**The GKR Protocol**

*Notes by Yael Kalai*

*MIT - 6.5610*

*Lecture 12 (March 13, 2024)*

---

**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

---

**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 least 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 the end of this protocol the verifier will receive a false value of $x_i$ and thus the prover will be rejected!

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 blueprint but achieves better soundness since it does this over a (polynomial) error correcting code.

We assume throughout that the fanin of every gate is $\leq 2$. 
**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 first send the function

\[ g_1(x) = \sum_{h_2, \ldots, h_m \in H} f(x, h_2, \ldots, h_m). \]

The verifier will then choose at random \( h_1^* \in H \) and will reduce it to a Sumcheck on \( m - 1 \) variables of the statement

\[ \sum_{h_2, \ldots, h_m \in H} f(h_1^*, h_2, \ldots, h_m) = \beta_1^* \]

where \( \beta_1^* = g_1(h_1^*) \). Note that if \( \beta^* \) is false then \( \beta_1^* \) is false with probability \( \frac{1}{|H|} \). This creates a large loss in soundness, and the 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 \( t \leftarrow R_F \) 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 \( \geq 1 - \frac{d}{|F|} \). 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

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

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

**Low-Degree Extension**

A low degree extension (LDE) of a function (not necessarily a polynomial) \( f : H^m \to \{0, 1\} \) is a polynomial function \( \tilde{f} : F^m \to F \) of (minimal) degree \( \leq |H| - 1 \) in each variable that agrees with \( f \) on all inputs in the range \( H^m \); i.e., \( \forall h \in H^m, \tilde{f}(h) = f(h) \), or more concisely \( \tilde{f} \mid_{H^m} \equiv f \). Notice that the domain of \( \tilde{f} \) is \( F^m \), a superset of \( H^m \), hence the name “extension”. It may be helpful to focus on the case \( H = \{0, 1\} \).
Theorem 1. Let \( f : H^m \rightarrow \{0, 1\} \) be any function (i.e., an arbitrary sequence of \(|H|^m\) bits). Let \( \mathbb{F} \) be any finite field s.t. \( H \subseteq \mathbb{F} \). Then there exists a unique \( \tilde{f} : \mathbb{F}^m \rightarrow \mathbb{F} \) s.t. \( \tilde{f} |_{H^m} \equiv f \) and the degree of every variable in \( \tilde{f} \) is at most \(|H| - 1\).

This theorem is a generalization of Langrange interpolation to the multi-variate setting. When \( m = 1 \), the LDE \( \tilde{f} : \mathbb{F} \rightarrow \mathbb{F} \) is a univariate polynomial, which follows from Lagrange Interpolation.

The univariate case: For any \( f : H \rightarrow \{0, 1\} \), we can write

\[
\tilde{f}(x) = \sum_{h \in H} f(h) \chi_h(x)
\]

where \( \chi_h \) satisfies that for every \( x \in H \):

\[
\chi_h(x) = \begin{cases} 
1 & x = h \\
0 & x \neq h
\end{cases}
\]

From Lagrange interpolation, we have an explicit formula for \( \chi_h(\cdot) \) as a degree \(|H| - 1\) polynomial function of \( x \):

\[
\chi_h(x) = \prod_{h' \in H \setminus \{h\}} \frac{h' - x}{h' - h} \quad (1)
\]

The LDE theorem (Theorem 1) is a multi-variate version of this.

The multivariate case: For any \( f : H^m \rightarrow \{0, 1\} \), we can write

\[
\tilde{f}(x_1, \ldots, x_m) = \sum_{(h_1, \ldots, h_m) \in H^m} f(h_1, \ldots, h_m) \chi_{h_1,\ldots,h_m}(x_1, \ldots, x_m)
\]

where \( \chi_{h_1,\ldots,h_m} \) satisfies that for every \( x \in H^m \):

\[
\chi_{h_1,\ldots,h_m}(x_1, \ldots, x_m) = \begin{cases} 
1 & (x_1, \ldots, x_m) = (h_1, \ldots, h_m) \\
0 & (x_1, \ldots, x_m) \neq (h_1, \ldots, h_m)
\end{cases}
\]

The explicit formula for \( \chi_{h_1,\ldots,h_m} \) is

\[
\chi_{h_1,\ldots,h_m}(x_1, \ldots, x_m) = \prod_{i=1}^{m} \chi_{h_i}(x_i). \quad (2)
\]

The GKR protocol

Fix boolean circuit \( C : \{0,1\}^n \rightarrow \{0,1\} \) 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 $M$. and the verifier has $M$. 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 denote the output layer and $D$ denote the input layer.

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 values (layer $d$), which are known to the verifier.

**Detailed description of the protocol**

**Step 1: Arithmetize $C$.** Convert $C$ to a (layered) arithmetic circuit (over $\mathbb{GF}[2]$) with fan-in 2. Arithmetic circuit (over $\mathbb{GF}[2]$) 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 $\land$ and $\neg$, into an arithmetic circuit, by converting a gate $\land$ into a gate $\text{MULT}$, and converting a gate $\neg$ into a gate ADD where we add a constant 1 as an input to the gate.

**Step 2:** Pick a subset $H \subseteq \mathbb{F}$ and an integer $m$ such that $S = |H|^m$. This way, we can give each of the $S$ gates in a given layer a unique label encoded in $H^m$. A nature choice is

$$H = \{0, 1\} \quad \text{and} \quad m = \log S.$$  

A less natural choice but a common one is

$$H = \{0, 1, \ldots, \log S - 1\} \quad \text{and} \quad m = \frac{\log S}{\log \log S}.$$  

Jumping ahead the reason the latter choice is that one can take a field $\mathbb{F}$ that contains $H$ of size $\text{poly}(|H|)$ so that

$$|\mathbb{F}| >> m \cdot |H| \quad \text{and} \quad |\mathbb{F}^m| = \text{poly}(S). \quad (3)$$  

This cannot be done with the natural choice of $H = \{0, 1\}$ since then we need to take $|\mathbb{F}| \geq \log S$, which results in $|\mathbb{F}^m| \geq S^{\log \log S}$ which is super-polynomial.

**Step 3:** The prover computes the values of all gates in every layer of the circuit. For layer $i$, define the function $V_i : H^m \rightarrow \{0, 1\}$ as the mapping from an encoding of a gate label to the value of the gate, and $\tilde{V}_i : \mathbb{F}^m \rightarrow \mathbb{F}$ be its low-degree extension (LDE), which is the

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


unique function of degree \( \leq |H| - 1 \) in each variable that agrees with \( V_i \) on inputs in \( H^m \).

**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 layer \( i + 1 \). We start with the output layer where the prover claims

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

\( \tilde{V}_0(z_0) = v_0 = 1 \) where \( z_0 \in H^m \) 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 can compute its LDE on its own as follows:

\[
\tilde{V}_d(z_d) = \sum_{h_1, \ldots, h_m \in H} V_d(h_1, \ldots, h_m) \chi_{h_1, \ldots, h_m}(z_d)
\]

\[
= \sum_{h_1, \ldots, h_m \in H} x_d \chi_{h_1, \ldots, h_m}(z_d)
\]

\[
= \sum_{h_1, \ldots, h_m \in H} x_i \prod_{j=1}^m \chi_{h_j}(z_d, i)
\]

where \( x_i \) is the \( i \)-th bit of the input \( x \) and \( i \in [n] \) is the index corresponding to the label \((h_1, \ldots, h_m)\).

**The reduction protocol**

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

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

\( \text{MULT}_i \) is defined similarly with \( \text{ADD} \) replaced with \( \text{MULT} \) in the definition. Let

\[
\tilde{\text{ADD}}_i, \tilde{\text{MULT}}_i : F^{3m} \to F
\]

be the LDEs of \( \text{ADD}_i \) and \( \text{MULT}_i \), respectively.

We can expand out the LDE definition and rewrite the claim

\( \tilde{V}_i(z_i) = v_i \) as

\[
v_i = \tilde{V}_i(z_i) = \sum_{p \in H^m} V_i(p) \chi_p(z_i)
\]

Then we can further express \( V_i(p) \) as combination of \( \tilde{\text{ADD}}_i, \tilde{\text{MULT}}_i \):

\[
v_i = \sum_{p \in H^m} \sum_{w_1, w_2 \in H^m} \left[ \tilde{\text{ADD}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) + \tilde{V}_{i+1}(w_2)) + \tilde{\text{MULT}}_i(p, w_1, w_2)(\tilde{V}_{i+1}(w_1) \cdot \tilde{V}_{i+1}(w_2)) \right] \chi_p(z_i)
\]

This almost looks like a claim that one can run Sum-Check on. However, recall that the Sum-Check protocol only works if the expression inside the sum is a low-degree (multi-variate) polynomial, so we need to argue that \( \chi_p(z_i) \) is a low-degree polynomial in \( p \).

Indeed, it turns that that we can compute \( \chi_p(z_i) \) via a low-degree polynomial, as follows. Define \( \beta : H \times H \rightarrow \{0,1\} \) as \( \beta(a,b) = \chi_a(b) \), and let \( \tilde{\beta} : F \times F \rightarrow F \) be its LDE. For every fixed \( a \in H \), we have that the univariate polynomial \( \tilde{\beta}(a,\cdot) \) is a LDE/Lagrange Interpolation of \( \beta(a,\cdot) \). In addition, \( \chi_a(\cdot) \) is also a LDE of \( \beta(a,\cdot) \). Then by the uniqueness of LDE, we have that \( \tilde{\beta}(a,b) = \chi_a(b) \) for all \( a \in H, b \in F \).

We can thus rewrite \( v_i \) as

\[
v_i = \sum_{p,w_1,w_2 \in H^m} \left[ \text{ADD}_i(p,w_1,w_2)(\tilde{V}_{i+1}(w_1) + \tilde{V}_{i+1}(w_2)) + \text{MULT}_i(p,w_1,w_2)(\tilde{V}_{i+1}(w_1) \cdot \tilde{V}_{i+1}(w_2)) \right] \tilde{\beta}(p,z_i) \]

Denote the polynomial inside the sum by

\[
f_i(z_i) = \left[ \text{ADD}_i(p,w_1,w_2)(\tilde{V}_{i+1}(w_1) + \tilde{V}_{i+1}(w_2)) + \text{MULT}_i(p,w_1,w_2)(\tilde{V}_{i+1}(w_1) \cdot \tilde{V}_{i+1}(w_2)) \right] \tilde{\beta}(p,z_i) \]

Thus after the last round in Sum-Check for

\[
v_i = \sum_{p,w_1,w_2 \in H^m} f_i(z_i(p,w_1,w_2)),\]

the verifier must be able to compute \( f_i(z_{i+1,0},z_{i+1,1},z_{i+1,2}) \) for some random \( z_{i+1,0},z_{i+1,1},z_{i+1,2} \in F^m \) chosen by the verifier. We assume for now that the verifier can compute on its own ADD_i and MULT_i. This is precisely where we use the log-space uniformity condition of the underlying circuit.

So we reduced checking the value \( v_i \) in layer 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