The Power of Interactive Proofs

CS6230: Topics in Information Security
Lecture 2 (Aug 17, 2021)
Lecture and notes: Prashant Nalini Vasudevan

Last time, we saw the definition of interactive proofs. Over the next few lectures, we will investigate several questions about these proofs:

1. How powerful are interactive proofs?
2. How robust is their definition?
3. How are they useful?

We started answering the first question, and shall continue to do so today. We saw that any language in NP or MA has an interactive proof, and that any language that has an IP is contained in PSPACE. Today, we will see that the latter is, in fact, a characterisation – any language in PSPACE has an interactive proof.

Theorem 0.1. $\mathsf{PSPACE} \subseteq \mathsf{IP}$

Put together with the theorem we saw last class, this implies that $\mathsf{IP} = \mathsf{PSPACE}$. This theorem, originally proven by Lund et al [LFKN92] and Shamir [Sha92], was a landmark achievement in theoretical computer science. Today, we shall see a proof of a slightly weaker theorem. More importantly, we will see the protocol that is central to this proof and is perhaps the most important protocol in the realm of interactive proofs overall – the sum-check protocol.

1 The Sumcheck Protocol

The sumcheck protocol is a protocol for the prover to prove to the verifier that the sum of evaluations of a multilinear polynomial on a structured subset of inputs is equal to what it claims. That is, consider any polynomial $g : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$, of degree at most d in each variable. In the sumcheck protocol, the prover proves statements of the form:

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} g(x_1,\ldots,x_n) = z$$

with perfect completeness and soundness error $O(nd/|\mathbb{F}|)$. Without the prover, the verifier would have had to perform 2^n evaluations of g in order to verify such a statement. In the sumcheck protocol, the verifier will run in time $O(nd \cdot \mathrm{polylog}(|\mathbb{F}|))$ plus $O(1)$ evaluations of g.

The idea behind the protocol is quite simple once stated. It is based on the following fact about polynomials. (Note that below, we use the term "distinct" to indicate that the evaluations of the polynomials differ at at least one point in $\mathbb{F}$.)

Fact 1.1. *Given any two distinct univariate polynomials f, g of degree d over a finite field $\mathbb{F}$, the number of points $x \in \mathbb{F}$ such that $f(x) = g(x)$ is at most d.*

Define the following univariate polynomial g_1:

$$g_1(x) = \sum_{x_2,\ldots,x_n \in \{0,1\}} g(x, x_2, \ldots, x_n) \tag{1}$$

That is, for any $x \in \mathbb{F}$, $g_1(x)$ is equal to the quantity obtained by setting $x_1 = x$ and summing up the evaluations of g when the other variables are set to 0 or 1. We make the following observations.

Claim 1.1. *The degree of g_1 is at most d.*

This is because the degree of g in x_1 is at most d, and the remaining variables are all set in the course of the summation in (1).

Claim 1.2. *Given g_1 (as a list of coefficients), the statement $\sum_{x_1,\ldots,x_n\in\{0,1\}} g(x_1,\ldots,x_n) = z$ can be verified with $O(d)$ field operations.*

This is because of the following:

$$\sum_{x_1,\ldots,x_n\in\{0,1\}} g(x_1,\ldots,x_n) = g_1(0) + g_1(1)$$

Thus, by evaluating g_1 at two points, the above sum can be computed. Evaluation of g_1 at any point needs only $O(d)$ operations given the $(d+1)$ coefficients of g_1.

So if the verifier can somehow obtain the coefficients of g_1, it can verify the original statement itself. Of course, these coefficients are also hard to compute, as this would also involve an exponential sum. What if we ask the prover to send the verifier these coefficients? If the prover sends g_1 correctly, then the verifier can proceed as above. But what if the prover tries to cheat?

In other words, suppose the prover sends the verifier the coefficients of some polynomial g_1'. The verifier then needs to verify that this is the same as the polynomial g_1 as defined in (1). This is where Fact 1.1 comes in. Notice that both g_1 and g_1' are degree-d polynomials. If g_1' is different from g_1, then the evaluations of the two can be equal on at most d points.

So if the verifier can evaluate both these polynomials at a random point x^1 and compare them, it can check, with some confidence, whether they are the same polynomial. It can, of course, evaluate g_1' anywhere it wants since it known the coefficients. But what about the implicitly defined g_1? How can it check that $g_1(x^1) = g_1'(x^1)$? Well, notice that this is the same problem we started with! Expanding the definition of g_1, this is the same as checking the following:

$$\sum_{x_2,\ldots,x_n\in\{0,1\}} g(x^1, x_2,\ldots,x_n) = g_1'(x^1)$$

So, the verifier can pick such a random x^1, send it to the prover, and run the protocol again recursively for the variables $x_2,\ldots,x_n$. The entire protocol may be described as follows. Given a polynomial $g : \mathbb{F}^n \to \mathbb{F}$ of degree at most d in each variables and a $z \in \mathbb{F}$,

<u>Sumcheck(n,g,z):</u>

1. If $n = 1$, the verifier evaluates and checks whether $(g(0)+g(1) = z)$. Accept if so, reject otherwise.
2. The prover sends the verifier the univariate polynomial g_1 (in (1) as a list of $(d+1)$ coefficients. Denote the polynomial that the prover sends as g_1'.
3. The verifier checks whether $(g_1'(0) + g_1'(1) = z)$. If not, reject.
4. The verifier picks a random $x^1 \leftarrow \mathbb{F}$ and sends it to the prover.
5. Define the polynomial $g'(x_2,\ldots,x_n) \triangleq g(x^1, x_2,\ldots,x_n)$.
6. The prover and verifier run Sumcheck($n-1$,g',$g_1'(x^1)$).

Efficiency. Note that the degree of g' in any variable is at most that of g in the same variable, which is bounded by d to begin with. This, in each recursion, the degree of g_1 is at most d, and so each of the steps of the verifier can be completed with $O(d)$ field operations. Since there are n variables, the verifier runs in $O(nd)$ field operations, plus two evaluations of g (when $n = 1$).

Exercise 1. *Show that the prover above can be implemented with $O(nd2^n)$ evaluations of g.*

Completeness. We prove completeness by induction on the number of variables n. For $n = 1$, it is clear that the protocol is perfectly sound, as the verifier simply evaluates g and checks the sum on its own.

Suppose the sumcheck protocol was perfectly complete for polynomials with $(n-1)$ variables. Then, if the statement being proven is indeed true, the prover will honestly report the polynomial g_1 as its g_1'. So it will indeed be the case that $g_1'(0) = g_1'(1) = z$ and the verifier will not reject in Step 3. The recursive call to sumcheck is now on an $(n-1)$-variate polynomial g', and, following the induction hypothesis, will accept because the following is true:

$$\sum_{x_2,\ldots,x_n\in\{0,1\}} g'(x_2,\ldots,x_n) = \sum_{x_2,\ldots,x_n\in\{0,1\}} g(x^1,x_2,\ldots,x_n) = g_1(x^1) = g_1'(x^1)$$

Thus, the verifier will always accept when the statement is true. This proves perfect completeness.

Soundness. Soundness is also proven inductively. For $n=1$, again the protocol is perfectly sound since the verifier checks the sum on its own. Suppose the soundness error with polynomials over n variables is ε_n. We will first compute a bound on ε_n in terms of ε_{n-1}.

Suppose the statement being proven – that $\sum_{x_1,\ldots,x_n\in\{0,1\}} g(x_1,\ldots,x_n) = z$ – is not true. Then, in order to pass the check in Step 3, the polynomial g_1' that a cheating prover sends has to be distinct from the actual g_1. As both of these are polynomials of degree at most d, the probability that $g_1(x^1) = g_1'(x^1)$ for a random $x^1 \leftarrow \mathbb{F}$ is at most $d/|\mathbb{F}|$. If this does not happen, the recursive call to sumcheck is for a false statement with $(n-1)$ variables, and thus is accepted with probability at most ε_{n-1}. Thus, by a union bound, the probability that the verifier accepts – which is ε_n – is at most $(d/|\mathbb{F}| + \varepsilon_{n-1})$.

This implies that $\varepsilon_n \leq (n-1)d/|\mathbb{F}|$. Thus, if $\mathbb{F}$ is large enough, this protocol is sound.

2 IP and coNP

We can now start working towards Theorem 0.1, though we will only prove a weaker version of it. For the complete proof, you are encouraged to read the original paper by Shen [She92] that presents a simplification of Shamir's proof, or the corresponding section in Chapter 8 of the book by Arora and Barak [AB09]. We will start by trying to prove the following relatively weak statement.

Theorem 2.1. coNP $\subseteq$ IP

We will prove this by showing an interactive proof for the coNP-complete problem UNSAT, which consists of all unsatisfiable Boolean formulas. While there are perhaps simpler complete problems we could use for this purpose, using UNSAT will demonstrate some interesting techniques that will be useful later.

Recall that a Boolean formula of size s over n variables is given by a rooted tree with S nodes where each of the leaves is labelled by one of the variables $x_1,\ldots,x_n$, and each internal node by an AND, OR, or NOT gate. Each internal node computes the specified function on its children (the inputs), and the output of the formula is the output of the root. Each NOT gate has only one input, and for simplicity, suppose that each OR and AND gate has two inputs – our proof will only need to be modified slightly otherwise.

The set of all such formulas that have no assignments to the inputs for which the output of the formula is TRUE is the language UNSAT. It is quite easy to prove that a given formula *has* a satisfying assignment – simply provide the assignment, which can then be checked by evaluating the formula. Proving that a formula does not have a satisfying assigment is much harder, and we will use the sumcheck protocol for this.

2.1 Low-Degree Extensions

In order to do so, we will first need to convert the lack of satisfying assignments to a formula into a statement about polynomials. Given a formula ϕ of size s over n variables and a large enough field $\mathbb{F}$, we will define an associated n-variate polynomial $g_\phi : \mathbb{F}^n \to \mathbb{F}$. This polynomial is defined inductively, going up the tree corresponding to the formula, and defining a polynomial g_u for every node u. (Below, $\boldsymbol{x}$ represents the vector of variables $(x_1,\ldots,x_n)$.)

- If u is a leaf labelled with input x_i, then $g_u(\boldsymbol{x}) = x_i$
- If u is labelled by NOT and its child is v, then $g_u(\boldsymbol{x}) = (1 - g_v(\boldsymbol{x}))$
- If u is labelled by AND and its children are v and w, then $g_u(\boldsymbol{x}) = g_v(\boldsymbol{x}) \cdot g_w(\boldsymbol{x})$

- If u is labelled by OR and its children are v and w, then $g_u(\boldsymbol{x}) = 1 - (1 - g_v(\boldsymbol{x}))(1 - \cdot g_w(\boldsymbol{x}))$

Adopting the semantics that TRUE corresponds to the element 1 in the field $\mathbb{F}$ and FALSE corresponds to 0, the following claim may be verified inductively. This is left as an exercise.

Claim 2.1. *For any formula ϕ of size s over n variables and any setting of the variables $x_1, \ldots, x_n \in \{0,1\}$:*

- *The degree of g_ϕ is at most s.*
- *If ϕ is satisfied by the assignment $\boldsymbol{x}$, $g_\phi(\boldsymbol{x}) = 1$.*
- *If ϕ is not satisfied by the assignment $\boldsymbol{x}$, then $g_\phi(\boldsymbol{x}) = 0$.*

Such a polynomial g_ϕ is referred to as a *low-degree extension* of ϕ. More generally, a low-degree extension of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ over a field $\mathbb{F}$ is a polynomial $g : \mathbb{F}^n \to \mathbb{F}$ that is of relatively low degree (as appropriate in the context), and agrees with f on all inputs in $\{0,1\}^n$. The significance of these extensions comes from the fact that they form an "error-correcting" encoding of the function f. That is, given two functions f and f' that are different in even one input, their low-degree extensions will be different on almost all inputs. We will see more about them later in the class.

2.2 PH and IP

To prove Theorem 2.1, we only need the following two observations about the extension g_ϕ of any formula ϕ of size s over n variables:

1. The degree of g_ϕ is at most s.
2. The number of satisfying assignments of ϕ is given by: $\sum_{x_1,\ldots,x_n \in \{0,1\}} g_\phi(x_1, \ldots, x_n)$.

Thus, all a prover needs to do to prove that ϕ has no satisfying assignments is to run a sumcheck protocol to show that the above sum is 0, and make sure to use a field that is much larger than s.

Note that this enables the prover to prove not just that there are no satisfying assignments, but also to prove that the number of satisfying assignments is equal to z for some $z \in [2^n]$. This problem of counting the number of satisfying assignments to a Boolean formula is called #SAT, and is complete for the class #P of counting problems. Further, by Toda's theorem that the polynomial hierarchy is contained in $\mathsf{P}^{\#\mathsf{P}}$, this implies that the hierarchy is also contained in IP.

Theorem 2.2. $\mathsf{PH} \subseteq \mathsf{IP}$

The proof of Theorem 0.1 is not far from here. It involves identifying a PSPACE-complete problem that can also be subjected to a process similar to the above, and a few more steps.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[She92] Alexander Shen. IP = PSPACE: simplified proof. *J. ACM*, 39(4):878–880, 1992.