Lecture 06
Inefficiency of Honest Provers

Our focus so far: achieve a polynomial-time verifier.

What about the honest prover?

Say we are given a boolean formula \( \phi(x_1, \ldots, x_n) \).
- in the sumcheck protocol (for #SAT): \( \text{time}(P) = \Omega(2^n / \phi) \).
- in Shamir's protocol (for TQBF): \( \text{time}(P) = \Omega(2^n / \phi) \).

[In fact, Shamir's original protocol, without Shor's simplification, reduced the QBF to a "simple" QBF, squaring #vars so \( \text{time}(P) = O(2^{n^2 / \phi}) \).]

Are these times useful for computations of interest?

Let \( M \) be a machine running in time \( T \) and space \( S \), and define

\[ L_M := \{ x \mid N(x) = 1 \} \]

The reduction from \( L_M \) to TQBF yields a boolean formula \( \phi(x_1, \ldots, x_n) \) with

\[ n \geq (\log T) \cdot S \]

Even if \( T, S = \text{poly}(n) \), the honest prover runs in time \( \Omega(2^n) \approx \Omega(T^S) = n^w(1) \).
Doubly-Efficient Interactive Proofs

New goal: additionally restrict honest prover to run in polynomial time.
We call this a doubly-efficient interactive proof (deIP).

claim: deIP ≤ BPP

proof: The probabilistic algorithm simulates the interaction between the honest prover and the honest verifier.

To make deIP non-trivial, we require the verifier to work less than deciding the language alone (e.g. less space, less time, ...).
This setting can be viewed as delegation of computation:

Q: what languages have doubly-efficient interactive proofs?
Delegation for Bounded-Depth Circuits

**Theorem.** Suppose that $L$ is decidable by $O(\log s)$-space uniform circuits of size $S$ and depth $D$. Then $L$ has a public-coin IP s.t.
- prover time is $\text{poly}(S)$
- verifier time is $(n+D)\cdot \text{polylog} S$ [& space is $O(\log s)$]
- communication (and # rounds) is $D \cdot \text{polylog}(S)$.

**Note:** a circuit family $\{C_n\}_{n \in \mathbb{N}}$ is $S$-space uniform if there exists a machine $M$ s.t. $M(1^n) = C_n$ runs in space $O(S(n))$.

The proof of the theorem is quite technical.
We will see one piece: the "bare bones" protocol, which is same as above except that the verifier has oracle access to information about the circuit's topology (it will make $O(D)$ calls to this oracle) which saves us from discussing uniformity.

Main tools for bare-bones protocol: this has been implemented and it is very efficient!
- mon arithmetization, more sumcheck, some new ideas.
Layered Arithmetic Circuits

A layered arithmetic circuit \( C : \mathbb{F}^n \to \mathbb{F} \) of size \( S \) and depth \( D \) (with \( n \leq S \)) is an arithmetic circuit with fan-in 2 arranged in \( D+1 \) layers:

- Layer 0: 1 output \( V_0 \in \mathbb{F} \)
- Layer 1: \( S \) internal values \( V_1 : [S] \to \mathbb{F} \)
- Layer 2: \( S \) internal values \( V_2 : [S] \to \mathbb{F} \)
- ... (Continuing in this manner)
- Layer \( D \): \( n \) inputs \( V_0 : [n] \to \mathbb{F} \)

The wiring predicates \( [(\text{add}_i, \text{mul}_i)]_{i=1, \ldots, D} \) describe the circuit \( C: \text{add}_i/\text{mul}_i \) at \((a, b, c)\) is 1 if the \( a \)-th value in layer \( i-1 \) is the addition/multiplication of the \( b \)-th & \( c \)-th values in layer \( i \).

For notational simplicity, we assume \( C \) has 1 type of gate: \( g: \mathbb{F}^2 \to \mathbb{F} \). We can take \((w_1, \ldots, w_D)\) to be the wiring predicates for \( g \). [Extending to multiple gate types is straightforward]
Low-Degree Extension

Let $H \subseteq F$ be a domain, and $f : H \rightarrow F$ a function.

A polynomial $p \in F[x]$ is an extension of $f$ if $p|_H \equiv f$.

It is a low-degree extension if $p$ has "low degree" [the specific condition varies].

The higher the allowed degree, the more low-degree extensions a function has.

The minimal degree one is unique: it has degree $< |H|$ and equals

$$p(x) = \sum_{\alpha \in H} f(\alpha) \cdot L_{H,\alpha}(x) = \sum_{\alpha \in H} f(\alpha) \cdot \left( \prod_{\beta \in H \setminus \{\alpha\}} \frac{x - \beta}{\alpha - \beta} \right).$$

The polynomials $\sum_{\alpha \in H} L_{H,\alpha}(x)^2$ are the Lagrange polynomials.

The multivariate case is straightforward:

- $p \in F[x_1, \ldots, x_n]$ extends $f : H^n \rightarrow F$ if $p|_{H^n} \equiv f$.

- $f$'s extension of minimal degree has individual degree $< |H|$ and equals

$$p(x_1, \ldots, x_n) = \sum_{\alpha_1, \ldots, \alpha_n \in H} f(\alpha_1, \ldots, \alpha_n) \cdot L_{H^n, \alpha_1, \ldots, \alpha_n}(x_1, \ldots, x_n) = \sum_{\alpha_1, \ldots, \alpha_n \in H} f(\alpha_1, \ldots, \alpha_n) \cdot \left( \prod_{i \in \{1, \ldots, n\}} L_{H, \alpha_i}(x_i) \right).$$
Arithmetize Each Layer

Fix a subset \( H \subseteq F \) of size \( \log S \) and set \( m := \frac{\log S}{\log |H|} \) and \( m^* := \frac{\log n}{\log |H|} \). This induces bijections \([S] \leftrightarrow H^m\) and \([n] \leftrightarrow H^{m^*}\).

**Step 1:** rewrite computation as summations

Let \( z \in F^n \) be an input to the circuit \( C : F^n \rightarrow F \).

- the input layer \( V_0 : H^{m^*} \rightarrow F \) is defined as \( V_0(a) := z_a \)
- for \( i = D-1, \ldots, 1, 0 \) : \( V_i : H^m \rightarrow F \) is defined as \( V_i(a) := \sum_{b,c \in H^m} w_{pi}(a,b,c) \cdot g(V_{i+1}(b), V_{i+1}(c)) \)

**Step 2:** low-degree extend each layer

- the extension of the input layer is \( \hat{V}_0 : F^{m^*} \rightarrow F \) where
  \[
  \hat{V}_0(x) := \sum_{a \in H^{m^*}} z_a \cdot L_{H^{m^*}, a}(x).
  \]
- the extension of the \( i \)-th layer is \( \hat{V}_i : F^m \rightarrow F \) where
  \[
  \hat{V}_i(x) := \sum_{a \in H^m} \left( \sum_{b,c \in H^m} \hat{V}_{i+1}(a,b,c) \cdot g(V_{i+1}(b), V_{i+1}(c)) \right) \cdot L_{H^m, a}(x).
  \]

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**can consider extension instead of function as summation is over \( H \)**

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**can consider extension instead of function as summation is over \( H \)**

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**can consider extension instead of function as summation is over \( H \)**

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**can consider extension instead of function as summation is over \( H \)**

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.

**can consider extension instead of function as summation is over \( H \)**

**Step 3:** replace \( L_{H^m, a}(x) \) with \( I_{H^m}(x,a) \) where \( I_{H^m}(x,y) := \prod_{i=1}^{m^*} \sum_{a \in H^m} L_{H^m, a}(x_i) L_{H^m, a}(y_i) \) to ensure that \( a \) has low degree in addend.
Rewrite Computation as Iterated Sumchecks

The statement "C(z) = y" is rewritten as "\( \hat{\nu}_0(0) = y \)."

Equivalently, \( \sum_{a,b,c \in \mathbb{H}} \hat{w}_1(a,b,c) \cdot g(\hat{\nu}_1(b), \hat{\nu}_1(c)) \cdot I_{H^m}(0,a) = y \).

So we can do a sumcheck on variables for \( a,b,c \). This involves:

- 3m rounds [we are summing over the hypercube \( H^{3m} \)]
- Soundness error \( O(m, \frac{1}{|H|}) \) [individual degrees are \( O(|H|) \) so we pay \( O(1/|H|) \) per round]
- \( \text{poly}(|H|^m) = \text{poly}(S) \) operation for the honest prover — THIS IS EFFICIENT
- \( \text{poly}(m, |H|) = \text{poly}(\log S) \) operations for the verifier, given
  - 1 query to \( \hat{w}_1 : \mathbb{F}^m \rightarrow \mathbb{F} \) & assume that verifier can evaluate on its own
  - 2 queries to \( \hat{\nu}_1 : \mathbb{F}^m \rightarrow \mathbb{F} \)

The prover sends the answers and we rewire on two claims: "\( \hat{\nu}_1(s) = 8 \)" and "\( \hat{\nu}_1(t) = 8 \).

Indeed, each claim is itself a sum:

\( \sum_{a,b,c \in \mathbb{H}} \hat{w}_2(a,b,c) \cdot g(\hat{\nu}_2(b), \hat{\nu}_2(c)) \cdot I_{H^m}(s,a) = 8 \)

\( \sum_{a,b,c \in \mathbb{H}} \hat{w}_2(a,b,c) \cdot g(\hat{\nu}_2(b), \hat{\nu}_2(c)) \cdot I_{H^m}(t,a) = 8 \)

**Problem:** The number of claims doubles at each layer.
Avoiding Claim Blowup

1 claim about layer 0

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_1(a,b,c) \cdot g(V_1(b), V_1(c)) \cdot I_H^m(0,a) = y \]

2 claims about layer 1

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_2(a,b,c) \cdot g(V_2(b), V_2(c)) \cdot I_H^m(s,a) = \delta \]

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_2(a,b,c) \cdot g(V_2(b), V_2(c)) \cdot I_H^m(t,a) = \delta \]

1 claim about layer 1

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_2(a,b,c) \cdot g(V_2(b), V_2(c)) \cdot [\alpha \cdot I_H^m(s,a) + \beta \cdot I_H^m(t,a)] = \alpha \cdot \gamma + \beta \cdot \delta \]

2 claims about layer 2

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_3(a,b,c) \cdot g(V_3(b), V_3(c)) \cdot I_H^m(s',a) = \delta \]

\[ \sum_{a,b,c \in \mathbb{H}} \hat{w}_3(a,b,c) \cdot g(V_3(b), V_3(c)) \cdot I_H^m(t',a) = \delta \]

\[ \alpha, \beta \in \mathbb{R} \]

random linear combination of the 2 claims

... and so on.
Protocol Summary

- **public coin**
- **number of rounds is**
  \[ D \cdot (\text{sumcheck on 3m vars} + 1) \]
  \[ = O(Dm) = O(D \cdot \log S) \]
- **communication complexity (in bits) is**
  \[ D \cdot (\text{sumcheck on 3m vars of deg } O(1H1) + 2) \]
  \[ = O(D \cdot m \cdot 1H1) = O(D \text{ poly/log } S) \]
- **soundness error is**
  \[ D \cdot (\text{sumcheck on 3m vars of deg } O(1H1) + \frac{1}{1H1}) \]
  \[ = O(D \cdot m \cdot 1H1) \rightarrow 1H1 \geq D \text{ poly/log } S \text{ suffices } \]
- **prover runtime (in field operations) is**
  \[ D \cdot (\text{sumcheck on 3m vars of deg } O(1H1)) \]
  \[ = D \cdot \text{poly (1H1)} = \text{poly (S)} \]
- **verifier runtime (in field operations) is**
  \[ O(D \cdot \text{poly (m, 1H1)}) = D \text{poly/log S} \text{ [eval each of } \hat{\psi}_1, ..., \hat{\psi}_D \text{ at one location} \]

\[ 1 \text{ query to } \hat{\psi}_1 \]
\[ 1 \text{ query to } \hat{\psi}_2 \]
\[ 2 \text{ claims about } \hat{\psi}_1 \]
\[ 2 \text{ claims about } \hat{\psi}_2 \]
\[ 1 \text{ claim about } \hat{\psi}_1 \text{ combination} \]
\[ 1 \text{ claim about } \hat{\psi}_2 \text{ combination} \]

1 claim about \( \hat{\psi}_0 \)

Did not discuss:

in last sumcheck one variable group is in \( H^m \) \( \\text{[not } H^m \] \)