Lecture 20: The Sum-Check Protocol

Notes by Yael Kalai

MIT - 6.5620
Lecture 20 (November 19, 2025)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Motivation
- Sumcheck protocol

Motivation

Suppose we store our data on a (possibly untrusted) platform and then request the platform to perform computations on our data. Recall that FHE allows us to carry out this task while ensuring the *secrecy* of our data. We now ask how do we ensure the *integrity* of the result? Specifically, how do we know that indeed the platform is doing the instructed computation? In other words, *can we efficiently verify that a computation was done correctly?* Namely, is there a *succinct* and *efficiently verifiable* proof that we can append to the output of a computation attesting to the fact that this output is indeed correct? This is the topic of the next two lectures.

We would like to have a proof of correctness for any time T computation that can be verified in time $<< T$ (say time T^ϵ or even polylog(T)). Unfortunately, we do not believe that every T-computable language has a proof (or a "witness") of size $<< T$.

As you learned by now, cryptography is an art of overcoming such barriers. We overcome this barrier by considering interactive proofs (as opposed to classical proofs, which are deterministic and non-interactive) and by making use of some cryptographic magic!

Interactive Proofs (Recap)

Proof systems have been studied by mathematicians for thousands of years, starting from Euclid (300 BCE). Yet, until recently, all proof systems were of a somewhat similar form which is simply a list of formulas that follow from a set of inference rules and axioms. This

changed in the mid eighties when Goldwasser, Micali and Rackoff defined the notion of a zero-knowledge proof [2] (which we talked about the last two lectures). They realized that zero-knowledge cannot be achieved using classical proofs, and to bypass this barrier they defined the notion of *interactive proofs* which completely changed the way we think about proofs.

In the next two lectures, we will learn about the power of interactive proofs, and their impact on how proofs are designed today. We will show how to use interactive proofs, together with cryptographic magic, to construct *"succinct proofs."* Specifically, we will show how given a Turing machine M, an input x and a time-bound T, one can compute the output $y = M(x)$ together with a "succinct proof" π that certifies that indeed M on input x outputs y within T steps. More generally, we will show how to convert a long proof w that certifies the correctness of a statement $x \in L$ into a "succinct proof" π that certify its correctness.

Think of L as an NP language, where every $x \in L$ has a polynomial size witness w. We will see how to use cryptography to shrink w into a "succinct proof" π.

The class IP is the set of all languages L that have such an interactive proof. Note that $\mathsf{NP} \subseteq \mathsf{IP}$ but IP may contain additional languages. In a celebrated result, Shamir [3] gave a characterization of the class IP by proving that $\mathsf{IP} = \mathsf{PSPACE}$, which means that and every language L in PSPACE has an interactive proof and every language L that has an interactive proof is in PSPACE (the latter is quite straightforward, but the former is highly non-trivial).

Sumcheck Protocol

We start by demonstrating the power of interactive proofs via the Sumcheck protocol, which is an interactive proof for a statement that we do not know how to prove succinctly using a classical proof. Intuitively, the Sumcheck protocol proves the value of the sum of a multivariate polynomial on exponentially many values. Specifically, let $\mathbb{F}$ be a finite field. One can think of $\mathbb{F} = \mathsf{GF}[p]$ which consists of the elements $\{0, 1, \ldots, p-1\}$ where addition and multiplication are modulo p.

Definition 1 (Sumcheck Problem)**.** Given a polynomial $f : \mathbb{F}^m \to \mathbb{F}$ of degree $\leq d$ in each variable and a fixed set $H \subseteq \mathbb{F}$,[1] compute

$$\beta = \sum_{h_1,\ldots,h_m \in H} f(h_1, \ldots, h_m).$$

[1] Often, H is fixed to be $H = \{0, 1\}$.

We will show an interactive proof for the Sumcheck problem, where the verifier runs in time $\mathrm{poly}(m, |H|, d, \log |\mathbb{F}|)$ given oracle access to f. While at first, this problem seems very specific (and possibly not interesting), it turns out that this is an important building block in many of our succinct proofs. In particular, it is the main

building block in Shamir's celebrated IP = PSPACE result [3], and is the main building block in the GKR protocol [1] which we will learn about in the next lecture.

The Sumcheck protocol proceeds as follows:

1. The prover computes and sends
$$g_1(x) = \sum_{h_2,\ldots,h_m \in H} f(x, h_2, \ldots, h_m).$$
This is a unvariate degree $\leq d$ polynomial where the first variable to f is a free variable.
2. The verifier checks that $g_1(x)$ is a univariate polynomial of degree $\leq d$ and that $\sum_{h_1 \in H} g_1(h_1) = \beta$. (Reject if either check fails.)
3. The verifier sends a uniformly sampled $t_1 \xleftarrow{R} \mathbb{F}$.
4. The prover sends
$$g_2(x) = \sum_{h_3,\ldots,h_m \in H} f(t_1, x, h_3, \ldots, h_m).$$
This is again a univariate degree $\leq d$ polynomial, where the first variable of f has been fixed and the second variable is a free variable.
5. The verifier checks that $g_2(x)$ is degree $\leq d$ and that $\sum_{h_2 \in H} g_2(h_2) = g_1(t_1)$.
6. The verifier sends a uniformly sampled $t_2 \xleftarrow{R} \mathbb{F}$.
7. The prover replies with
$$g_3(x) = \sum_{h_4,\ldots,h_m \in H} f(t_1, t_2, x, h_4, \ldots, h_m).$$
8. The verifier checks that $g_3(x)$ is degree $\leq d$ and that $\sum_{h_3 \in H} g_3(h_3) = g_2(t_2)$.
9. Repeat this procedure on all other variables. The final check will be as follows:
10. The prover sends $g_m(x) = f(t_1, t_2, \ldots, t_{m-1}, x)$.
11. The verifier samples a uniform $t_m \xleftarrow{R} \mathbb{F}$ and checks that $g_m(t_m) = f(t_1, t_2, \ldots, t_m)$ using its oracle access to f. It Accepts if and only if all the checks have passed.

Analysis of the Sumcheck protocol

Completeness. The completeness of this protocol is straightforward so we will focus on soundness.

Soundness. The soundness analysis is "round-by-round". Suppose that the instance is false. Namely suppose the instance is $f : \mathbb{F}^m \to \mathbb{F}$ of degree $\leq d$ in each variable, a set $H \subset \mathbb{F}$ and an element $\beta \in \mathbb{F}$ such that

$$\sum_{h_1,\ldots,h_m \in H} f(h_1, \ldots, h_m) \neq \beta.$$

Fix any cheating prover P^* that tries convince the verifier to accept this false statement. We argue that for each round i, if we start with a false claim of the form

$$g^*_{i-1}(t_{i-1}) = \sum_{h_i,\ldots,h_m \in H} f(t_1, \ldots, t_{i-1} h_i, \ldots, h_m) \tag{1}$$

(where $g^*_0 = \beta$), then the next round claim, which is of the form

$$g^*_i(t_i) = \sum_{h_{i+1},\ldots,h_m \in H} f(t_1, \ldots, t_i, h_{i+1}, \ldots, h_m), \tag{2}$$

is also false with with probability $\geq 1 - \frac{d}{|\mathbb{F}|}$ (assuming the verifier does not reject $g^*_i(\cdot)$). Thus, by a union bound, at the end of the Sumcheck protocol the verifier will reject P^* with probability $\geq 1 - \frac{dm}{|\mathbb{F}|}$. So we get "good" soundness if $|\mathbb{F}| >> dm$. To see why the round-by-round soundness holds note that if $g^*_{i-1}(t_{i-1})$ is false then g^*_i must also be false or else the verifier will reject it. This is the case since the verifier checks that

$$\sum_{h \in H} g^*_i(h) = g^*_{i-1}(t_{i-1}).$$

If g^*_i is false and is of degree d then it agrees with the true polynomial on at most d points, and thus $g^*_i(t)$ on a random $t \xleftarrow{R} \mathbb{F}$ remains incorrect with probability $1 - \frac{d}{|\mathbb{F}|}$.

Communication complexity. The protocol has m rounds of communication, one for each variable of f. In each round, the prover sends one degree-d polynomial, which is represented by d field elements; and the verifier sends one field element t_i. Therefore the communication complexity is $O(dm \log |\mathbb{F}|)$.

Runtime. In each of the m rounds, the verifier evaluates a degree-d polynomial on $|H|$ points; so the verifier runtime is $O(m \cdot |H| \cdot d \cdot \text{polylog}\,|\mathbb{F}|)$. The prover runs in time $O(m \cdot |H|^m \cdot T_f)$, where T_f denotes the time to compute f.

Remark. The Sum-Check protocol has the desirable property that the verifier only sends uniformly sampled field elements in each round (each field element constitutes $\log |\mathbb{F}|$ random bits), namely it is a public-coin protocol.[2] Public-coin protocols are of great interest because as we discussed during the last lecture, we use cryptography to eliminate interaction from such protocols using the Fiat-Shamir transform.

[2] Recall that the ZK proofs that we saw are also public coin.

Why do we care about the Sumcheck protocol?

Beyond being a proof of concept that interactive proofs are powerful, the Sumcheck protocol is extremely important in the design of succinct proof systems. Indeed, the Sumcheck protocol was used by Shamir [4] to construct an interactive proof for any language in PSPACE. We will not show Shamir's protocol, rather we will show an alternative protocol (the GKR protocol [1]) that has efficiency advantages and is conceptually simpler. The main drawback of Shamir's protocol is that to prove the correctness of a time T space-S computation, the runtime of the prover is $\geq 2^{S \cdot log T}$, which may be exponential in T. The runtime of the verifier is proportional to S. This raises the following fundamental question:

Is proving necessarily harder than computing?

Doubly Efficient Interactive Proofs

So far we placed no restriction on the prover's runtime, and restricted only the verifier's runtime. Indeed, when interactive proofs were original defined they referred to the prover as Merlin (an all powerful wizard). In reality, however, we do care about the computational power of the prover. Of course, we still need to allow the prover more computational power than the verifier, as otherwise the prover is not helpful.

Definition 2 (Doubly-Efficient Interactive Proof (DE-IP))**.** A doubly-efficient interactive proof for a language $L \in \mathsf{DTIME}(T(n))$ is an interactive proof such that:

1. The honest prover's runtime is $\text{poly}(T)$.
2. The verifier's runtime is much less, ideally $\text{polylog}(T) + \tilde{O}(n)$, where $\tilde{O}$ omits $\text{polylog}(n)$ factors.

In practice it is desirable that the prover's runtime is $O(T)$.

We will show how to use the Sumcheck protocol to construct a doubly efficient interactive proof for every bounded depth computation.

Theorem 3. *[1] For any circuit C of depth D and size S (that is log-space uniform) there exists a doubly efficient interactive proof such that*

- *The number of rounds is* $D \cdot \text{polylog}(S)$.
- *The communication complexity is* $D \cdot \text{polylog}(S)$.
- *The verifier's runtime is* $O(n) + D \cdot \text{polylog}(S)$ *where n is the input length (assuming the circuit is log-space uniform)*
- *The prover's runtime is* $\text{poly}(S)$.

We will explain what the log-space uniformity condition is when we describe the GKR protocol

The doubly efficient interactive proof that achieves this theorem is referred to in the literature as the GKR protocol. The **only** ingredient used in the GKR protocol is the Sumcheck protocol!

Intuition for the GKR protocol

Given a circuit C of depth D and size S, an input $x \in \{0,1\}^n$ and an output y, the prover needs to convince the verifier that $C(x) = y$. In other words, the verifier wants to catch the prover if y is incorrect. Here is a simple idea: The verifier will ask the prover for the value of the two children corresponding to the output gate, and will check consistency with y. Note that if the values are consistent and y is false, then the value of at lease one of its children must be false. The verifier will guess which one is false randomly, and will continue this process until a leaf x_i is reached. Note that if y is false, and every time the verifier guesses correctly who the false child is, then at the end of this protocol the verifier will recieve a false value of x_i and thus the prover will be rejected!

We assume throughout that the fanin of every gate is ≤ 2.

This interactive proof is extremely simple, but its soundness guarantee is pathetic! The soundness is $1 - 2^{-D}$, since it will catch the prover cheating only if in each and every layer it guesses correctly who the false child is. This happens with probability 2^{-D}. The GKR protocol follows this blue-print but achieves better soundness since it does this over an *error correcting code*.

Analogy to the Sumcheck protocol

Recall that in the Sumcheck protocol the prover convinces the verifier that

$$\sum_{h_1,\ldots,h_m \in H} f(h_1,\ldots,h_m) = \beta.$$

One way to do this is to have the prover send the $|H|$ values $\beta_{1,h}$ where

$$\beta_{1,h} = \sum_{h_2,\ldots,h_m \in H} f(h,h_2,\ldots,h_m).$$

The verifier will check that $\beta = \sum_{h \in H} \beta_{1,h}$ and then will choose at random $h_1^* \xleftarrow{R} H$ and will reduce it to a Sumcheck on $m-1$ variables of the statement

$$\beta_{1,h_1^*} = \sum_{h_2,\ldots,h_m \in H} f(h_1^*,h_2\ldots,h_m) = \beta_1$$

Note that if β is false then β_1 is false with probability $\frac{1}{|H|}$. Therefore, this creates a large loss in soundness (similarly to the simplified GKR protocol presented above). The main idea behind the Sumcheck

protocol is to use the fact that f is of low degree, which intuitively allows the prover to check that all the values in H are correct simultaneously. Specifically, the verifier chooses $t \xleftarrow{R} \mathbb{F}$ and uses the fact that f is a low-degree polynomial to argue that in each round the verifier chooses a "false" t_i with probability $\geq 1 - \frac{d}{|\mathbb{F}|}$. So, by relying on the fact that f is a low-degree polynomial we managed to reduce the loss significantly!

At first it may be unclear how we use this for the GKR protocol since we do not have any low-degree function f there. However, the Sumcheck protocol can be used to prove that

$$\sum_{h_1,\ldots,h_m \in H} f(h_1,\ldots,h_m) = \beta.$$

for *any* $f : H^m \to H$, where f is not necessarily low degree. Namely, we can use the Sumcheck protocol to prove that for *any* function $f : H^m \to H$ it holds that

It may be helpful to focus on the case $H = \{0,1\}$.

$$\sum_{h_1,\ldots,h_m \in H} f(h_1,\ldots,h_m) = \beta.$$

This can be done using the concept called *low-degree extension*.

References

[1] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008.

[2] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[3] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[4] Adi Shamir. Ip=pspace. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 11–15. IEEE Computer Society, 1990.