SumCheck

Notes by 6.5610 Staff

MIT - 6.5610
Lecture 15 (April 1, 2026)

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

Outline

- Motivation
- Interactive proofs (Recap)
- 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 few lectures.

Following our convention that "efficient" means polynomial time, we ask which computations have proofs of correctness that can be verified in polynomial time? This is precisely the definition of the complexity class NP, which is the set of all languages that have membership proofs (aka witnesses) that can be checked by a polynomial-time verifier algorithm, denoted $\mathcal{V}$.

We would like proofs of correctness for languages outside of NP! Specifically, we would like to have a proof of correctness for time T computations that can be verified in time $<< T$ (say time T^{ϵ} or even $\text{polylog}(T)$). 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 earlier in the semester). They realized that zero-knowledge cannot be achieved using classical proofs, and to bypass this barrier they completely changed the way we think about proofs.

They defined a new notion called *interactive proofs*. Such proofs extend upon the classical notion of proofs in two ways. First, rather than solely considering a verifier algorithm $\mathcal{V}$, we instead think of the proof as arising from *interaction* between the verifier $\mathcal{V}$ and a prover algorithm $\mathcal{P}$. Second, we allow the verifier be *randomized* and allow a small (negligible) probability of error.

Remark. In many of our interactive proofs the verifer simply sends its random coins and does not have any private randomness. Such interactive proofs are called **public-coin** interactive proofs. Public-coin protocols are of great interest because as we will see, we can later use cryptography to eliminate interaction from such protocols using the Fiat-Shamir transform (coming up, stay tuned!). We will also cover Groth16, which is not a public-coin protocol.

Both the verifier and prover algorithms will have access to the input of the problem instance. The two algorithms will exchange messages sequentially, computing the next message in the sequence as a function of the messages up to that point. Ultimately, the verifier algorithm will decide whether to accept or reject the problem instance. We can think of the interaction metaphorically as the prover trying to "convince" the verifier of the problem instance being true, and of the verifier trying to verify that the prover is not "dishonest" or "cheating" and misleading the verifier into accepting a false statement.

We will learn about the power of interactive proofs, and their impact on how proofs are designed today. Jumping ahead, 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.

We start by recalling the formal definition of an interactive proof.

Definition

Definition 1 (Interactive Proof system (IP)). An interactive proof system for a language L consists of an interactive p.p.t. verifier algorithm $\mathcal{V}$ and an interactive (possibly inefficient) prover $\mathcal{P}$ algorithm, which exchange a series of *messages*, where we denote the verifier's messages by $r_1, \ldots, r_k$ and the provers messages by $m_1, \ldots, m_k$, where $m_i = \mathcal{P}(x, r_1, m_1, \ldots, r_{i-1}, m_{i-1}, r_i)$, and likewise for $\mathcal{V}$. Denote by $(\mathcal{P}, \mathcal{V}(r))(x) = 1$ the event that the verifier $\mathcal{V}$, with private randomness r, accepts the interactive proof after communicating with the prover $\mathcal{P}$ on the joint input x and assuming $\mathcal{V}$ has randomness r. The following two properties are required to hold:

Notably, the verifier's computations may also depend upon private random bits not revealed to the prover, though we will consider only public-coin interactive proofs, and hence the notation of the verifier's message as $r_1, \ldots, r_k$.

1. *Completeness*: $\forall x \in L$,
$$\Pr[(P, V(r))(x) = 1] \geq \frac{2}{3}$$
2. *Soundness*: $\forall x \notin L$ and $\forall$ (malicious and possibly all powerful) $\mathcal{P}^*$,
$$\Pr[(P^*, V(r))(x) = 1] \leq \frac{1}{3}.$$

Remark. These numbers ($\frac{2}{3}$ and $\frac{1}{3}$) are arbitrary. By repeating the interactive proof λ times and accepting if and only if at least $\frac{\lambda}{2}$ are accepting we can get completeness $1 - \text{negl}(\lambda)$ and soundness $\text{negl}(\lambda)$. This follows from the Chernoff bound, which is a concentration bound that says that if $X_1, \ldots, X_\lambda$ are independent and identically distributed Bernouli random variables such that $\Pr[X_i = 1] = p$ then $\Pr[|\frac{1}{\lambda}\sum_{i\in\lambda} X_i - p| > \delta] \leq 2^{-O(\delta^2 \cdot p \cdot \lambda)}$.

See this for information about the Chernoff bound.

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 [4] 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 [3]. 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 2 (Sumcheck Problem)**.** Given a polynomial $f : \mathbb{F}^m \to \mathbb{F}$ of degree $\leq d$ in each variable and a fixed set $H \subseteq \mathbb{F}$, compute

Often we consider the special case where $H = \{0, 1\}$.

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

Assuming that the verifier has oracle access to f, we will exhibit an interactive proof for the Sumcheck problem. While this problem seems very specific (and possibly not interesting) at first, 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 $\mathsf{IP} = \mathsf{PSPACE}$ result [4], 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 univariate 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 $t_1 \overset{R}{\leftarrow} \mathbb{F}$.

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 $t_2 \overset{R}{\leftarrow} \mathbb{F}$.

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 $t_m \overset{R}{\leftarrow} \mathbb{F}$ 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.

A Simple Example Where All Round Checks Pass but the Final Check Fails

We give a concrete example over a small field showing that a cheating prover can satisfy all per–round checks of the Sumcheck protocol while still being caught in the final oracle check. This illustrates why the final evaluation of f at a random point is essential for soundness.

Setup. Work over the field $\mathbb{F}_7$ and let $H = \{0, 1\}$. Consider the polynomial

$$f(x_1, x_2) = x_1 + x_2.$$

The true sum is

$$S = \sum_{h_1, h_2 \in H} f(h_1, h_2) = 0 + 1 + 1 + 2 = 4 \pmod 7.$$

Round 1 (honest). The prover must send

$$g_1(x_1) = \sum_{x_2 \in H} f(x_1, x_2).$$

A correct prover computes

$$g_1(0) = 1, \qquad g_1(1) = 3.$$

The verifier checks

$$g_1(0) + g_1(1) = 1 + 3 = 4 = S,$$

so the round passes. The verifier samples a random $r_1 = 5 \in \mathbb{F}_7$ and interpolates

$$g_1(5) = 1 + (3 - 1) \cdot 5 = 1 + 10 = 11 \equiv 4 \pmod 7.$$

Round 2 (cheating). The true polynomial in this round is

$$\tilde{g}_2(x_2) = f(5, x_2) = 5 + x_2,$$

so $\tilde{g}_2(0) = 5$ and $\tilde{g}_2(1) = 6$. A cheating prover instead sends the degree–1 polynomial

$$g_2(x_2) = 4x_2.$$

This is *not* equal to $\tilde{g}_2$, but it satisfies the required sum condition:

$$g_2(0) + g_2(1) = 0 + 4 = 4 = g_1(5).$$

Thus the verifier's degree check and sum–consistency check both pass.

The verifier samples $r_2 = 3$.

Final check (failure). The prover claims

$$g_2(3) = 4 \cdot 3 = 12 \equiv 5 \pmod 7.$$

However, the verifier evaluates the true polynomial:

$$f(5, 3) = 5 + 3 = 8 \equiv 1 \pmod 7.$$

Since $5 \neq 1$ in $\mathbb{F}_7$, the final check fails and the verifier rejects.

Conclusion. This example shows that even if a cheating prover satisfies all intermediate round checks (degree bounds and sum consistency), the prover can still be caught in the final oracle check. The final evaluation of f at a random point is therefore essential for the soundness of the Sumcheck protocol.

Analysis of the Sumcheck protocol

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

Setup and notation. Let F be a finite field and let $f(X_1, \ldots, X_m) \in F[X_1, \ldots, X_m]$ be a polynomial with $\deg_{X_j}(f) \leq d$ for every $j \in \{1, \ldots, m\}$. Fix a subset $H \subseteq F$ (in the usual Boolean case $H = \{0, 1\}$). The Sum-Check protocol is used to verify a claimed value

$$H_{\text{claim}} \overset{?}{=} S := \sum_{h_1, \ldots, h_m \in H} f(h_1, \ldots, h_m).$$

Theorem (Soundness): Suppose the prover and verifier run the Sum-Check protocol over the field F with per-variable degree bound d. If the claimed sum $H_{\text{claim}} \neq S$, then a (possibly malicious) prover can cause the verifier to accept with probability at most

$$\frac{m \cdot d}{|F|}.$$

In particular, for multilinear f (so $d = 1$) the soundness error is at most $m/|F|$.

Proof. We follow the standard round-by-round argument.

True partial-sum polynomials. Define the *true* partial-sum univariate polynomials inductively as follows. For the first variable set

$$\tilde{g}_1(X_1) := \sum_{x_2,\dots,x_m \in H} f(X_1, x_2, \dots, x_m),$$

and for $j \geq 2$, after the verifier has sampled (and fixed) values $r_1, \dots, r_{j-1} \in F$, define

$$\tilde{g}_j(X_j) := \sum_{x_{j+1},\dots,x_m \in H} f(r_1, \dots, r_{j-1}, X_j, x_{j+1}, \dots, x_m).$$

Each $\tilde{g}_j$ is a univariate polynomial with $\deg(\tilde{g}_j) \leq d$.

Behavior of a cheating prover. Let the prover send polynomials $g_1, g_2, \dots, g_m$ (these are the messages the prover supplies in each round). If for some round j the verifier finds $g_j(0) + g_j(1) + \cdots + g_j(h)$ (sum over H) not equal to the previously checked value, the verifier rejects immediately. Thus, any cheating prover that reaches the final check must ensure that all the per-round sum equalities hold up to the final round.

Assume $H_{\text{claim}} \neq S$. Consider the smallest index $t \in \{1, \dots, m\}$ such that the prover's sent polynomial g_t is not identical (as a polynomial) to the true polynomial $\tilde{g}_t$. Such a t must exist: otherwise the equalities checked in every round would force $H_{\text{claim}} = S$, contradicting the assumption.

Detecting a disagreement at round t. Define the difference polynomial

$$h(X) := g_t(X) - \tilde{g}_t(X).$$

By choice of t we have $h \not\equiv 0$, and $\deg(h) \leq d$. The verifier samples r_t uniformly from F at round t. The protocol will detect the disagreement at round t unless $h(r_t) = 0$. Since a nonzero univariate polynomial of degree at most d has at most d roots in F, the probability (over the verifier's choice of r_t) that $h(r_t) = 0$ is at most $d/|F|$.

Union bound over rounds. The prover could attempt to hide disagreements in any of the m rounds. Applying a union bound over the m rounds, the probability that *all* disagreements (if any) evade detection is at most

$$\frac{m \cdot d}{|F|}.$$

Therefore the verifier rejects with probability at least $1 - md/|F|$, and accepts a false claim with probability at most $md/|F|$, as claimed. $\square$

Remark. For the common multilinear case $d = 1$ and a large field (e.g., a 128-bit prime), the soundness error $m/|F|$ is negligible.

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.

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 [5] 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 $\geq 2^{S \cdot log S}$, 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 originally 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 3 (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: 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 called the GKR protocol [1]. The **only** ingredient used in the GKR protocol is the Sumcheck protocol!

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] Joe Kilian. A note on efficient interactive proofs. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 484–490, New York, NY, USA, 1992. ACM.

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

[5] 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.