A Crash Course on Fast Interactive Proofs

Justin Thaler

Georgetown University

Interactive Proofs

- Prover P and Verifier V.

- P solves problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.

- Requirements:
  - 1. Completeness: an honest P can convince V to accept.
  - 2. Soundness: V will catch a lying P with high probability.

Interactive Proofs

- Prover P and Verifier V.

- P solves problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.

- Requirements:
  - 1. Completeness: an honest P can convince V to accept.
  - 2. Soundness: V will catch a lying P with high probability.
    - This must hold even if P is computationally unbounded and trying to trick V into accepting the incorrect answer.

Interactive Proof Techniques: Preliminaries

Schwartz-Zippel Lemma

- Recall **FACT:** Let $p \neq q$ be univariate polynomials over field $\boldsymbol{F}$, of degree at most d. Then $\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{d}{|\boldsymbol{F}|}$.

Schwartz-Zippel Lemma

- Recall **FACT:** Let $p \neq q$ be univariate polynomials over field $\boldsymbol{F}$, of degree at most d. Then $\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{d}{|\boldsymbol{F}|}$.
- The **Schwartz-Zippel lemma** is a multivariate generalization:
  - Let $p \neq q$ be ℓ-variate polynomials over field $\boldsymbol{F}$ of total degree at most d. Then $\Pr_{r \in \boldsymbol{F}^{\ell}}[p(r) = q(r)] \leq \frac{d}{|\boldsymbol{F}|}$.

Schwartz-Zippel Lemma

- Recall **FACT:** Let $p \neq q$ be univariate polynomials over field $\boldsymbol{F}$, of degree at most d. Then $\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{d}{|\boldsymbol{F}|}$.
- The **Schwartz-Zippel lemma** is a multivariate generalization:
  - Let $p \neq q$ be ℓ-variate polynomials over field $\boldsymbol{F}$ of total degree at most d. Then $\Pr_{r \in \boldsymbol{F}^{\ell}}[p(r) = q(r)] \leq \frac{d}{|\boldsymbol{F}|}$.
  - "Total degree" refers to the maximum sum of degrees of all variables in any term. E.g., $x_1^2 x_2 + x_1 x_2$ has total degree 3.

Low-Degree and Multilinear Extensions

- Definition [**Extensions**]. Given a function $f\colon \{0,1\}^{\ell} \to \boldsymbol{F}$, a ℓ-variate polynomial g over **F** is said to **extend** f if $f(x) = g(x)$ for all $x \in \{0,1\}^{\ell}$.

- Definition [**Multilinear Extensions].** Any function $f\colon \{0,1\}^{\ell} \to \boldsymbol{F}$ has a **unique** multilinear extension (MLE), denoted $\tilde{f}$.

Low-Degree and Multilinear Extensions

- Definition [**Extensions**]. Given a function $f: \{0,1\}^\ell \to \boldsymbol{F}$, a ℓ-variate polynomial g over **F** is said to **extend** f if $f(x) = g(x)$ for all $x \in \{0,1\}^\ell$.

- Definition [**Multilinear Extensions].** Any function $f: \{0,1\}^\ell \to \boldsymbol{F}$ has a **unique** multilinear extension (MLE), denoted $\tilde{f}$.
  - Multilinear means the polynomial has degree at most 1 in each variable.
  - $(1 - x_1)(1 - x_2)$ is multilinear, $x_1^2 x_2$ is not.

$$f : \{0,1\}^2 \to \mathbf{F}$$

	0	1
0	1	2
1	8	10

$$\tilde{f} : \mathbf{F}^2 \to \mathbf{F}$$

	0	1	2	3	4	5	
0	1	2	3	4	5	6	
1	8	10	12	14	16	18	
2	15	18	21	24	27	30	$\cdots$
3	22	26	30	34	38	42	
4	29	34	39	44	49	56	
5	36	42	48	54	60	68	
			$\vdots$				

Low-Degree and Multilinear Extensions

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f: \{0,1\}^\ell \rightarrow \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.
  - Assuming field addition and multiplication operations take one time step.

Low-Degree and Multilinear Extensions

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f: \{0,1\}^\ell \rightarrow \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.
- The following slides prove a slightly weaker $O(\ell \cdot 2^\ell)$ time bound.

A Useful Expression for the MLE

- For any $w \in \{0,1\}^{\ell}$, define $\delta_w \colon \{0,1\}^{\ell} \to \boldsymbol{F}$ via:

  $\delta_w(w) = 1$ and $\delta_w(y) = 0$ for all $y \neq w$.

A Useful Expression for the MLE

- For any $w \in \{0,1\}^\ell$, define $\delta_w \colon \{0,1\}^\ell \to \boldsymbol{F}$ via:

  $$\delta_w(w) = 1 \text{ and } \delta_w(y) = 0 \text{ for all } y \neq w.$$

- **Fact 1:** For any $z \in \{0,1\}^\ell$, $f(z) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \delta_w(z)$.

A Useful Expression for the MLE

- For any $w \in \{0,1\}^\ell$, define $\delta_w \colon \{0,1\}^\ell \to \boldsymbol{F}$ via:

  $\delta_w(w) = 1$ and $\delta_w(y) = 0$ for all $y \neq w$.
- **Fact 1:** For any $z \in \{0,1\}^\ell$, $f(z) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \delta_w(z)$.
- Lemma (**Lagrange Interpolation**): Let $f \colon \{0,1\}^\ell \to \boldsymbol{F}$. Then as formal polynomials, $\tilde{f}(x) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(x)$.
- Proof: The RHS is multilinear. The lemma then follows from Fact 1 and uniqueness of the MLE.

A Useful Expression for the MLE

- For any $w \in \{0,1\}^\ell$, define $\delta_w \colon \{0,1\}^\ell \to \boldsymbol{F}$ via:

  $$\delta_w(w) = 1 \text{ and } \delta_w(y) = 0 \text{ for all } y \neq w.$$

- **Fact 1:** For any $z \in \{0,1\}^\ell$, $f(z) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \delta_w(z)$.

- Lemma (**Lagrange Interpolation**): Let $f \colon \{0,1\}^\ell \to \boldsymbol{F}$. Then as formal polynomials, $\tilde{f}(x) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(x)$.
- Proof: The RHS is multilinear. The lemma then follows from Fact 1 and uniqueness of the MLE.
- **Fact 2:** (Explicit Expression for Lagrange Basis Polynomials)

  $$\tilde{\delta}_w(x) = \prod_{i=1}^{\ell} \left(x_i w_i + (1 - x_i)(1 - w_i)\right).$$

Evaluating The MLE At Any Point, Efficiently

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f: \{0,1\}^\ell \to \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

Evaluating The MLE At Any Point, Efficiently

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f\colon\{0,1\}^\ell \to \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

- Proof of slightly weaker $O(\ell \cdot 2^\ell)$ time bound:

Evaluating The MLE At Any Point, Efficiently

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f:\{0,1\}^\ell \rightarrow \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

- Proof of slightly weaker $O(\ell \cdot 2^\ell)$ time bound:
  - Recall $\tilde{f}(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(r)$.

Evaluating The MLE At Any Point, Efficiently

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f: \{0,1\}^\ell \to \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

- Proof of slightly weaker $O(\ell \cdot 2^\ell)$ time bound:
  - Recall $\tilde{f}(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(r)$.
  - For each $w \in \{0,1\}^\ell$, $\tilde{\delta}_w(r)$ can be computed with $O(\ell)$ field operations.
    - $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell}(r_i w_i + (1-r_i)(1-w_i))$.

Evaluating The MLE At Any Point, Efficiently

- Fact [VSBW13]: Given as input all 2^ℓ evaluations of a function $f: \{0,1\}^\ell \rightarrow \boldsymbol{F}$, for any point $r \in \boldsymbol{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

- Proof of slightly weaker $O(\ell \cdot 2^\ell)$ time bound:
  - Recall $\tilde{f}(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(r)$.
  - For each $w \in \{0,1\}^\ell$, $\tilde{\delta}_w(r)$ can be computed with $O(\ell)$ field operations.
    - $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell} (r_i w_i + (1 - r_i)(1 - w_i))$.

- Can reduce to time $O(2^\ell)$ via dynamic programming.

The Sum-Check Protocol [LFKN90]

Sum-Check Protocol [LFKN90]

- Input: V given oracle access to a ℓ-variate polynomial g over field $\boldsymbol{F}$.
- Goal: compute the quantity:

$$\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$\sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- If this check passes, it is safe for V to believe that C_1 is the correct answer, so long as V believes that $s_1 = H_1$.
- How to check this? Just check that s_1 and H_1 agree at a random point r_1.

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- If this check passes, it is safe for V to believe that C_1 is the correct answer, so long as V believes that $s_1 = H_1$.
- How to check this? Just check that s_1 and H_1 agree at a random point r_1.
- V can compute $s_1(r_1)$ directly from P's first message, but not $H_1(r_1)$.

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks r_1 at random from $\boldsymbol{F}$ and sends r_1 to P.
- **Round 2**: They recursively check that $s_1(r_1) = H_1(r_1)$.

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks r_1 at random from $\boldsymbol{F}$ and sends r_1 to P.
- **Round 2**: They recursively check that $s_1(r_1) = H_1(r_1)$.

  i.e., that $s_1(r_1) = \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(r_1, b_2, \dots, b_\ell)$.

- **Start**: P sends claimed answer C_1. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks r_1 at random from $\boldsymbol{F}$ and sends r_1 to P.
- **Round 2**: They recursively check that $s_1(r_1) = H_1(r_1)$.

  i.e., that $s_1(r_1) = \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(r_1, b_2, \dots, b_\ell)$.

- **Round ℓ (Final round):** P sends univariate polynomial $s_\ell(X_\ell)$ claimed to equal

  $H_\ell := g(r_1, \dots, r_{\ell-1}, X_\ell)$.

- V checks that $s_{\ell-1}(r_{\ell-1}) = s_\ell(0) + s_\ell(1)$.
- V picks r_ℓ at random, and needs to check that $s_\ell(r_\ell) = g(r_1, \dots, r_\ell)$.
  - No need for more rounds. V can perform this check with one oracle query.

Analysis of the Sum-Check Protocol

Completeness and Soundness

- Completeness holds by design: If P sends the prescribed messages, then all of V's checks will pass.

Completeness and Soundness

- Completeness holds by design: If P sends the prescribed messages, then all of V's checks will pass.
- Soundness: If P does not send the prescribed messages, then V rejects with probability at least $1 - \frac{\ell \cdot d}{|F|}$, where d is the maximum degree of g in any variable.
- Proof is by induction on the number of variables ℓ.

Costs of the Sum-Check Protocol

- Total communication is $O(d\ell)$ field elements.
  - P sends ℓ messages, each a univariate polynomial of degree at most d. V sends $\ell - 1$ messages, each consisting of one field element.

Costs of the Sum-Check Protocol

- Total communication is $O(d\ell)$ field elements.
  - P sends ℓ messages, each a univariate polynomial of degree at most d. V sends $\ell - 1$ messages, each consisting of one field element.
- V's runtime is:

$$O(d\ell + [\textit{time required to evaluate g at one point}]).$$

Costs of the Sum-Check Protocol

- Total communication is $O(d\ell)$ field elements.
  - P sends ℓ messages, each a univariate polynomial of degree at most d. V sends $\ell - 1$ messages, each consisting of one field element.
- V's runtime is:

$$O(d\ell + [\textit{time required to evaluate g at one point}]).$$

- P's runtime is at most:

$$O\left(d \cdot 2^{\ell} \cdot [\textit{time required to evaluate g at one point}]\right).$$

An Application of the Sum-Check Protocol

A Doubly-Efficient Interactive Proof for Matrix Multiplication

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given $n \times n$ input matrices A, B over field $\boldsymbol{F}$, compute $C = A \cdot B$.

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given $n \times n$ input matrices A, B over field $\boldsymbol{F}$, compute $C = A \cdot B$.
- P simply determines the "right answer", and then P does $O(n^2)$ extra work to prove its correctness.
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.
- V runs in linear time (which is also optimal).

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given $n \times n$ input matrices A, B over field $\boldsymbol{F}$, compute $C = A \cdot B$.
- P simply determines the “right answer”, and then P does $O(n^2)$ extra work to prove its correctness.
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.
- V runs in linear time (which is also optimal).

Problem Size	Naïve MatMult Time	Additional P time	V Time	Rounds	Protocol Comm
1024 x 1024	2.17 s	0.03 s	0.09 s	11	264 bytes
2048 x 2048	18.23 s	0.13 s	0.30 s	12	288 bytes

Comparison to Freivalds' Algorithm

- Freivalds (MFCS, 1979) gave the following protocol for MatMult. To check $A \cdot B = D$:
  - V picks random vector x.
  - Accepts if $A \cdot (Bx) = Dx$.
  - **No** extra work for P, $O(n^2)$ time for V.

Comparison to Freivalds' Algorithm

- Freivalds (MFCS, 1979) gave the following protocol for MatMult. To check $A \cdot B = D$:
  - V picks random vector x.
  - Accepts if $A \cdot (Bx) = Dx$.
  - **No** extra work for P, $O(n^2)$ time for V.
- Our big win: verifying algorithms that invoke MatMult, but aren't really interested in matrices.
  - E.g., Best-known subgraph-counting algorithms square the adjacency matrix, but are only interested in a single number.
  - Total communication for us is $O(\log n)$, Freivalds' is $\Omega(n^2)$.

MatMult Protocol: Technical Details

Notation

- Given $n \times n$ input matrices A, B over field $\boldsymbol{F}$, interpret A and B as functions mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n}$ to $\boldsymbol{F}$ via $A\left(i_1, \ldots, i_{\log n}, j_1, \ldots, j_{\log n}\right) = A_{ij}$.

- Let $C = A \cdot B$ denote the true answer.

- Let $\tilde{A}, \tilde{B}$ denote the multilinear extensions of the functions A and B.

MatMult Protocol

- P sends a matrix D claimed to equal $C = A \cdot B$.

MatMult Protocol

- P sends a matrix D claimed to equal $C = A \cdot B$.
- V evaluates $\widetilde{D}$ at a random point $(\boldsymbol{r_1}, \boldsymbol{r_2}) \in \boldsymbol{F}^{\log n} \times \boldsymbol{F}^{\log n}$.

MatMult Protocol

- P sends a matrix D claimed to equal $C = A \cdot B$.
- V evaluates $\widetilde{D}$ at a random point $(\boldsymbol{r_1}, \boldsymbol{r_2}) \in \boldsymbol{F}^{\log n} \times \boldsymbol{F}^{\log n}$.
- By Schwartz-Zippel: it is safe for V to believe that D equals the correct answer C as long as $\widetilde{D}(\boldsymbol{r_1}, \boldsymbol{r_2}) = \tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.

MatMult Protocol

- P sends a matrix D claimed to equal $C = A \cdot B$.
- V evaluates $\widetilde{D}$ at a random point $(\boldsymbol{r_1}, \boldsymbol{r_2}) \in \boldsymbol{F}^{\log n} \times \boldsymbol{F}^{\log n}$.
- By Schwartz-Zippel: it is safe for V to believe that D equals the correct answer C as long as $\widetilde{D}(\boldsymbol{r_1}, \boldsymbol{r_2}) = \tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.
- Goal becomes: compute $\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$

MatMult Protocol

- Goal: Compute $\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.

MatMult Protocol

- Goal: Compute $\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.
- For Boolean vectors $(\boldsymbol{i}, \boldsymbol{j}) \in \{0,1\}^{\log n}$, clearly:

$$C(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} A(\boldsymbol{i}, \boldsymbol{k}) B(\boldsymbol{k}, \boldsymbol{j})$$

MatMult Protocol

- Goal: Compute $\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.
- For Boolean vectors $(\boldsymbol{i}, \boldsymbol{j}) \in \{0,1\}^{\log n}$, clearly:

$$C(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} A(\boldsymbol{i}, \boldsymbol{k}) B(\boldsymbol{k}, \boldsymbol{j})$$

- This implies the following **polynomial** identity:

$$\tilde{C}(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} \tilde{A}(\boldsymbol{i}, \boldsymbol{k}) \tilde{B}(\boldsymbol{k}, \boldsymbol{j}).$$

MatMult Protocol

- Goal: Compute $\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2})$.
- For Boolean vectors $(\boldsymbol{i}, \boldsymbol{j}) \in \{0,1\}^{\log n}$, clearly:

$$C(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} A(\boldsymbol{i}, \boldsymbol{k}) B(\boldsymbol{k}, \boldsymbol{j})$$

- This implies the following **polynomial** identity:

$$\tilde{C}(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} \tilde{A}(\boldsymbol{i}, \boldsymbol{k}) \tilde{B}(\boldsymbol{k}, \boldsymbol{j}).$$

- So V applies sum-check protocol to compute

$$\tilde{C}(\boldsymbol{r_1}, \boldsymbol{r_2}) = \sum_{b_1 \in \{0,1\}} \dots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \dots, b_{\log n}),$$

where:

$$g(\boldsymbol{z}) := \tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})\, \tilde{B}(\boldsymbol{z}, \boldsymbol{r_2}).$$

Making V Fast

- At end of sum-check, V must evaluate

$$g(\boldsymbol{r_3}) = \tilde{A}(\boldsymbol{r_1}, \boldsymbol{r_3})\,\tilde{B}(\boldsymbol{r_3}, \boldsymbol{r_2}).$$

- Suffices to evaluate $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{r_3})$ and $\tilde{B}(\boldsymbol{r_3}, \boldsymbol{r_2})$.

Making V Fast

- At end of sum-check, V must evaluate
$$g(\boldsymbol{r_3}) = \tilde{A}(\boldsymbol{r_1}, \boldsymbol{r_3})\, \tilde{B}(\boldsymbol{r_3}, \boldsymbol{r_2}).$$
- Suffices to evaluate $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{r_3})$ and $\tilde{B}(\boldsymbol{r_3}, \boldsymbol{r_2})$.
- Can be done in $O(n^2)$ time by "Fast Evaluation of MLE" lemma in preliminaries.

Making P Fast: A First Attempt

- Recall: we're using sum-check to compute $\sum_{b_1 \in \{0,1\}} \dots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \dots, b_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal: $\sum_{b_{i+1} \in \{0,1\}} \dots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \dots, r_{3,i-1}, X_i, b_{i+1}, \dots, b_{\log n})$
  - Suffices for P to specify $s_i(0), s_i(1)$, $s_i(2)$

Making P Fast: A First Attempt

- Recall: we're using sum-check to compute $\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \ldots, b_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal: $\sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \ldots, r_{3,i-1}, X_i, b_{i+1}, \ldots, b_{\log n})$
  - Suffices for P to specify $s_i(0), s_i(1), s_i(2)$
  - Thus: **Enough for P to evaluate g at all points of the form** $(r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$

Making P Fast: A First Attempt

- Recall: we're using sum-check to compute $\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \ldots, b_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal: $\sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \ldots, r_{3,i-1}, X_i, b_{i+1}, \ldots, b_{\log n})$
  - Suffices for P to specify $s_i(0), s_i(1), s_i(2)$
  - Thus: **Enough for P to evaluate g at all points of the form** $(r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$
  - This is $O(\frac{n}{2^i})$ points.

Making P Fast: A First Attempt

- Recall: we're using sum-check to compute $\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \dots, b_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal: $\sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \dots, r_{3,i-1}, X_i, b_{i+1}, \dots, b_{\log n})$
  - Suffices for P to specify $s_i(0), s_i(1), s_i(2)$
  - Thus: **Enough for P to evaluate g at all points of the form** $(r_{3,1}, \dots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \dots, b_{\log n})$: $b_{i+1}, \dots, b_{\log n} \in \{0,1\}^{\log n - i}$
  - This is $O(\frac{n}{2^i})$ points.
  - Recall $\tilde{A}$ and $\tilde{B}$ can each be evaluated at any input in $O(n^2)$ time, and hence so can g.
  - So P can compute s_i in $O\left(\frac{n}{2^i} \cdot n^2\right) = O(n^3/2^i)$ time.

Making P Fast: A First Attempt

- Recall: we're using sum-check to compute $\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \ldots, b_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal: $\sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \ldots, r_{3,i-1}, X_i, b_{i+1}, \ldots, b_{\log n})$
  - Suffices for P to specify $s_i(0), s_i(1), s_i(2)$
  - Thus: **Enough for P to evaluate g at all points of the form** $(r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$
  - This is $O(\frac{n}{2^i})$ points.
  - Recall $\tilde{A}$ and $\tilde{B}$ can each be evaluated at any input in $O(n^2)$ time, and hence so can g.
- Over **all** rounds, this is $O\left(\sum_i n^3 / 2^i\right) = O(n^3)$ total time.

Making P Fast: Second Attempt

- **Recall: Enough to evaluate g at all points of the form:** $\boldsymbol{z} = (r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$
- Already showed: how to do this in $O\left(n^3/2^i\right)$ time.
- Can we improve this to $O(n^2)$ time?

Making P Fast: Second Attempt

- **Recall: Enough to evaluate g at all points of the form:**
  $\boldsymbol{z} = (r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$
- Already showed: how to do this in $O\left(n^3/2^i\right)$ time.
- Can we improve this to $O(n^2)$ time?
- Key observation each entry A_{ij} contributes to $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Similarly entry B_{ij} contributes to $\tilde{B}(\boldsymbol{z}, \boldsymbol{r_2})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.

Making P Fast: Second Attempt

- **Recall: Enough to evaluate g at all points of the form:** $\boldsymbol{z} = (r_{3,1}, \dots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \dots, b_{\log n})$: $b_{i+1}, \dots, b_{\log n} \in \{0,1\}^{\log n - i}$
- Already showed: how to do this in $O\left(n^3/2^i\right)$ time.
- Can we improve this to $O(n^2)$ time?
- Key observation each entry A_{ij} contributes to $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Similarly entry B_{ij} contributes to $\tilde{B}(\boldsymbol{z}, \boldsymbol{r_2})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Recall: $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z})$, where

$$\tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1})\tilde{\delta}_j(\boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1}) \cdot \prod_{t=1}^{\ell}(j_t z_t + (1-j_t)(1-z_t)).$$

Making P Fast: Second Attempt

- **Recall: Enough to evaluate g at all points of the form:**
  $\boldsymbol{z} = (r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_{\log n})$: $b_{i+1}, \ldots, b_{\log n} \in \{0,1\}^{\log n - i}$
- Already showed: how to do this in $O\left(n^3/2^i\right)$ time.
- Can we improve this to $O(n^2)$ time?
- Key observation each entry A_{ij} contributes to $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Similarly entry B_{ij} contributes to $\tilde{B}(\boldsymbol{z}, \boldsymbol{r_2})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Recall: $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z})$, where

$$\tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1})\tilde{\delta}_j(\boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1}) \cdot \prod_{t=1}^{\ell} (j_t z_t + (1-j_t)(1-z_t)).$$

  - For $t \geq i+1$, the t'th entry of $\boldsymbol{z}$ is $b_t \in \{0,1\}$.
  - If $b_t \neq j_t$, then $\tilde{\delta}_j(\boldsymbol{z}) = 0$, so $\tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z}) = 0$.

Making P Fast: Second Attempt

- **Recall: Enough to evaluate g at all points of the form:** $\boldsymbol{z} = (r_{3,1}, \dots, r_{3,i-1}, \{0,1,2\}, b_{i+1}, \dots, b_{\log n})$: $b_{i+1}, \dots, b_{\log n} \in \{0,1\}^{\log n - i}$
- Already showed: how to do this in $O\left(n^3/2^i\right)$ time.
- Can we improve this to $O(n^2)$ time?
- Key observation each entry A_{ij} contributes to $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Similarly entry B_{ij} contributes to $\tilde{B}(\boldsymbol{z}, \boldsymbol{r_2})$ for less than 3 tuples $\boldsymbol{z}$ of the above form.
  - Recall: $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z})$, where

$$\tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1})\tilde{\delta}_j(\boldsymbol{z}) = \tilde{\delta}_i(\boldsymbol{r_1}) \cdot \prod_{t=1}^{\ell} (j_t z_t + (1-j_t)(1-z_t)).$$

  - For $t \geq i+1$, the t'th entry of $\boldsymbol{z}$ is $b_t \in \{0,1\}$.
  - If $b_t \neq j_t$, then $\tilde{\delta}_j(\boldsymbol{z}) = 0$, so $\tilde{\delta}_{(i,j)}(\boldsymbol{r_1}, \boldsymbol{z}) = 0$.
  - i.e., $A(i,j)$ only contributes to $\tilde{A}(\boldsymbol{r_1}, \boldsymbol{z})$ if $\left(j_{i+1}, \dots, j_{\log n}\right) = (b_{i+1}, \dots, b_{\log n})$

Making P Fast: Second Attempt

- Summary: In round i, P must evaluate g at $O(\frac{n}{2^i})$ points of a special form (trailing entries are **Boolean**).
- Each matrix entry A_{ij}, B_{ij} contributes to only **at most three** of these evaluations.
- So P can run in $O(n^2)$ time per round, or $O(n^2 \log n)$ time across all $O(\log n)$ rounds.

Making P Fast: Third Attempt

- With care: can bring P's time down to $O(n^2)$ across **all** rounds.
- Key idea: **Reuse work** across rounds.
  - If two matrix index pairs $(\boldsymbol{i},\boldsymbol{j})$ and $(\boldsymbol{i}',\boldsymbol{j}')$ in $\{0,1\}^{\log n}\times\{0,1\}^{\log n}$ agree in their last k bits, then $A_{\boldsymbol{ij}}$ and $A_{\boldsymbol{i}'\boldsymbol{j}'}$ contribute to the **same** points $\boldsymbol{z}$ in rounds k and up.
  - Can treat $(\boldsymbol{i},\boldsymbol{j})$ and $(\boldsymbol{i}',\boldsymbol{j}')$ as a single entity thereafter.
  - Only $O(n/2^k)$, entities of interest in round k.
  - Total work across all rounds is proportional to

$$\sum_{1\le k\le \log n} n/2^k = 2n.$$

Details of Third Attempt

- $\tilde{A}(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{x_1}, \boldsymbol{x_2})$

where $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell}(r_i w_i + (1 - r_i)(1 - w_i))$.

Details of Third Attempt

- $\tilde{A}(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{x_1}, \boldsymbol{x_2})$

where $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell}(r_i w_i + (1 - r_i)(1 - w_i))$.

- Consider two indices $(\boldsymbol{i}, \boldsymbol{j})$ and $(\boldsymbol{i}', \boldsymbol{j}')$ that differ in only their first bit, e.g., $(\boldsymbol{i}, \boldsymbol{j}) = 00\ldots0$ and $(\boldsymbol{i}', \boldsymbol{j}')=10\ldots0$.

Details of Third Attempt

- $\tilde{A}(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{x_1}, \boldsymbol{x_2})$

where $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell}(r_i w_i + (1-r_i)(1-w_i))$.

- Consider two indices $(\boldsymbol{i}, \boldsymbol{j})$ and $(\boldsymbol{i'}, \boldsymbol{j'})$ that differ in only their first bit, e.g., $(\boldsymbol{i}, \boldsymbol{j}) =$ 00…0 and $(\boldsymbol{i'}, \boldsymbol{j'})$=10…0.
- Then the products defining $\tilde{\delta}_{(\boldsymbol{i},\boldsymbol{j})}$ and $\tilde{\delta}_{(\boldsymbol{i'},\boldsymbol{j'})}$ are the same **except for the first term**.
  - For $\tilde{\delta}_{(\boldsymbol{i},\boldsymbol{j})}(\boldsymbol{x_1}, \boldsymbol{x_2})$ the first term is $x_{1,1} \cdot 0 + (1-x_{1,1}) \cdot 1 = (1-x_{1,1})$.
  - For $\tilde{\delta}_{(\boldsymbol{i'},\boldsymbol{j'})}(\boldsymbol{x_1}, \boldsymbol{x_2})$ the first term is $x_{1,1} \cdot 1 + (1-x_{1,1}) \cdot 0 = x_{1,1}$.

Details of Third Attempt

- $\tilde{A}(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sum_{(i,j)\in\{0,1\}^{2\log n}} A(i,j) \cdot \tilde{\delta}_{(i,j)}(\boldsymbol{x_1}, \boldsymbol{x_2})$

where $\tilde{\delta}_w(r) = \prod_{i=1}^{\ell}(r_i w_i + (1-r_i)(1-w_i))$.

- Consider two indices $(\boldsymbol{i}, \boldsymbol{j})$ and $(\boldsymbol{i'}, \boldsymbol{j'})$ that differ in only their first bit, e.g., $(\boldsymbol{i}, \boldsymbol{j}) =$ 00…0 and $(\boldsymbol{i'}, \boldsymbol{j'})$=10…0.
- Then the products defining $\tilde{\delta}_{(\boldsymbol{i},\boldsymbol{j})}$ and $\tilde{\delta}_{(\boldsymbol{i'},\boldsymbol{j'})}$ are the same **except for the first term**.
  - For $\tilde{\delta}_{(\boldsymbol{i},\boldsymbol{j})}(\boldsymbol{x_1}, \boldsymbol{x_2})$ the first term is $x_{1,1} \cdot 0 + (1 - x_{1,1}) \cdot 1 = (1 - x_{1,1})$.
  - For $\tilde{\delta}_{(\boldsymbol{i'},\boldsymbol{j'})}(\boldsymbol{x_1}, \boldsymbol{x_2})$ the first term is $x_{1,1} \cdot 1 + (1 - x_{1,1}) \cdot 0 = x_{1,1}$.
- Once the **first variable** $x_{1,1}$ is bound to a fixed value $r_{1,1}$ by the MatMult protocol, this difference is fixed.
  - i.e., no need for P to remember both $A(\boldsymbol{i}, \boldsymbol{j})$ and $A(\boldsymbol{i'}, \boldsymbol{j'})$.
  - Suffices just to remember $(1 - r_{1,1})A(\boldsymbol{i}, \boldsymbol{j}) + r_{1,1}A(\boldsymbol{i'}, \boldsymbol{j'})$.

Details of Third Attempt

A(0,0,0)	A(0,0,1)	A(0,1,0)	A(0,1,1)	A(1,0,0)	A(1,0,1)	A(1,1,0)	A(1,1,1)

$(1-r_1)$ A(0,0,0) $+ r_1$A(1,0,0) := B(0,0)	$(1-r_1)$ A(0,0,1) $+ r_1$A(1,0,1) := B(0,1)	$(1-r_1)$ A(0,1,0) $+ r_1$A(1,1,0) := B(1,0)	$(1-r_1)$ A(0,1,1) $+ r_1$A(1,1,1) := B(1,1)

$(1-r_2)$ B(0,0) $+ r_2$B(1,0) :=C(0)	$(1-r_2)$ B(0,1) $+ r_2$B(1,1) := C(1)

$(1-r_3)$ C(0) $+ r_3$C(1)

A Second Application of the Sum-Check Protocol

A Doubly-Efficient Interactive Proof for Counting Triangles

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6}\cdot\sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ki}$

$$= \frac{1}{6}\cdot\sum_{(i,j)\in[n]^2}(A^2)_{ij}\cdot A_{ij}.$$

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6}\cdot\sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ki}$

$$= \frac{1}{6}\cdot\sum_{(i,j)\in[n]^2} (A^2)_{ij}\cdot A_{ij}.$$

- View A and A^2 as functions mapping $\{0,1\}^{\log n}\times\{0,1\}^{\log n}$ to $\boldsymbol{F}$.
- Define the polynomial $h(X,Y) = \widetilde{(A^2)}(X,Y)\,\tilde{A}(X,\mathrm{Y})$.

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6} \cdot \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ki}$

$$= \frac{1}{6} \cdot \sum_{(i,j)\in[n]^2} (A^2)_{ij} \cdot A_{ij}.$$

- View A and A^2 as functions mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n}$ to $\boldsymbol{F}$.
- Define the polynomial $h(X,Y) = \widetilde{(A^2)}(X,Y)\, \tilde{A}(X, \mathrm{Y})$.
- The Protocol:
  - Apply the sum-check protocol to h.

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6}\cdot\sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ki}$

$$= \frac{1}{6}\cdot\sum_{(i,j)\in[n]^2}(A^2)_{ij}\cdot A_{ij}.$$

- View A and A^2 as functions mapping $\{0,1\}^{\log n}\times\{0,1\}^{\log n}$ to $\boldsymbol{F}$.
- Define the polynomial $h(X,Y) = \widetilde{(A^2)}(X,Y)\,\tilde{A}(X,\mathrm{Y})$.
- The Protocol:
  - Apply the sum-check protocol to h.
  - At the end of the protocol, V needs to evaluate:

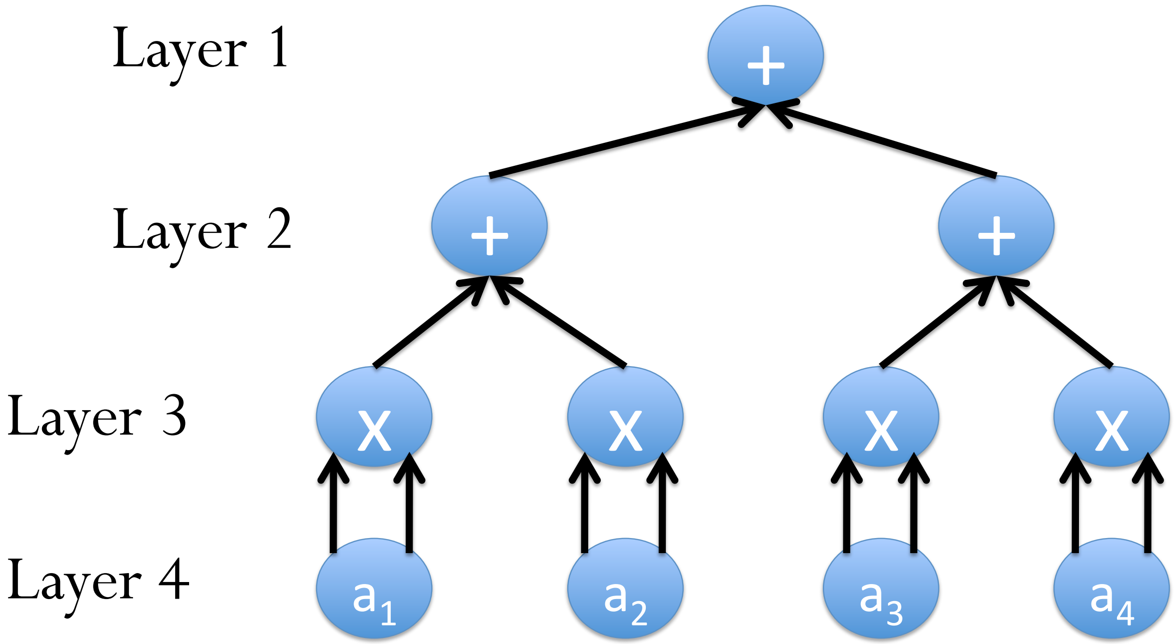
$$h(r_1,r_2)=\widetilde{(A^2)}(r_1,r_2)\,\tilde{A}(r_1,r_2).$$

  - V can evaluate $\tilde{A}(r_1,r_2)$ on its own in $O(n^2)$ time. V uses the MatMult protocol to force P to compute $\widetilde{(A^2)}(r_1,r_2)$ for her.

The GKR Protocol

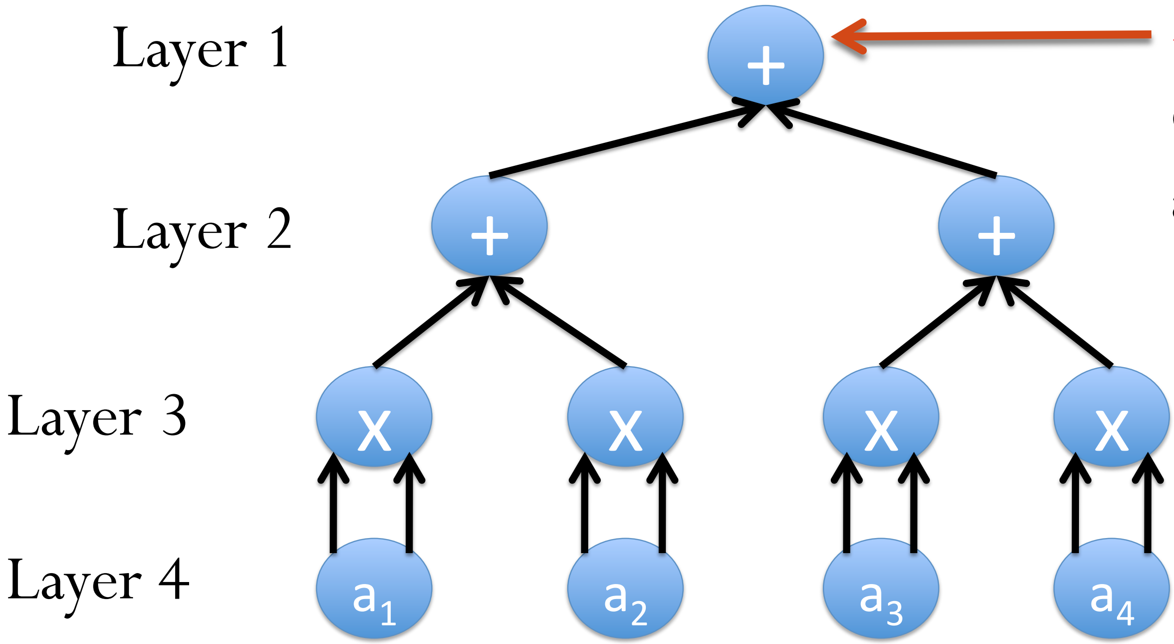
A General-Purpose Doubly-Efficient Interactive Proof

The GKR Protocol: Overview



F_2 circuit

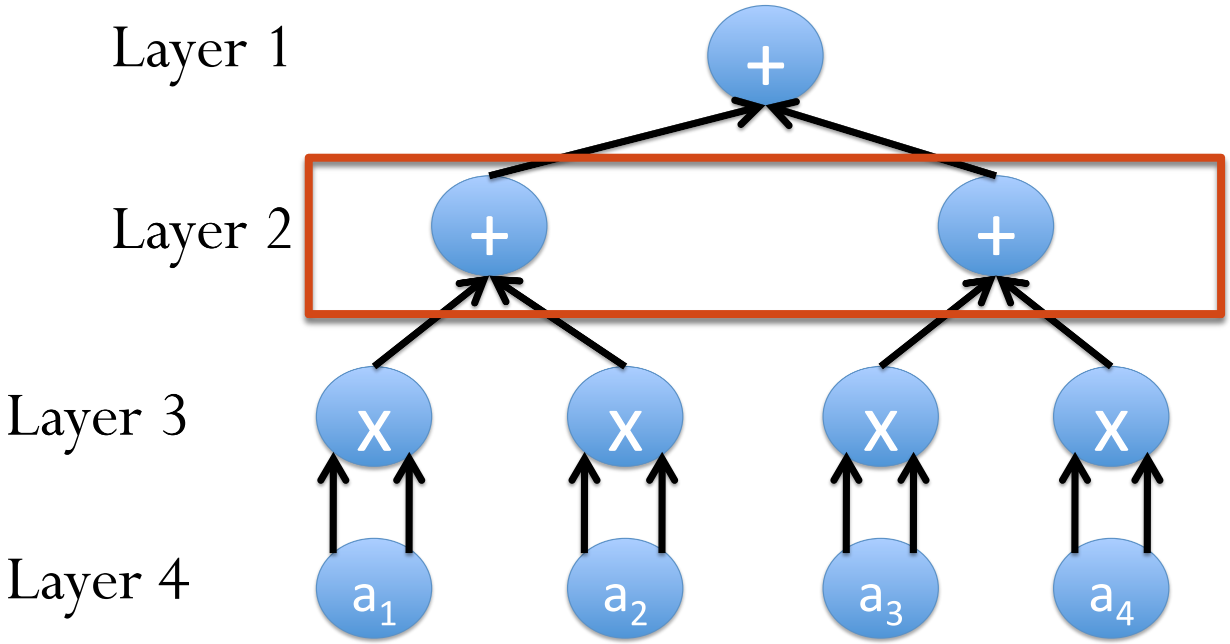
The GKR Protocol: Overview



P starts the conversation with an answer (output).

F_2 circuit

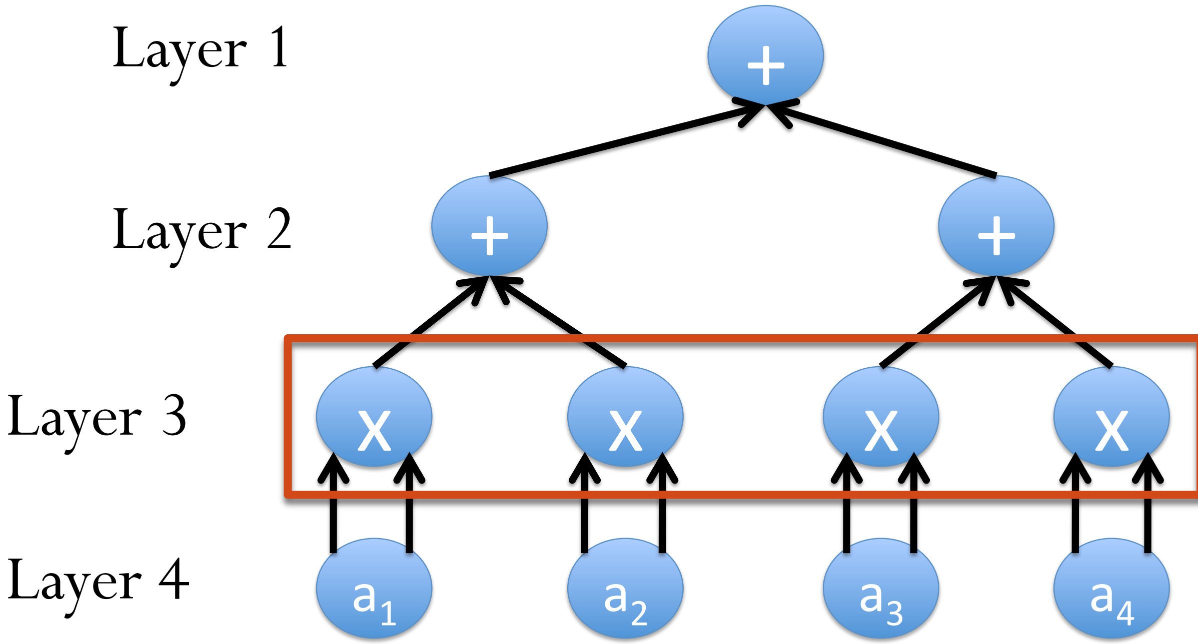
The GKR Protocol: Overview



F_2 circuit

V sends series of challenges. P responds with info about next circuit level.

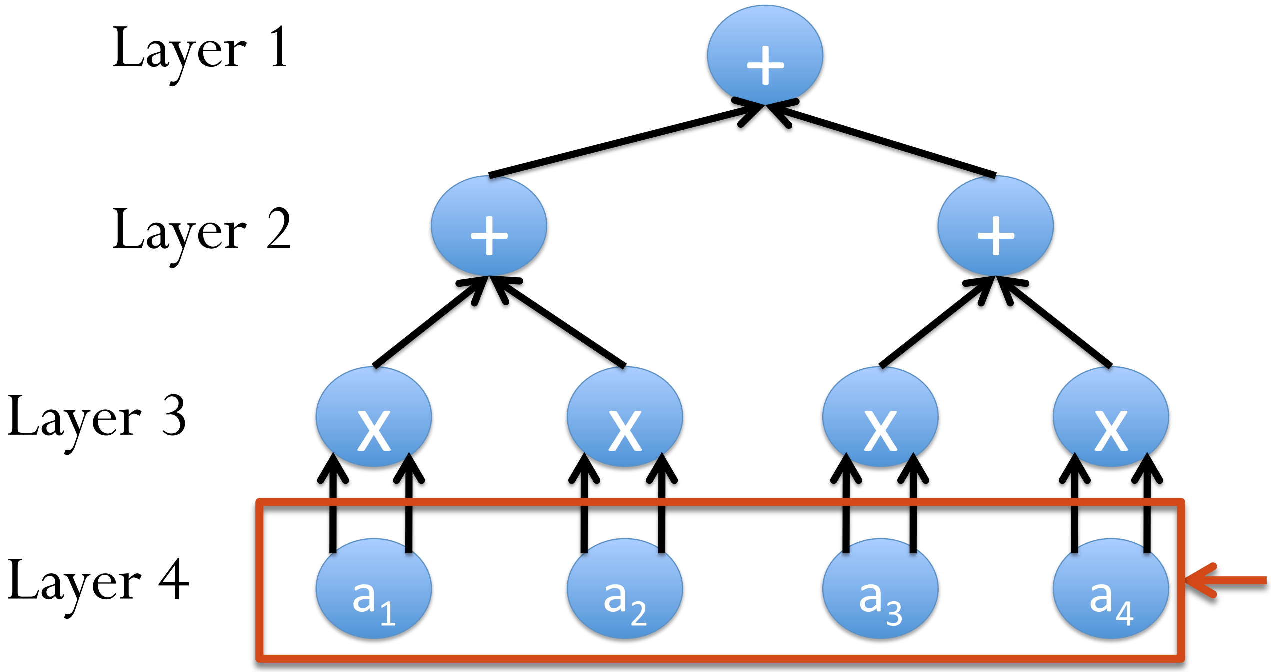
The GKR Protocol: Overview



Challenges continue, layer by layer down to the the input.

F_2 circuit

The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

Notation

- Assume layers i and $i + 1$ of C have S gates each.
  - Assign each gate a binary label ($\log S$ bits).
- Let $W_i(\boldsymbol{a})\colon \{0,1\}^{\log S} \to \boldsymbol{F}$ output the value of gate $\boldsymbol{a}$ at layer i.

Notation

- Assume layers i and $i+1$ of C have S gates each.
  - Assign each gate a binary label ($\log S$ bits).
- Let $W_i(\boldsymbol{a})\colon \{0,1\}^{\log S} \to \boldsymbol{F}$ output the value of gate $\boldsymbol{a}$ at layer i.
- Let $\mathrm{add}_i(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})\colon \{0,1\}^{3\log S} \to \boldsymbol{F}$ output 1 iff $(\boldsymbol{b}, \boldsymbol{c}) = \left(\mathrm{in}_{\mathbf{1}}(\boldsymbol{a}), \mathrm{in}_{\mathbf{2}}(\boldsymbol{a})\right)$ and gate $\boldsymbol{a}$ at layer i is an addition gate.

Notation

- Assume layers i and $i+1$ of C have S gates each.
  - Assign each gate a binary label ($\log S$ bits).
- Let $W_i(\boldsymbol{a})\colon \{0,1\}^{\log S} \to \boldsymbol{F}$ output the value of gate $\boldsymbol{a}$ at layer $\boldsymbol{i}$.
- Let $\mathrm{add}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})\colon \{0,1\}^{3\log S} \to \boldsymbol{F}$ output 1 iff $(\boldsymbol{b},\boldsymbol{c}) = \left(\mathrm{in}_{\mathbf{1}}(\boldsymbol{a}), \mathrm{in}_{\mathbf{2}}(\boldsymbol{a})\right)$ and gate $\boldsymbol{a}$ at layer i is an addition gate.
- Let $\mathrm{mult}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})\colon \{0,1\}^{3\log S} \to \boldsymbol{F}$ output 1 iff $(\boldsymbol{b},\boldsymbol{c}) = \left(\mathrm{in}_{\mathbf{1}}(\boldsymbol{a}), \mathrm{in}_{\mathbf{2}}(\boldsymbol{a})\right)$ and gate $\boldsymbol{a}$ at layer i is a multiplication gate.

GKR Protocol: Goal of Iteration i

- Iteration i starts with a claim from P about $\widetilde{W}_i(\boldsymbol{r}_1)$ for a random point $\boldsymbol{r}_1 \in \boldsymbol{F}^{\log S}$.
- Goal: Reduce this to a claim about $\widetilde{W}_{i+1}(\boldsymbol{r}_2)$ for a random point $\boldsymbol{r}_2 \in \boldsymbol{F}^{\log S}$.

GKR Protocol: Goal of Iteration i

- Iteration $\boldsymbol{i}$ starts with a claim from P about $\widetilde{W}_i(\boldsymbol{r}_1)$ for a random point $\boldsymbol{r}_1 \in \boldsymbol{F}^{\log S}$.
- Goal: Reduce this to a claim about $\widetilde{W}_{i+1}(\boldsymbol{r}_2)$ for a random point $\boldsymbol{r}_2 \in \boldsymbol{F}^{\log S}$.
- Observation: $W_i(\boldsymbol{a}) =$

$$\sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{\log S}} [\mathrm{add}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(W_{i+1}(\boldsymbol{b}) + W_{i+1}(\boldsymbol{c})) + \mathrm{mult}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(W_{i+1}(\boldsymbol{b}) \cdot W_{i+1}(\boldsymbol{c}))]$$

GKR Protocol: Goal of Iteration i

- Iteration i starts with a claim from P about $\widetilde{W}_i(\boldsymbol{r}_1)$ for a random point $\boldsymbol{r}_1 \in \boldsymbol{F}^{\log S}$.
- Goal: Reduce this to a claim about $\widetilde{W}_{i+1}(\boldsymbol{r}_2)$ for a random point $\boldsymbol{r}_2 \in \boldsymbol{F}^{\log S}$.
- Observation: $W_i(\boldsymbol{a}) =$

$$\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log S}} [\mathrm{add}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(W_{i+1}(\boldsymbol{b}) + W_{i+1}(\boldsymbol{c})) + \mathrm{mult}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(W_{i+1}(\boldsymbol{b}) \cdot W_{i+1}(\boldsymbol{c}))]$$

- Hence, the following equality holds as formal polynomials: $\widetilde{W}_i(\boldsymbol{a}) =$

$$\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log S}} [\widetilde{\mathrm{add}}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mult}}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))]$$

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute
- $\widetilde{W}_i(\boldsymbol{r}_1) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{\log s}} g(\boldsymbol{b}, \boldsymbol{c})$, where:

$$\begin{aligned} g(\boldsymbol{b}, \boldsymbol{c}) = &\ \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1, \boldsymbol{b}, \boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) \\ &+ \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1, \boldsymbol{b}, \boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c})) \end{aligned}$$

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute
- $\widetilde{W}_i(\boldsymbol{r}_1) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c})$, where:

$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

- At end of sum-check protocol, V must evaluate $g(\boldsymbol{r}_2,\boldsymbol{r}_3)$.

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute
- $\widetilde{W}_i(\boldsymbol{r}_1) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c})$, where:

$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

- At end of sum-check protocol, V must evaluate $g(\boldsymbol{r}_2,\boldsymbol{r}_3)$.
- Let us assume V can compute $\widetilde{\mathrm{add}}_i(\boldsymbol{r}_1,\boldsymbol{r}_2,\boldsymbol{r}_3)$ and $\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1,\boldsymbol{r}_2,\boldsymbol{r}_3)$ unaided in time $\mathrm{polylog}(n)$.
- Then V only needs to know $\widetilde{W}_{i+1}(\boldsymbol{r}_2)$ and $\widetilde{W}_{i+1}(\boldsymbol{r}_3)$ to complete this check.

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute
- $\widetilde{W}_i(\boldsymbol{r}_1) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{\log s}} g(\boldsymbol{b}, \boldsymbol{c})$, where:

$$g(\boldsymbol{b}, \boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1, \boldsymbol{b}, \boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) \\ + \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1, \boldsymbol{b}, \boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

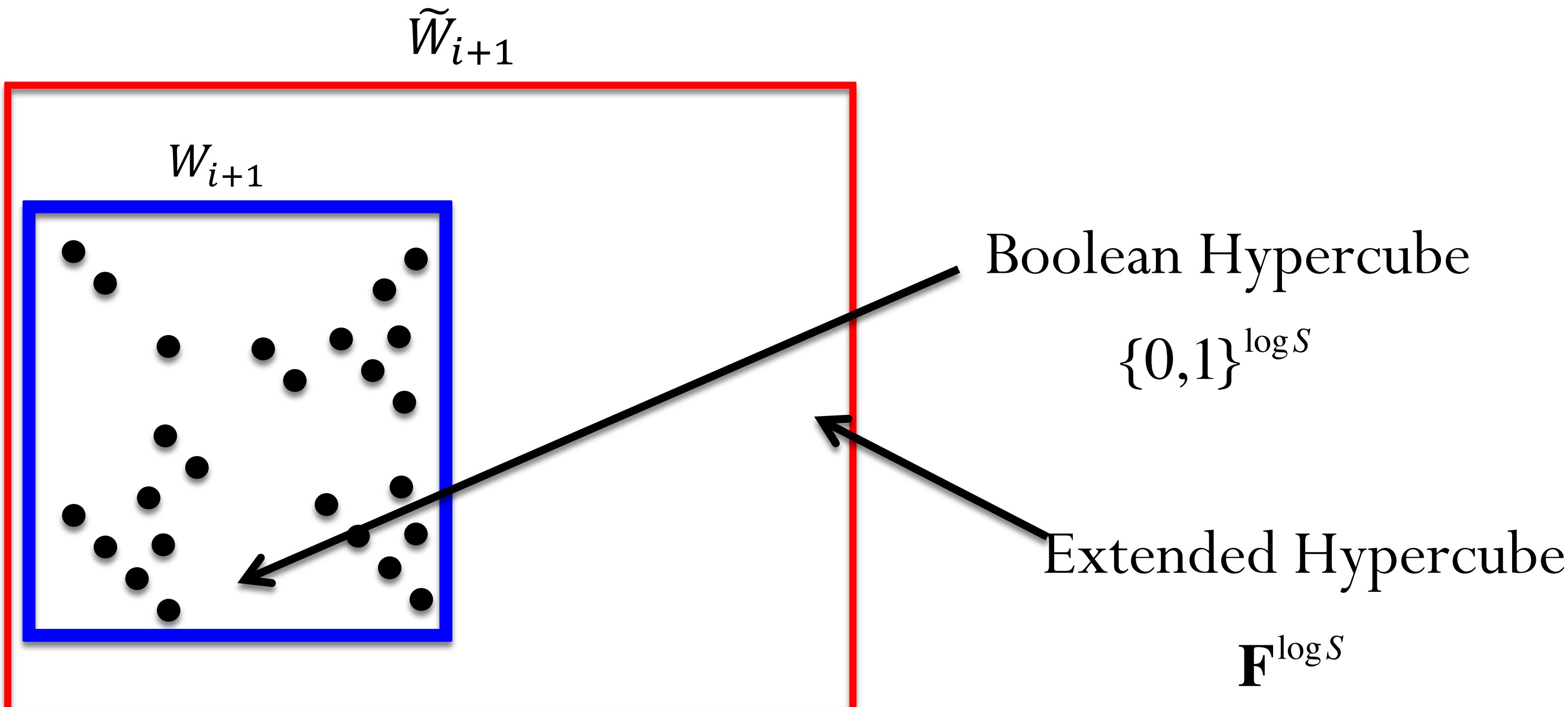
- At end of sum-check protocol, V must evaluate $g(\boldsymbol{r}_2, \boldsymbol{r}_3)$.
- Let us assume V can compute $\widetilde{\mathrm{add}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ and $\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ unaided in time $\mathrm{polylog}(n)$.
- Then V only needs to know $\widetilde{W}_{i+1}(\boldsymbol{r}_2)$ and $\widetilde{W}_{i+1}(\boldsymbol{r}_3)$ to complete this check.
- Iteration $i + 1$ is devoted to computing these values.

Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.

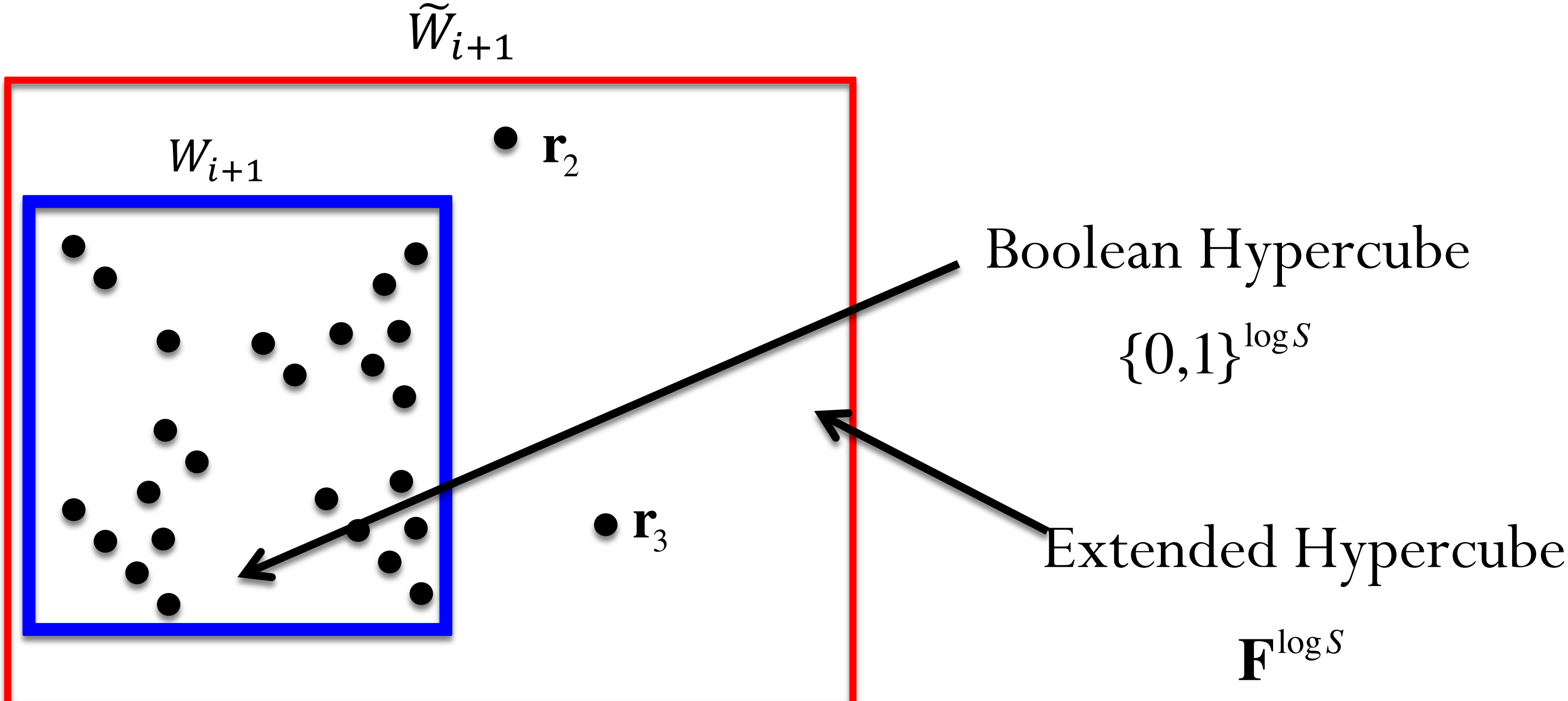
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



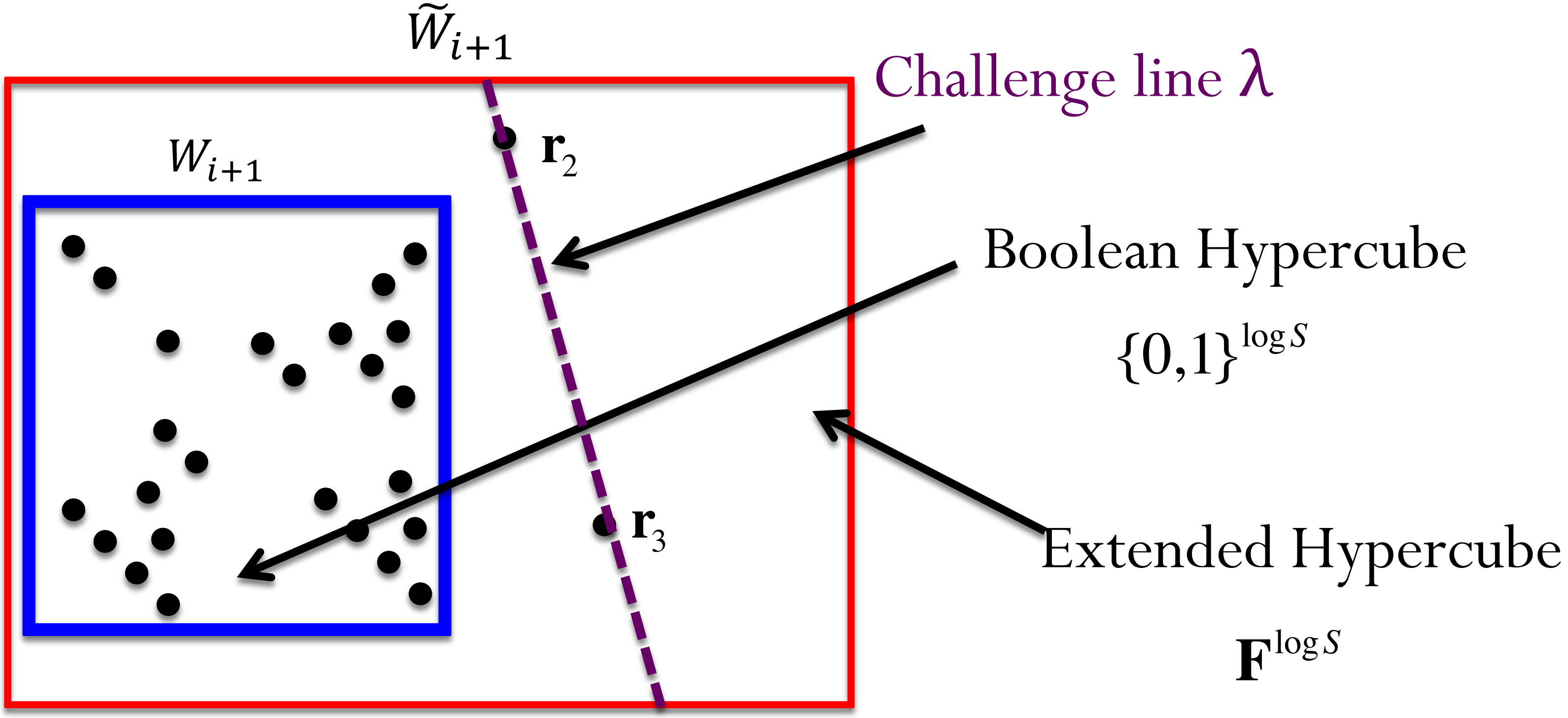
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



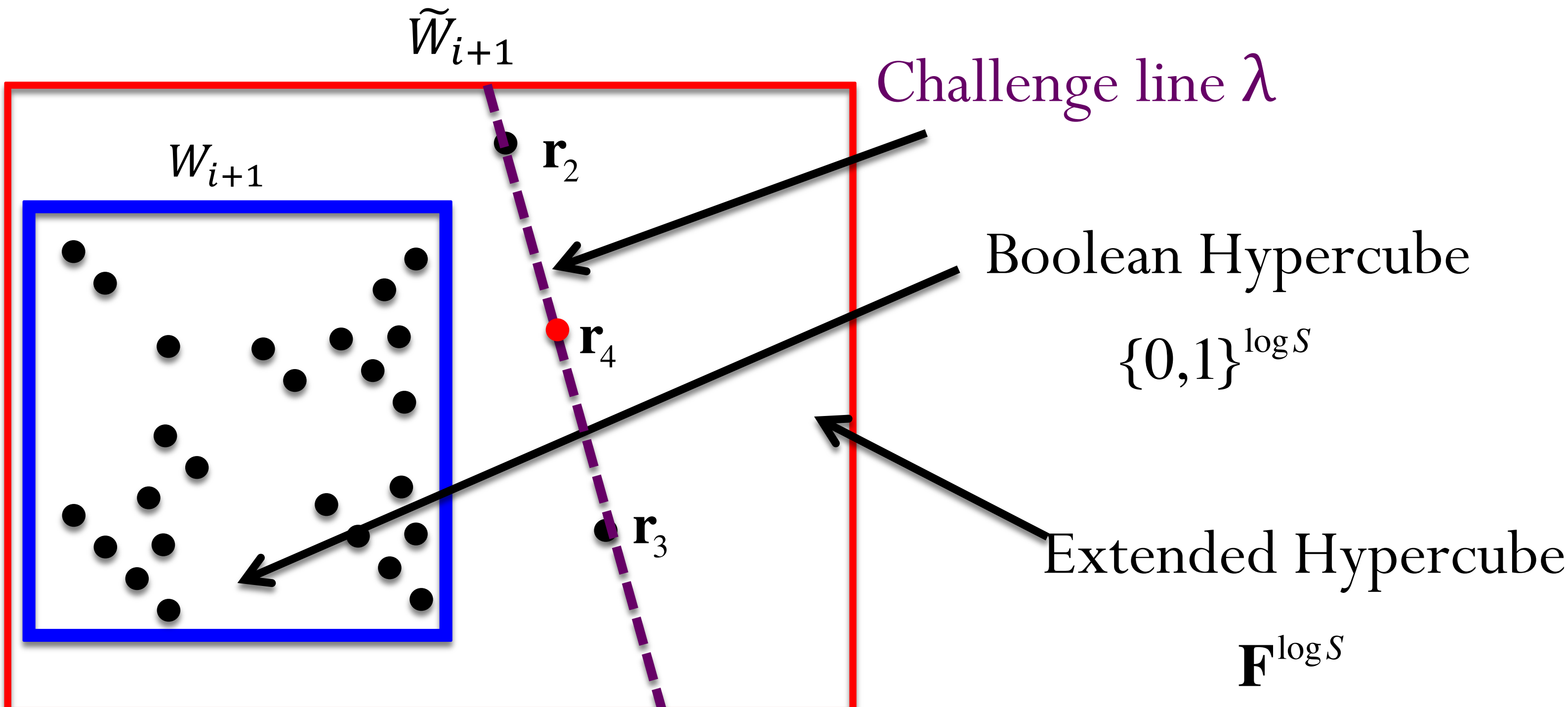
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



Costs of the GKR protocol

- V time is $O(n + D \log S)$ where n is input size, D is circuit depth, and S is circuit size.
  - Assumes V can compute $\widetilde{\mathrm{add}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ and $\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ unaided in time $\mathrm{polylog}(n)$
- Communication cost is $O(D \log S)$.

Costs of the GKR protocol

- V time is $O(n + D \log S)$ where n is input size, D is circuit depth, and S is circuit size.
  - Assumes V can compute $\widetilde{\mathrm{add}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ and $\widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ unaided in time $\mathrm{polylog}(n)$
- Communication cost is $O(D \log S)$.

- P time is $O(S)$.
  - A naïve implementation of P takes $\Omega(S^3)$ time, where S is circuit size.
  - A sequence of works has brought this down to $O(S)$, for arbitrary circuits [CMT12, Thaler13, WJBSTWW17, XZZPS19]

GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute

$$\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c}) \text{ where}$$

$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute $\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c})$ where

$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\text{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\text{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

- A naïve implementation of P takes $\Omega(S^3)$ time, where S is circuit size.
  - Same idea as "Approach 1" from the MatMult protocol.
  - i.e., P evaluates g in each round of sum-check at all $O(S^2/2^i)$ necessary points $\boldsymbol{z}$, taking $O(S)$ time per point.

GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute $\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c})$ where

$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

- A naïve implementation of P takes $\Omega(S^3)$ time, where S is circuit size.
  - Same idea as "Approach 1" from the MatMult protocol.
  - i.e., P evaluates g in each round of sum-check at all $O(S^2/2^i)$ necessary points $\boldsymbol{z}$, taking $O(S)$ time per point.
- [CMT12]: P time is $O(S \log S)$.
  - Achieved via "Approach 2" from the MatMult protocol: each gate of C contributes to $g(\boldsymbol{z})$ for $O(1)$ relevant points $\boldsymbol{z}$ in each round.

GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute $\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{\log s}} g(\boldsymbol{b},\boldsymbol{c})$ where

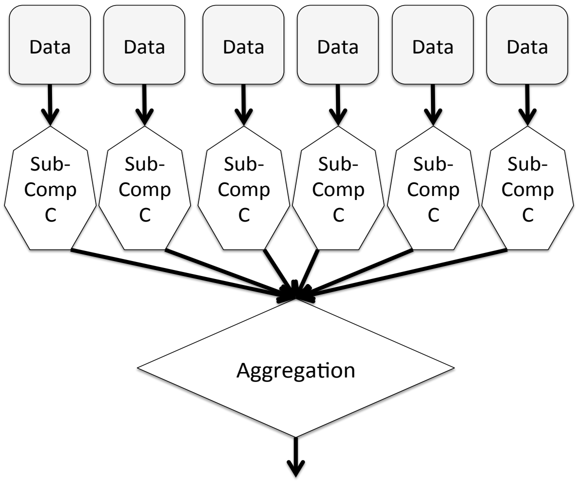
$$g(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\text{add}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\text{mult}}_i(\boldsymbol{r}_1,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

- A naïve implementation of P takes $\Omega(S^3)$ time, where S is circuit size.
  - Same idea as "Approach 1" from the MatMult protocol.
  - i.e., P evaluates g in each round of sum-check at all $\mathrm{O}(S^2/2^i)$ necessary points $\boldsymbol{z}$, taking $\mathrm{O}(S)$ time per point.
- [CMT12]: P time is $O(S \log S)$.
  - Achieved via "Approach 2" from the MatMult protocol: each gate of C contributes to $g(\boldsymbol{z})$ for $O(1)$ relevant points $\boldsymbol{z}$ in each round.
- All subsequent works seek to bring "Approach 3" to bear on the GKR protocol, letting P reuse work across rounds.

GKR Prover Runtime: Details

- [Thaler13]:
  1. P time $O(S)$ for circuits with "nice" wiring patterns.
  2. P time $O(S \log S')$ for data parallel circuits

     i.e., that apply the same subcomputation (of size S') independently to different pieces of data



GKR Prover Runtime: Details

- [Thaler13]:
  1. P time $O(S)$ for circuits with "nice" wiring patterns.
  2. P time $O(S \log S')$ for data parallel circuits

     i.e., that apply the same subcomputation (of size S') independently to different pieces of data
- [WJBSTWW17] improved the data parallel time to $O(S + S' \log S')$.

GKR Prover Runtime: Details

- [Thaler13]:
  1. P time $O(S)$ for circuits with "nice" wiring patterns.
  2. P time $O(S \log S')$ for data parallel circuits

     i.e., that apply the same subcomputation (of size S') independently to different pieces of data
- [WJBSTWW17] improved the data parallel time to $O(S + S' \log S')$.
- [ZGKPP18] extends to data parallel computations where each subcomputation may not be the same.

GKR Prover Runtime: Details

- [Thaler13]:
  1. P time $O(S)$ for circuits with "nice" wiring patterns.
  2. P time $O(S \log S')$ for data parallel circuits

     i.e., that apply the same subcomputation (of size S') independently to different pieces of data
- [WJBSTWW17] improved the data parallel time to $O(S + S' \log S')$.
- [ZGKPP18] extends to data parallel computations where each subcomputation may not be the same.
- [XZZPS19] achieved $O(S)$ time for general circuits.

Rumination on Generality Vs. Efficiency

Generality vs. Efficiency

- The GKR protocol for circuit evaluation has now been rendered optimally efficient for P (up to constant factors).
- **Any** computation can be represented as a circuit evaluation (or satisfiability) problem.
  - But this can introduce tremendous overheads.
  - The GKR protocol **forces** the prover to compute the output in a prescribed manner, which may be far from optimal (gate-by-gate evaluation of a circuit).
- To achieve scalability, the gold standard is really something like the counting triangles protocol.
  - i.e., P computed the right answer directly using the fastest known algorithm, and did a low-order amount of extra work to prove correctness

Succinct ZK Arguments for Circuit-SAT from Interactive Proofs

Succinctness for Circuit-SAT

- The GKR protocol solves arithmetic circuit **evaluation**.
- Applications often require solving circuit **satisfiability**, i.e., given a circuit C and public in put x, prove there exists a "witness" w such that $C(x, w) = y$.
- Naïve approach: have P send w to V and then apply the GKR protocol to check that $C(x, w) = y$.
  - Downside: Proof length is $|w|$.
- [ZGKPP17]: Can decrease the proof length by having the prover **cryptographically commit** to w, without sending w in full to V.

Known Cryptographic Commitment Schemes for Multilinear Polynomials

1. [KZG 2010, PST 2013, ZGKPP17]: Simple, based on bilinear maps, requires trusted setup (SRS of size $|w|$), not quantum secure.
2. [Groth09, BG12, BCCGP16, BG18, BBBPWM18, WSTTW18]: Based on homomorphic commitments. Transparent but not quantum secure.
3. [BSBHR18, BSGKS19]: IOP-based commitment scheme for **univariate** polynomials. Transparent and secure in quantum Random Oracle model. [ZXZS19] combines this with Aurora to handle **multilinear** polynomials.

Known Cryptographic Commitment Schemes for Multilinear Polynomials

1. [KZG 2010, PST 2013, ZGKPP17]: Simple, based on bilinear maps, requires trusted setup (SRS of size $|w|$), not quantum secure.
2. [Groth09, BG12, BCCGP16, BG18, BBBPWM18, WSTTW18]: Based on homomorphic commitments. Transparent but not quantum secure.
3. [BSBHR18, BSGKS19]: IOP-based commitment scheme for **univariate** polynomials. Transparent and secure in quantum Random Oracle model. [ZXZS19] combines this with Aurora to handle **multilinear** polynomials.

- P complexity in 1) and 2): $O(|w|)$ **public-key** crypto operations.
- P complexity in 3): $O(|w| \log |w|)$ field operations and $O(|w|)$ **private-key** crypto operations.
- Not discussed: V time, restrictions on the field size, etc.

Succinctness for Circuit-SAT

- Assume for simplicity that $|x| = |w| = n$.
- When applying the GKR protocol to check that $C(x, w) = y$, V views the input $z := (x, w)$ as a function mapping $\{0,1\} \times \{0,1\}^{\log n}$ to $\boldsymbol{F}$.
  - And the only information V needs to know about z is $\tilde{z}(\boldsymbol{r})$ for a random input $\boldsymbol{r} = (r_1, \boldsymbol{r}') \in \{0,1\} \times \{0,1\}^{\log n}$.
  - **Fact:** $\tilde{z}(\boldsymbol{r}) = (1 - r_1)\, \tilde{x}(\boldsymbol{r}') + r_1\, \tilde{w}(\boldsymbol{r}')$.

Succinctness for Circuit-SAT

- Assume for simplicity that $|x| = |w| = n$.
- When applying the GKR protocol to check that $C(x, w) = y$, V views the input $z := (x, w)$ as a function mapping $\{0,1\} \times \{0,1\}^{\log n}$ to $\boldsymbol{F}$.
  - And the only information V needs to know about z is $\tilde{z}(\boldsymbol{r})$ for a random input $\boldsymbol{r} = (r_1, \boldsymbol{r}') \in \{0,1\} \times \{0,1\}^{\log n}$.
  - **Fact:** $\tilde{z}(\boldsymbol{r}) = (1 - r_1)\, \tilde{x}(\boldsymbol{r}') + r_1\, \tilde{w}(\boldsymbol{r}')$.
- The Argument for CIRCUIT-SAT:
  1. P cryptographically commits to the multilinear polynomial $\widetilde{w}$.
  2. V and P apply to GKR protocol to the claim $C(x, w) = y$.
  3. To perform V's final check in the protocol (which requires knowing $\tilde{z}(\boldsymbol{r})$), V makes P reveal $\widetilde{w}(\boldsymbol{r}')$, and derives $\tilde{z}(\boldsymbol{r})$ using **Fact.**

Zero Knowledge for Circuit-SAT

- Two practical techniques:
  1. [WsTTW18, ZGKPP18]: Efficient implementation of Cramer-Damgård transformation (based on homomorphic commitments).
  2. [CFS17, XZZPS19] IOP-based transformation.
- Both transparent. Only 2) is quantum secure.

THANK YOU!

MIPs and Succinct Arguments Derived Thereof

Arithmetic Circuit Satisfiability

- Given: An arithmetic circuit C over **F** of size S with explicit input x and non-deterministic input w, and claimed output(s) y.
- Goal: Determine if there exists a w such that C(x, w)=y.

Arithmetic Circuit Satisfiability

- Given: An arithmetic circuit C over **F** of size S with explicit input x and non-deterministic input w, and claimed output(s) y.
- Goal: Determine if there exists a w such that C(x, w)=y.
- Assign each gate in C a (log S)-bit label.
- Call a function $W:\{0,1\}^{\log S} \rightarrow \mathbf{F}$ a *transcript* for C.
  - Say that W is *correct* on x if it satisfies the following properties:
    - The values W assigns to the explicit input gates equal x.
    - The value W assigns to the output gates is y.
    - The values W assigns to the intermediate gates correspond to the correct operation of the gates.
    - Clearly there is a w such that C(x, w)=1 iff there is a correct transcript for C.

Sketch of 2-Prover MIP for Arithmetic Circuit SAT

[Blumberg, Thaler, Vu, Walfish, 2014]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

    Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.

Sketch of 2-Prover MIP for Arithmetic Circuit SAT

[Blumberg, Thaler, Vu, Walfish, 2014]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z}: \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that:

    Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
  - V checks this by running sum-check protocol with P_1 to compute

$$0 \stackrel{?}{=} \sum_{(a,b,c)\in\{0,1\}^{3\log S}} g^2_{x,Z}(a,b,c).$$

Sketch of 2-Prover MIP for Arithmetic Circuit SAT

[Blumberg, Thaler, Vu, Walfish, 2014]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that: Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
  - V checks this by running sum-check protocol with P_1 to compute
    - $$0 \stackrel{?}{=} \sum_{(a,b,c)\in\{0,1\}^{3\log S}} g^2_{x,Z}(a,b,c).$$
  - To perform final check in sum-check protocol, V needs to evaluate $g^2_{x,Z}$ at a random point. But this requires evaluating Z at a random point, and Z only "exists" in P_1's head.
    - So V asks P_2 for the evaluation of Z.
    - Soundness analysis of sum-check is valid as long as P_2's claim about Z is consistent with a **low-degree** polynomial. So V also runs a **low-degree test** with P_1 and P_2.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
- Let $\text{add}(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is an addition gate.
- Let $\text{mult}(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is a mult gate.
- Let $\text{io}(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff gate $\mathbf{a}$ is in the explicit input x and $(\mathbf{b},\mathbf{c})=(\mathbf{0},\mathbf{0})$, or if $\mathbf{a}$ is an output gate and $\mathbf{b}$ and $\mathbf{c}$ are in-neighbors of $\mathbf{a}$.
- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if $\mathbf{a}$ is an input gate, $y_{\mathbf{a}}$ if $\mathbf{a}$ is an output gate, and 0 otherwise.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
- Let add($\mathbf{a,b,c}$) output 1 iff $(\mathbf{b,c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is an addition gate.
- Let mult($\mathbf{a,b,c}$) output 1 iff $(\mathbf{b,c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is a mult gate.
- Let io($\mathbf{a,b,c}$) output 1 iff gate $\mathbf{a}$ is in the explicit input x and $(\mathbf{b,c})=(\mathbf{0,0})$,

  or if $\mathbf{a}$ is an output gate and $\mathbf{b}$ and $\mathbf{c}$ are in-neighbors of $\mathbf{a}$.
- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if $\mathbf{a}$ is an input gate, $y_{\mathbf{a}}$ if $\mathbf{a}$ is an output gate, and 0 otherwise.
- Key Lemma: For $G_{x,W} : \{0,1\}^{3\log S} \to \mathbf{F}$ defined below, W is a correct transcript on x iff $G_{x,W}(\mathbf{a,b,c}) = 0$ for all $(\mathbf{a,b,c})$ in $\{0,1\}^{3\log S}$.

$$G_{x,W}(\mathbf{a,b,c}) := \text{io}(\mathbf{a,b,c}) \bullet (I_x(\mathbf{a}) - W(\mathbf{a})) + \text{add}(\mathbf{a,b,c})(W(\mathbf{a}) - (W(\mathbf{b}) + W(\mathbf{c})) + \text{mult}(\mathbf{a,b,c}) \bullet (W(\mathbf{a}) - W(\mathbf{b}) \bullet W(\mathbf{c}))$$

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c) = 0 \;\forall\; (a,b,c) \in \{0,1\}^{3\log S}$.

- Let add$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})$=(in$_1(\mathbf{a})$, in$_2(\mathbf{a})$) and gate $\mathbf{a}$ is an addition gate.
- Let mult$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})$=(in$_1(\mathbf{a})$, in$_2(\mathbf{a})$) and gate $\mathbf{a}$ is a mult gate.
- Let io$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff gate $\mathbf{a}$ is in the explicit input x and $(\mathbf{b},\mathbf{c})=(\mathbf{0},\mathbf{0})$,

  or if $\mathbf{a}$ is an output gate and $\mathbf{b}$ and $\mathbf{c}$ are in-neighbors of $\mathbf{a}$.

- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if $\mathbf{a}$ is an input gate, $y_{\mathbf{a}}$ if $\mathbf{a}$ is an output gate, and 0 otherwise.
- Key Lemma: For $G_{x,W} : \{0,1\}^{3\log S} \to \mathbf{F}$ defined below, W is a correct transcript on x iff $G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c}) = 0$ for all $(\mathbf{a},\mathbf{b},\mathbf{c})$ in $\{0,1\}^{3\log S}$.

$$G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c}) := \mathrm{io}(\mathbf{a},\mathbf{b},\mathbf{c}) \bullet (\mathrm{I_x}(\mathbf{a})\text{-}\mathrm{W}(\mathbf{a})) + \mathrm{add}(\mathbf{a},\mathbf{b},\mathbf{c})(\mathrm{W}(\mathbf{a})\text{-}(\mathrm{W}(\mathbf{b})+\mathrm{W}(\mathbf{c})) + \mathrm{mult}(\mathbf{a},\mathbf{b},\mathbf{c}) \bullet (\mathrm{W}(\mathbf{a})\text{-}\mathrm{W}(\mathbf{b}) \bullet \mathrm{W}(\mathbf{c}))$$

- So we define:

$$g_{x,Z}(\mathbf{a},\mathbf{b},\mathbf{c}) = \widetilde{\mathrm{io}}(\mathbf{a},\mathbf{b},\mathbf{c}) \bullet (\widetilde{\mathrm{I}}_{\mathrm{x}}(\mathbf{a})\text{-}\mathrm{Z}(\mathbf{a})) + \widetilde{\mathrm{add}}(\mathbf{a},\mathbf{b},\mathbf{c})(\mathrm{Z}(\mathbf{a})\text{-}(\mathrm{Z}(\mathbf{b})+\mathrm{Z}(\mathbf{c})) + \widetilde{\mathrm{mult}}(\mathbf{a},\mathbf{b},\mathbf{c}) \bullet (\mathrm{Z}(\mathbf{a})\text{-}\mathrm{Z}(\mathbf{b}) \bullet \mathrm{Z}(\mathbf{c}))$$

Costs of the 2-Prover MIP for Non-Deterministic Circuit Evaluation

Rounds	V Time	P_1 Time	P_2 Time
$\log S$	$O(n + \log^2 S)$	$O(S)$	$O(S \log S)$

[RRR16] and Open Questions

Another General-Purpose Doubly-Efficient Interactive Proof

What We Really Want

- In the cloud computing scenario at the start of the talk, we really wanted the following:
  1. V asks P to run some computer program on her data.
  2. P proves that she correctly ran the program on the data.
- V should not do much more work than read the input.
- P should not do much more work than run the program.

What We Really Want

- In the cloud computing scenario at the start of the talk, we really wanted the following:
  1. V asks P to run some computer program on her data.
  2. P proves that she correctly ran the program on the data.
- V should not do much more work than read the input.
- P should not do much more work than run the program.
  - If the program runs in time T, and space s, then P should run in time $O(T)$ and space $O(s)$.

What We Really Want

- In the cloud computing scenario at the start of the talk, we really wanted the following:
  1. V asks P to run some computer program on her data.
  2. P proves that she correctly ran the program on the data.
- V should not do much more work than read the input.
- P should not do much more work than run the program.
  - If the program runs in time T, and space s, then P should run in time $O(T)$ and space $O(s)$.
- The GKR protocol only achieves a linear-time for V **parallelizable** programs.

What We Really Want

- In the cloud computing scenario at the start of the talk, we really wanted the following:
  1. V asks P to run some computer program on her data.
  2. P proves that she correctly ran the program on the data.
- V should not do much more work than read the input.
- P should not do much more work than run the program.
  - If the program runs in time T, and space s, then P should run in time $O(T)$ and space $O(s)$.
- Unfortunately, we **cannot** hope for V to run in time $O(n)$ for space-intensive computations.
  - If f has an **interactive proof** with V runtime c, then f can be solved in space $\tilde{O}(c)$.
  - So we can only hope to achieve a linear-time verifier for problems solvable in quadratic space.

What We Really Want

- In the cloud computing scenario at the start of the talk, we really wanted the following:
  1. V asks P to run some computer program on her data.
  2. P proves that she correctly ran the program on the data.
- V should not do much more work than read the input.
- P should not do much more work than run the program.
  - If the program runs in time T, and space s, then P should run in time $O(T)$ and space $O(s)$.
- Unfortunately, we **cannot** hope for V to run in time $O(n)$ for space-intensive computations.
  - If f has an **interactive proof** with V runtime c, then f can be solved in space $\tilde{O}(c)$.
  - So we can only hope to achieve a linear-time verifier for problems solvable in quadratic space.
  - [RRR16] come close to achieving the best we can hope for.

[RRR16]

- Let f be a problem solvable in time T and space s. Then for any constant $\varepsilon > 0$, f has an interactive proof where:
  - V runs in time $\tilde{O}(n + T^{\varepsilon} \cdot \text{poly}(s))$.
  - P runs in time $\tilde{O}\,(T^{1+\varepsilon} \cdot \text{poly}(s))$.

[RRR16]

- Let f be a problem solvable in time T and space s. Then for any constant $\varepsilon > 0$, f has an interactive proof where:
  - V runs in time $\tilde{O}(n + T^{\varepsilon} \cdot \mathrm{poly}(s))$.
  - P runs in time $\tilde{O}\,(T^{1+\varepsilon} \cdot \mathrm{poly}(s))$.
- In particular, if $T = \mathrm{poly}(n)$ and s is a small enough polynomial in n, then this is a doubly-efficient interactive proof system.

[RRR16]

- Let f be a problem solvable in time T and space s. Then for any constant $\varepsilon > 0$, f has an interactive proof where:
  - V runs in time $\tilde{O}(n + T^{\varepsilon} \cdot \mathrm{poly}(s))$.
  - P runs in time $\tilde{O}\,(T^{1+\varepsilon} \cdot \mathrm{poly}(s))$.
- In particular, if $T = \mathrm{poly}(n)$ and s is a small enough polynomial in n, then this is a doubly-efficient interactive proof system.
- The number of rounds is **constant**.
  - More precisely, it is $\exp\left(\frac{1}{\varepsilon}\right)$.

Open Questions (Theory)

- Improve V's runtime in [RRR16] from $\tilde{O}(n + T^{\varepsilon} \cdot \mathrm{poly}(s))$ to $\tilde{O}(n + \mathrm{poly}(s, \log T))$? Maybe even $\tilde{O}(n + s \cdot \log T)$?
- Improve the round complexity from $\exp\left(\frac{1}{\varepsilon}\right)$ to $\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$?

Open Questions (Theory)

- Improve V's runtime in [RRR16] from $\tilde{O}(n + T^{\varepsilon} \cdot \mathrm{poly}(s))$ to $\tilde{O}(n + \mathrm{poly}(s, \log T))$? Maybe even $\tilde{O}(n + s \cdot \log T)$?
- Improve the round complexity from $\exp\left(\frac{1}{\varepsilon}\right)$ to $\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$?
- Give an interactive proof for **batch-verification of NP statements**?
  - Under standard complexity assumptions, interactive proofs cannot be **succinct** [GH98, GVW01].
    - I.e., for a general **NP** relation, cannot do much better than just having the prover send the **NP** witness to the verifier.

Open Questions (Theory)

- Improve V's runtime in [RRR16] from $\tilde{O}(n + T^{\varepsilon} \cdot \text{poly}(s))$ to $\tilde{O}(n + \text{poly}(s, \log T))$? Maybe even $\tilde{O}(n + s \cdot \log T)$?
- Improve the round complexity from $\exp\left(\frac{1}{\varepsilon}\right)$ to $\text{poly}\left(\frac{1}{\varepsilon}\right)$?
- Give an interactive proof for **batch-verification of NP statements**?
  - Under standard complexity assumptions, interactive proofs cannot be **succinct** [GH98, GVW01].
    - I.e., for a general **NP** relation, cannot do much better than just having the prover send the **NP** witness to the verifier.
  - Open: given k instances of the same **NP** problem, is there an interactive proof for verifying that the answer to all k instances is YES, with communication that grows sublinearly with k?

A Parting Remark

- We've seen some fundamental limitations of interactive proofs.
  - V can't run in linear time for space-intensive problems.
  - They cannot be succinct.
  - They are interactive.
  - They are not publicly verifiable.

A Parting Remark

- We've seen some fundamental limitations of interactive proofs.
  - V can't run in linear time for space-intensive problems.
  - They cannot be succinct.
  - They are interactive.
  - They are not publicly verifiable.
- All of these limitations can be addressed by combining interactive proofs with cryptography.
  - This yields **succinct non-interactive arguments**.
  - See tomorrow's talks.
- There are many practically-relevant open questions about the best way to combine interactive proofs with cryptography.

THANK YOU!

A Simple Triangles Protocol with Sub-Optimal Prover Time

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6} \cdot \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik}$.
- Fastest known algorithm runs in matrix-multiplication time, currently about $n^{2.37}$.

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6} \cdot \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik}$.
- The Protocol:
  - View A as a function mapping $\{0,1\}^{\log n}\times\{0,1\}^{\log n}$ to $\boldsymbol{F}$.
  - Recall that $\tilde{A}$ denotes the multilinear extension of A.
  - Define the polynomial $g(X,Y,Z) = \tilde{A}(X,Y)\,\tilde{A}(Y,\mathrm{Z})\,\tilde{A}(X,\mathrm{Z})$
  - Apply the sum-check protocol to g to compute:

$$\sum_{(a,b,c)\,\in\{0,1\}^{3\log n}} g(a,b,c)$$

Counting Triangles

- Input: $A \in \{0,1\}^{n\times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\frac{1}{6} \cdot \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik}$.
- The Protocol:
  - View A as a function mapping $\{0,1\}^{\log n}\times\{0,1\}^{\log n}$ to $\boldsymbol{F}$.
  - Recall that $\tilde{A}$ denotes the multilinear extension of A.
  - Define the polynomial $g(X,Y,Z) = \tilde{A}(X,Y)\,\tilde{A}(Y,Z)\,\tilde{A}(X,Z)$
  - Apply the sum-check protocol to g to compute:

$$\sum_{(a,b,c)\,\in\{0,1\}^{3\log n}} g(a,b,c)$$

- Costs:
  - Total communication is $O(\log n)$, V runtime is $O(n^2)$, P runtime is $O(n^3)$.
  - V's runtime dominated by evaluating:

$$g(r_1,r_2,r_3) = \tilde{A}(r_1,r_2)\,\tilde{A}(r_2,r_3)\,\tilde{A}(r_1,r_3).$$