(S)|].$$

The upper bound follows since $|h(S)| \leq |S|$ holds with probability one. For the lower bound we have

$$|h(S)| \geq |S| - \sum_{x \neq x' \in S} 1_{h(x)=h(x')}.$$

Hence,

$$\mathbb{E}_h[|h(S)|] \geq |S| - \binom{|S|}{2} 2^{-k} \geq |S| - \frac{|S|^2}{2} 2^{-k},$$

which gives

$$2^{-k}\mathbb{E}[|h(S)|] \geq p - p^2/2.$$

□

Consider the case where $|S| = K$ or $|S| = 2K$, and let k be such that $2K \leq 2^k < 4K$. Set $\alpha = K/2^k$ so that $1/8 < \alpha < 1/4$. Let $q(S)$ be as in Claim 3.5. Then

- If $|S| = K$ then $p = \alpha$ and hence $q(S) \leq \alpha$.
- If $|S| = 2K$ then $p = 2\alpha$ and hence $q(S) \geq 2(\alpha - \alpha^2) \geq (3/2)\alpha$.

This then gives us the desired AM[2] protocol:

Algorithm 3: AM[2] protocol for size checking (P, basic idea)

Input: Set S in P (given implicitly)
Output: Is $|S| = K$ (reject) or $|S| = 2K$ (accept)?

1 The verifier sends a uniform $h \in H_{n,k}$ and $y \in \{0,1\}^k$ to the prover
2 The prover responds with $x \in \{0,1\}^n$
3 The verifier accepts if $x \in S$ and $h(x) = y$

Claim 3.6. *For $\alpha = K/2^k$, which is in the range $\alpha \in (1/8, 1/4)$, we have:*

- *If $|S| = 2K$ then, there exists a prover strategy, such that the verifier accepts with probability $\geq (3/2)\alpha$.*
- *If $|S| = K$ then, for any prover strategy, the verifier accepts with probability $\leq \alpha$.*

3.2 Size checking protocol for P: full details

We can now describe the full protocol. The main idea is to repeat the core protocol some $t = O(1)$ times, and then accept if the verifier accepts a significant number of times more that αt (recall that we can calculate α). Concretely, we would set the threshold at $(5/4)\alpha t$.

Algorithm 4: AM[2] protocol for size checking (P)

Input: Set S in P (given implicitly)
Output: Is $|S| = K$ (reject) or $|S| = 2K$ (accept)?

1 The verifier sends uniform $h_1, \ldots, h_t \in H_{n,k}$ and $y_1, \ldots, y_t \in \{0,1\}^k$ to the prover
2 The prover responds with $x_1, \ldots, x_t \in \{0,1\}^n$
3 The verifier accepts if the number of x_i such that $x_i \in S$ and $h(x_i) = y_i$ is at least $(5/4)\alpha t$

Claim 3.7. *By choosing t a large enough constant,*

- *If $|S| = 2K$ then, there exists a prover strategy, such that the protocol accepts with probability $\geq 2/3$.*
- *If $|S| = K$ then, for any prover strategy, the protocol rejects with probability $\geq 2/3$.*

Proof. Let $w_i \in \{0,1\}$ be the indicator for $x_i \in S$ and $h(x_i) = y_i$. If $|S| = 2K$ then there is a prover strategy such that $\Pr[w_i = 1] \geq (3/2)\alpha$. If $|S| = K$ then for every prover strategy $\Pr[w_i = 1] \leq \alpha$. The result follows by a concentration bound (either Chernoff of Chebychev would do). □

3.3 Size checking protocol for NP

The set S that we care about for graph non-isomorphism is not in P but is in NP. So, the verifier cannot check on its own if $x \in S$ or not. The solution is simple: the prover would also give a witness w that $x \in S$, and the verifier would use this witness to verify that $x \in S$.

Algorithm 5: AM[2] protocol for size checking (NP)

Input: Set S in NP (given implicitly)
Output: Is $|S| = K$ (reject) or $|S| = 2K$ (accept)?

1 The verifier sends uniform $h_1, \ldots, h_t \in H_{n,k}$ and $y_1, \ldots, y_t \in \{0,1\}^k$ to the prover
2 The prover responds with $x_1, \ldots, x_t \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_t$
3 The verifier accepts if the number of x_i such that $x_i \in S$ (verified using w_i) and $h(x_i) = y_i$ is at least $(5/4)\alpha t$

4 Interactive protocols for coNP

We will prove that coNP has an interactive protocol, which was proved by Lund et al. [2]. This idea was extended by Shamir [3] to show that IP = PSPACE.

Let $\varphi(x) = C_1 \wedge \ldots \wedge C_m(x)$ be a 3-CNF. The prover wants to convince the verifier that there are no satisfying assignments to φ. More generally, we will build a protocol where the prover would prove that φ has exactly k satisfying assignments.

4.1 Simplified protocol

Given a formula φ, let us denote by $\#\varphi$ the number of solutions of φ, namely

$$\#\varphi = \sum_{x \in \{0,1\}^n} \varphi(x).$$

The prover wants to prove that $\#\varphi = k$. The main idea is that the prover would first commit to the number of solutions of simpler sub-formulas, and then verify recursively one of them.

For $b \in \{0,1\}$ define φ_b to be the formula φ restricted to $x_1 = b$. The prover would first send k_0, k_1 to the verifier, with the idea that is should be that $k_b = \#\varphi_b$. The verifier would

first verify that $k_0 + k_1 = k$, then choose a random $b \in \{0, 1\}$, and ask the prover to prove that $\#\varphi_b$ is k_b.

We first present a simplified protocol that gives a tiny advantage to the verifier: if $\#\varphi = k$ then the prover can make the verifier accept always, however if $\#\varphi \neq k$ then then the verifier has a positive (but tiny, 2^{-n}) probability of rejecting.

Algorithm 6: Solution-Count(φ, k)

Input: 3-CNF formula φ, integer k
Output: Accept if $\#\varphi = k$

1 If φ has no variables, then the verifier computes $\#\varphi$, accepts if it equals k and rejects otherwise.
2 The prover sends two numbers k_0, k_1.
3 The verifier checks that $k_0 + k_1 = k$. If not it rejects.
4 The verifier chooses a random $b \in \{0, 1\}$ and runs Solution-Count(φ_b, k_b).

Claim 4.1. *For any formula φ the following holds:*

- *If $\#\varphi = k$ then there is a prover strategy that makes the verifier accept with probability one.*
- *If $\#\varphi \neq k$ then for any prover strategy, the verifier rejects with probability $\geq 2^{-n}$.*

Proof. If $\#(\varphi) = k$ then the prover can always return the correct answers, namely $k_b = \#\varphi_b$. Hence the prover accepts with probability one. If $\#(\varphi) \neq k$ and $k_0 + k_1 = k$ then there must be some $b \in \{0, 1\}$ such that $k_b \neq \#\varphi_b$. The verifier would choose to ask about it with probability $1/2$. If he happens to ask the correct question n times, he would catch that the prover is cheating and would reject. □

4.2 Actual protocol

The main problem with the previous protocol was that if the prover was trying to cheat then, in every iteration, it could have been that one of the numbers k_0, k_1 is correct and the other is wrong. If the verifier happened to ask about the correct number, then the prover would fool the verifier, as he can be honest from that point on.

It turns out that a useful tool to do so is using polynomials over large fields. Let p be prime, and let $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$ denote the finite field of order p. We would consider multivariate polynomials $f(x_1, \ldots, x_n)$ over $\mathbb{F}_p$. Let us first see how to encode formulas using polynomials.

Claim 4.2. *Let φ be a 3-CNF with n variables and m clauses. Let $n < p < 2n$ be prime. There exists a polynomial $f(x_1, \ldots, x_n)$ over $\mathbb{F}_p$ such that:*

1. *$f(x) = \varphi(x)$ for all $x \in \{0, 1\}^n$.*
2. *For any $a \in \mathbb{F}_p^n$ we can compute $f(a)$ in polynomial time.*

3. The degree of f is $d \leq 3m$.

Proof. Let $C_1, \ldots, C_m$ be the clauses of f. For each C_i we can define a polynomial f_i such that $f_i(a,b,c) = C_i(a,b,c)$ for all $a,b,c \in \{0,1\}$, where $\deg(f_i) = 3$. Define $f(x) = \prod_{i=1}^{m} f_i(x)$. □

Given a polynomial f let us define

$$\#\varphi = \sum_{x \in \{0,1\}^n} f(x).$$

The prover needs to convince the verifier that $\#f \equiv k \pmod{p}$. We should choose $p > 2^n$, so that the number of solutions mod p is the same as the global number of solutions. Given a polynomial f, for $b \in \mathbb{F}_p$ let us denote by f_b the polynomial f restricted to $x_1 = b$. Namely, $f_b(x_2, \ldots, x_n) = f(b, x_2, \ldots, x_n)$.

Algorithm 7: Polynomial-Sum(f, k)

Input: Polynomial f on n variables over $\mathbb{F}_p$, integer k
Output: Accept if $\#f = k$

1 If $n = 0$ then the verifier accepts if $f \equiv k$ and rejects otherwise
2 The prover sends the univariate polynomial $g(x_1) = \sum_{y \in \{0,1\}^{n-1}} f(x_1, y)$
3 The verifier checks that $\deg(g) \leq 3m$ and $g(0) + g(1) = k$. Otherwise he rejects.
4 The verifier chooses a random $b \in \mathbb{F}_p$, sets f_b to be the restriction of f to $x_1 = b$, and runs Polynomial-Sum$(f_b, g(b))$.

Claim 4.3. *For any polynomial f the following holds:*

- *If $\#f \equiv k \pmod{p}$ then there is a prover strategy that makes the verifier accept with probability one.*
- *If $\#f \not\equiv k \pmod{p}$ then for any prover strategy, the verifier rejects with probability $\geq 1 - 3mn/p$. In particular, if $p \geq 9mn$ then the verifier rejects with probability $\geq 2/3$.*

Proof. If $\#f \equiv k \pmod{p}$ then the prover can be honest. Note that if $\deg(f) = d$ then also $\deg(g) \leq d$. So assume that $\#f \not\equiv k \pmod{p}$. Let $h(x_1) = \sum_{y \in \{0,1\}^{n-1}} f(x_1, y)$ be the true polynomial for f, where the prover sends g. As $g(0) + g(1) = k$ and $h(0) + h(1) \neq k$ we must have that $g \neq h$. Thus, the number of points $a \in \mathbb{F}_p$ on which $g(a) = h(a)$ agree is at most their degree $d \leq 3m$. Thus the probability that the random b that the verifier chooses happens to be one of these points is at most $3m/p$. We thus get that the probability that the verifier catches the prover is at least $(1 - (3m/p))^n \geq 1 - 3mn/p$. □

References

[1] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 1986.

[2] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 2–10. IEEE, 1990.

[3] A. Shamir. Ip=pspace (interactive proof= polynomial space). In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 11–15. IEEE, 1990.