Lecture 21: The GKR Protocol

Notes by Yael Kalai

MIT - 6.5620

Lecture 21 (November 24, 2025)

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

Outline

- Warmup for the GKR protocol
- Low-degree Extension
- The GKR protocol

Recap

- Sumcheck protocol
- Doubly efficient interactive proofs

Sumcheck Protocol

For the sake of simplicity, we state the properties of the Sumcheck protocol over the hypercube $\{0,1\}^\ell$ (as opposed to over H^ℓ for an arbitrary set H).

Theorem 1. *There exists an interactive proof, called the Sumcheck protocol, for proving that*

$$\sum_{b_1,\ldots,b_\ell \in \{0,1\}} f(b_1,\ldots,b_\ell) = v$$

where f is an ℓ-variate polynomial of degree $\leq d$ in each variable. The verifier is given only oracle access to f and the prover is given the description of f. It has the following guarantees:

1. **Completeness:** *The honest prover is accepted with probability* 1.
2. **Soundness:** *Every P^* convinces V to accept a false statement with probability $\leq \frac{m \cdot d}{|\mathbb{F}|}$.*
3. **Complexity:** *The number of rounds is ℓ. The verifier makes a single oracle access to the oracle f and runs in time $\ell \cdot d \cdot \text{polylog}(|\mathbb{F}|)$. The prover runs in time $O(T_f \cdot 2^\ell \cdot \ell)$.*

Doubly Efficient Interactive Proofs

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 $O(T)$.

Today we will see how to use the Sumcheck protocol to construct a doubly efficient interactive proof for every *bounded depth* computation.

Theorem 3. *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 $\tilde{O}(n) + D \cdot \text{polylog}(S)$ 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!

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

$$\sum_{b_1,\ldots,b_m \in \{0,1\}} f(b_1,\ldots,b_m) = v.$$

One way to do this is to have the prover send the two values v_0 and v_1 where

$$v_b = \sum_{b_2,\ldots,b_m \in \{0,1\}} f(b, b_2,\ldots,b_m).$$

The verifier will check that $v = v_0 + v_1$ and then will choose at random $b_1^* \xleftarrow{R} \{0,1\}$ and will reduce it to a Sumcheck on $\ell - 1$ variables of the statement

$$v_{b_1^*} = \sum_{b_2,\ldots,b_\ell \in \{0,1\}} f(b_1^*, b_2 \ldots, b_\ell)$$

Note that if v is false then $v_{b_1^*}$ is false with probability $\frac{1}{2}$. 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 both the values in $\{0,1\}$ 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

$$\sum_{b_1,\ldots,b_\ell \in \{0,1\}} f(b_1,\ldots,b_\ell) = v.$$

for *any* $f : \{0,1\}^\ell \to \{0,1\}$, that has no algebraic structure, as long as the verifier is given oracle access to the *multi-linear extension* of f, defined below.[1]

[1] Oracle access to any *low-degree extension* suffices, where the runtime grows with the degree.

Multi-linear Extension

A multi-linear extension (MLE) of any function (not necessarily a polynomial) $f : \{0,1\}^m \to \{0,1\}$,[2] is a *polynomial* function

[2] One can think of f as an arbitrary string of length 2^m.

$$\tilde{f} : \mathbb{F}^m \to \mathbb{F}$$

that is linear in each variable that agrees with f on all inputs in $\{0,1\}^m$; i.e., $\forall x \in \{0,1\}^m, \tilde{f}(x) = f(x)$, or more concisely $\tilde{f} \mid_{\{0,1\}^m} \equiv f$. Notice that the domain of $\tilde{f}$ is $\mathbb{F}^m$, a superset of $\{0,1\}^m$, hence the name "extension."

Theorem 4. *Let $f : \{0,1\}^m \to \{0,1\}$ be any function (i.e., an arbitrary sequence of 2^m bits). Let $\mathbb{F}$ be any finite field. Then there exists a unique multi-linear function $\tilde{f} : \mathbb{F}^m \to \mathbb{F}$ s.t. $\tilde{f} \mid_{\{0,1\}^m} \equiv f$. Moreover, $\tilde{f}$ can be computed in time $2^m \cdot \text{poly}(m)$.*

This theorem is a generalization of Langrange interpolation to the multi-variate setting (though focusing on the linear setting).

Proof. Let

$$\tilde{f}(x_1, \ldots, x_m) = \sum_{(b_1,\ldots,b_m)\in\{0,1\}^m} f(b_1, \ldots, b_m) \cdot \chi((b_1, \ldots, b_m), (x_1, \ldots, x_m))$$

where χ is a multi-linear function that satisfies that for every $x_1, \ldots, x_m \in \{0,1\}$ and every $b_1, \ldots, b_m \in \{0,1\}$,

$$\chi((b_1, \ldots, b_m), (x_1, \ldots, x_m)) = \begin{cases} 1 & (x_1, \ldots, x_m) = (b_1, \ldots, b_m) \\ 0 & (x_1, \ldots, x_m) \neq (b_1, \ldots, b_m) \end{cases}$$

χ is defined as follows:

$$\chi((b_1, \ldots, b_m), (x_1, \ldots, x_m)) = \prod_{i=1}^{m} \chi(b_i, x_i)$$

where

$$\chi(b, x) = 1 - b - x + 2 \cdot b \cdot x \tag{1}$$

To prove uniqueness we need to argue that if two multi-linear polynomials on m variables agree on $\{0,1\}^m$ then they must be equal. Equivalently, we need to prove that if a mutli-linear polynomial on m variables is 0 on the hypercube $\{0,1\}^m$ then it must be the zero polynomial. This can be proved by induction on m.

Base case: $m = 1$. A linear function is of the form $f(x) = ax + b$. It is easy to see that $f(0) = 0$ implies that $b = 0$. Similarly $f(1) = 0$ implies that $a + b = 0$. Thus, f is identically 0.

The induction step: Suppose uniqueness holds for $m - 1$ variables and we will prove that it holds for m variables. One can write

$$f(x_1, \ldots, x_m) = A(x_1, \ldots, x_{m-1}) + x_m \cdot B(x_1, \ldots, x_{m-1}).$$

Note that for every $x_1, \ldots, x_{m-1} \in \{0,1\}$,

$$0 = f(x_1, \ldots, x_{m-1}, 0) = A(x_1, \ldots, x_{m-1}),$$

which by our induction hypothesis implies that $A \equiv 0$. In addition, for every $x_1, \ldots, x_{m-1} \in \{0,1\}$,

$$0 = f(x_1, \ldots, x_{m-1}, 1) = A(x_1, \ldots, x_{m-1}) + B(x_1, \ldots, x_{m-1}) = B(x_1, \ldots, x_{m-1}),$$

which by our induction hypothesis implies that $B \equiv 0$. Thus $f \equiv 0$, as desired. □

The GKR protocol

Fix boolean circuit $C : \{0,1\}^n \to \{0,1\}$ of size (number of gates) S and depth D. The GKR protocol is an interactive proof for the fact that $C(x) = 1$. We assume that the verifier has a succinct description of C. Formally, we assume that C is log-space uniform i.e. it can be generated by some log-space Turing Machine M,[3] and we assume that the verifier has a description of M. Assume without loss of generality that C is layered which means that each gate belongs to a layer, and each gate in layer i is connected by neighbors only in layer $i+1$. Let layer 0 denotes the output layer and D denotes the input layer.

[3] The need for this uniformity condition will be explained below

One can always layer a circuit by adding dummy intermediate gates. This can be done while increasing the depth to depth at most D^2.

Recall the intuitive protocol above, where we reduce a claim about the value of a gate in layer i to a claim about the value of a gate in layer $i+1$. The GKR protocol follows this blueprint. The protocol consists of D-subprotocols, where a claim about the (joint) values of gates in layer i is converted to a claim about the (joint) values of gates in layer $i+1$. Eventually, it will be reduced to a claim about the input layer (layer D), which is known to the verifier.

Detailed description of the protocol

Step 1: Arithmetize C. Convert C to a (layered) arithmetic circuit (over $\mathsf{GF}[2]$) with fan-in 2. Arithmetic circuit (over $\mathsf{GF}[2]$) means that it consists only of gates of the form ADD and MULT (where addition and multiplication are done modulo 2). We can convert any Boolean circuit, with gates $\wedge$ and $\neg$, into an arithmetic circuit, by converting a gate $\wedge$ into a gate MULT, and converting a gate $\neg$ into a gate ADD where we add a constant 1 as an input to the gate.

Step 2: We assume for simplicity, and without loss of generality, that each layer has exactly S gates and that S is a power of 2, i.e., $S = 2^m$ for some integer m. This can be done by adding dummy gates.

We can give each of the S gates in a given layer a unique label encoded in $\{0,1\}^m$.

Step 3: The prover computes the values of all gates in every layer of the circuit. For layer i, define the function $V_i : \{0,1\}^m \to \{0,1\}$ as the mapping from an encoding of a gate label to the value of the gate. Let $\tilde{V}_i : \mathbb{F}^m \to \mathbb{F}$ be its multi-linear extension (MLE), which is the unique multi-linear function that agrees with V_i on inputs in $\{0,1\}^m$.

The Protocol: The protocol consists with D "reduction" protocols, where each reduction protocol reduces a claim of the form $\tilde{V}_i(z_i) = v_i$ about layer i to a claim of the form $\tilde{V}_{i+1}(z_{i+1}) = v_{i+1}$ about about layer $i+1$. We start with the output layer where the prover claims that $\tilde{V}_0(z_0) = v_0 = 1$ where $z_0 \in \{0,1\}^m$ is the label of the only non-dummy gate in layer 0 that holds the output of the circuit. At the end of these D reduction protocols, we will be left with a claim of the form $\tilde{V}_d(z_d) = v_d$. The verifier can check this on its own since it knows V_d from its input values and thus can compute its multi-linear extension on its own.

Actually, the reduction protocol will reduce checking two such claims about layer i to two such claims about layer $i+1$.

The reduction protocol

For every $i \in [D]$ we define two functions $\mathsf{ADD}_i, \mathsf{MULT}_i : (\{0,1\}^m)^3 \to \{0,1\}$ as follows:

$$\mathsf{ADD}_i(p, w_1, w_2) = \begin{cases} 1 & \text{gate } p \text{ in layer } i \text{ is an ADD gate connecting } w_1 \text{ and } w_2 \text{ in layer } i+1 \\ 0 & \text{Otherwise} \end{cases}$$

MULT_i is defined similarly with ADD replaced with MULT in the definition. Let

$$\widetilde{\mathsf{ADD}}_i, \widetilde{\mathsf{MULT}}_i : \mathbb{F}^{3m} \to \mathbb{F}$$

be the multi-linear extensions of ADD_i and MULT_i, respectively.

We can expand out the MLE definition and rewrite the claim $\tilde{V}_i(z_i) = v_i$ as

$$v_i = \tilde{V}_i(z_i) = \sum_{p \in \{0,1\}^m} V_i(p) \cdot \chi(p, z_i)$$

Then we can further express $V_i(p)$ as a combination of $\widetilde{\mathsf{ADD}}_i, \widetilde{\mathsf{MULT}}_i$:

$$v_i = \sum_{p \in \{0,1\}^m} \sum_{w_1, w_2 \in \{0,1\}^m} \Big[\widetilde{\mathsf{ADD}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) + \tilde{V}_{i+1}(w_2)) + \widetilde{\mathsf{MULT}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) \cdot \tilde{V}_{i+1}(w_2)) \Big] \chi(p, z_i)$$

This is exactly a claim that one can run Sumcheck on.
Denote the polynomial inside the sum by

$$f(p, w_1, w_2) = \Big[\widetilde{\mathsf{ADD}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) + \tilde{V}_{i+1}(w_2)) + \widetilde{\mathsf{MULT}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) \cdot \tilde{V}_{i+1}(w_2))\Big]\chi(p, z_i)$$

Thus after the last round in Sumcheck for

$$v_i = \sum_{p, w_1, w_2 \in H^m} f(p, w_1, w_2),$$

the verifier must be able to compute $f(z_0, z_1, z_2)$ for some random $z_0, z_1, z_2 \in \mathbb{F}^m$ chosen by the verifier. We assume for now that the verifier can compute on its own $\widetilde{\mathsf{ADD}}_i$ and $\widetilde{\mathsf{MULT}}_i$. This is precisely where we use the log-space uniformity condition of the underlying circuit C.

So we reduced checking the value v_i in layer i to checking the value of two elements in round $i+1$. It seems like if we continue in this way the number of elements we will need to check will grow exponentially! However, surprisingly this is not the case! We can reduce checking two elements in round $i+1$ to checking two elements in round $i+2$.

The idea is to run two Sumcheck protocols (one for each element) but where the verifier uses the same randomness in both these Sumcheck protocols! This will convert checking two elements in round i to checking two elements in round $i+1$.

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.