Interactive Proofs

Justin Thaler
Georgetown University

Talk Outline

1. Definition of Interactive Proofs
2. The Power of Randomness
    - Reed-Solomon Fingerprinting
    - Freivalds' Protocol for Verifying Matrix Products
3. Technical Concepts: low-degree extensions, arithmetization
4. The Sum-Check Protocol
5. An Interactive Proof for #SAT
6. Doubly-Efficient Interactive Proofs

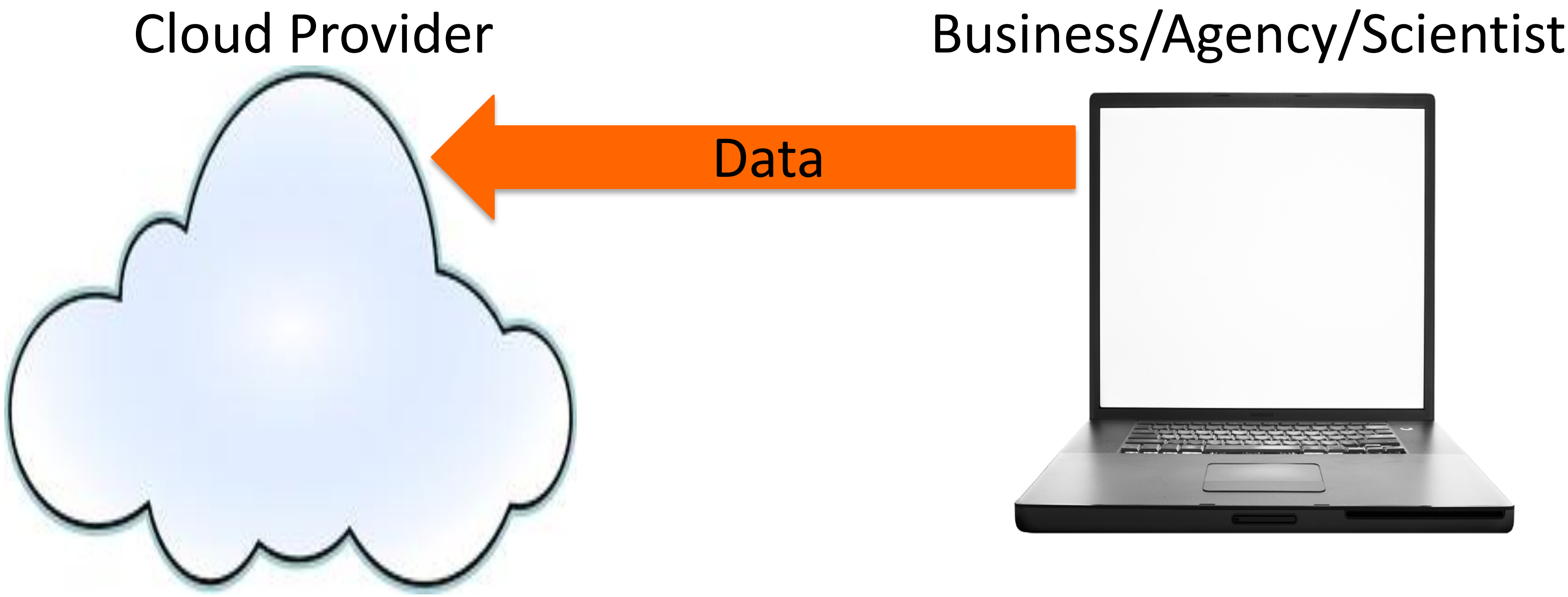
Interactive Proofs: Motivation and Model

Interactive Proofs

Cloud Provider

Business/Agency/Scientist

Interactive Proofs

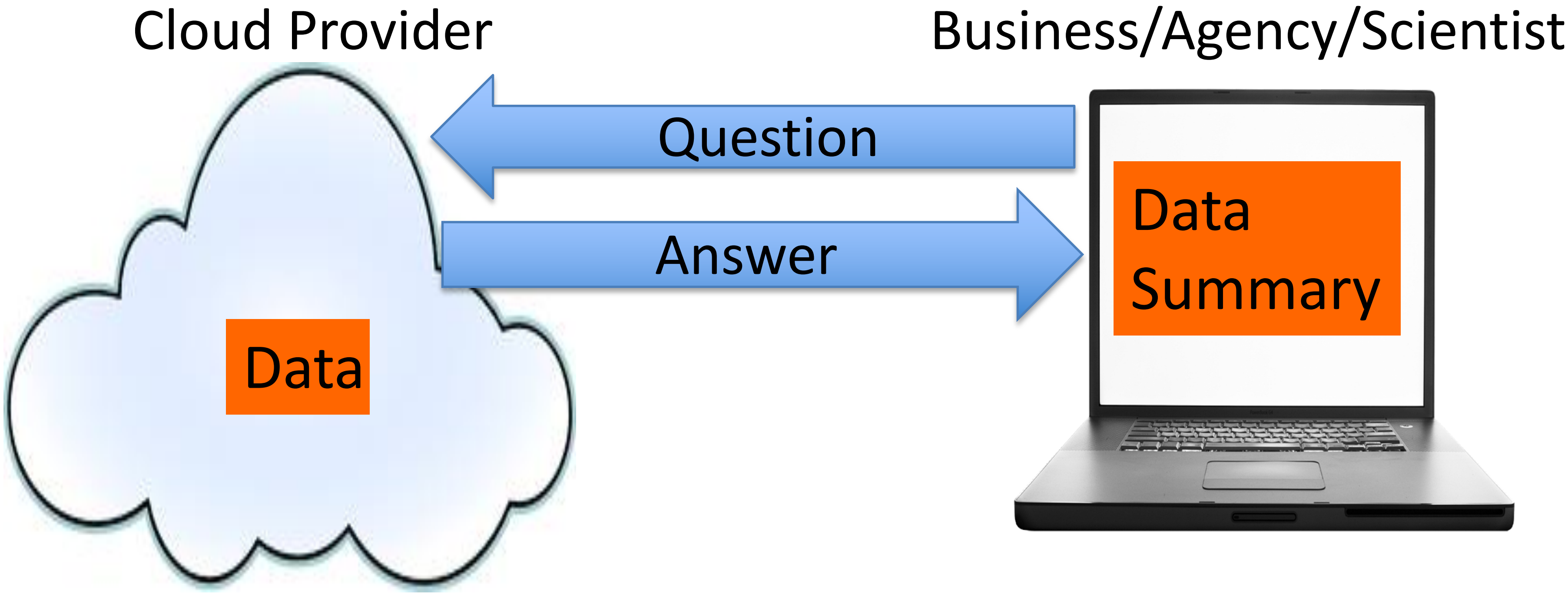


Interactive Proofs

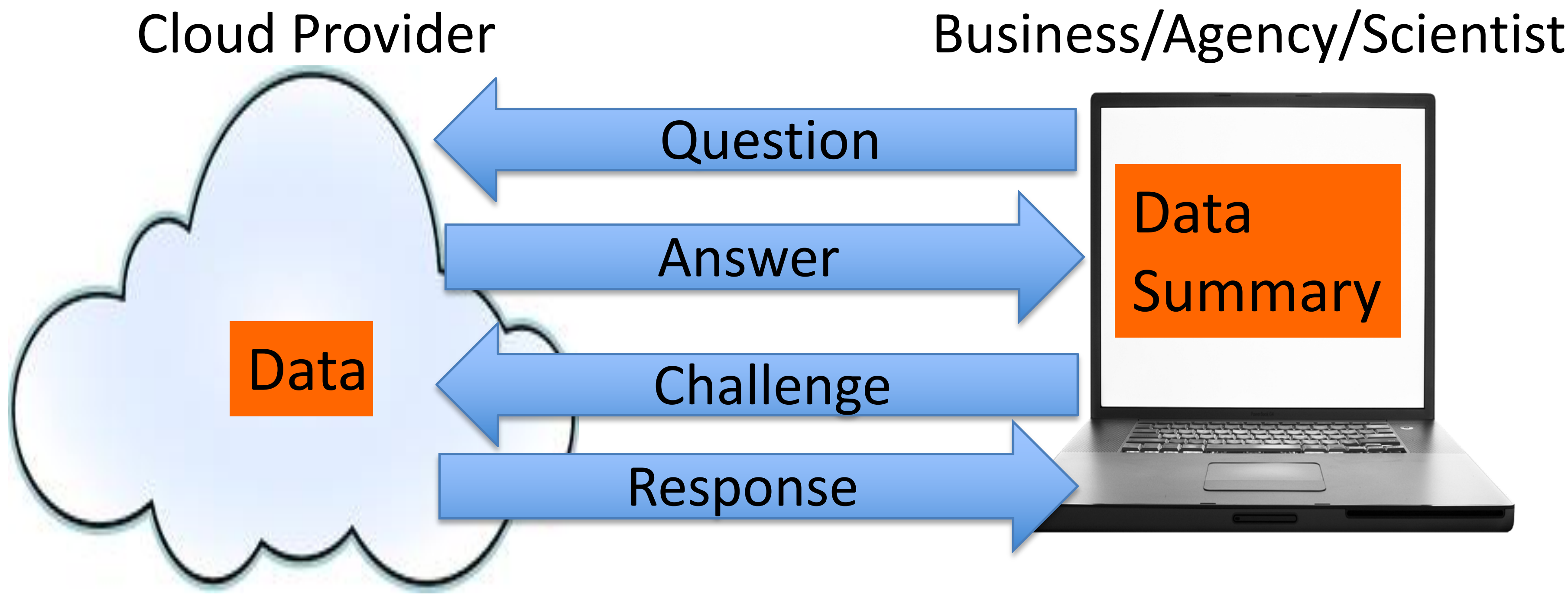
Cloud Provider

Business/Agency/Scientist

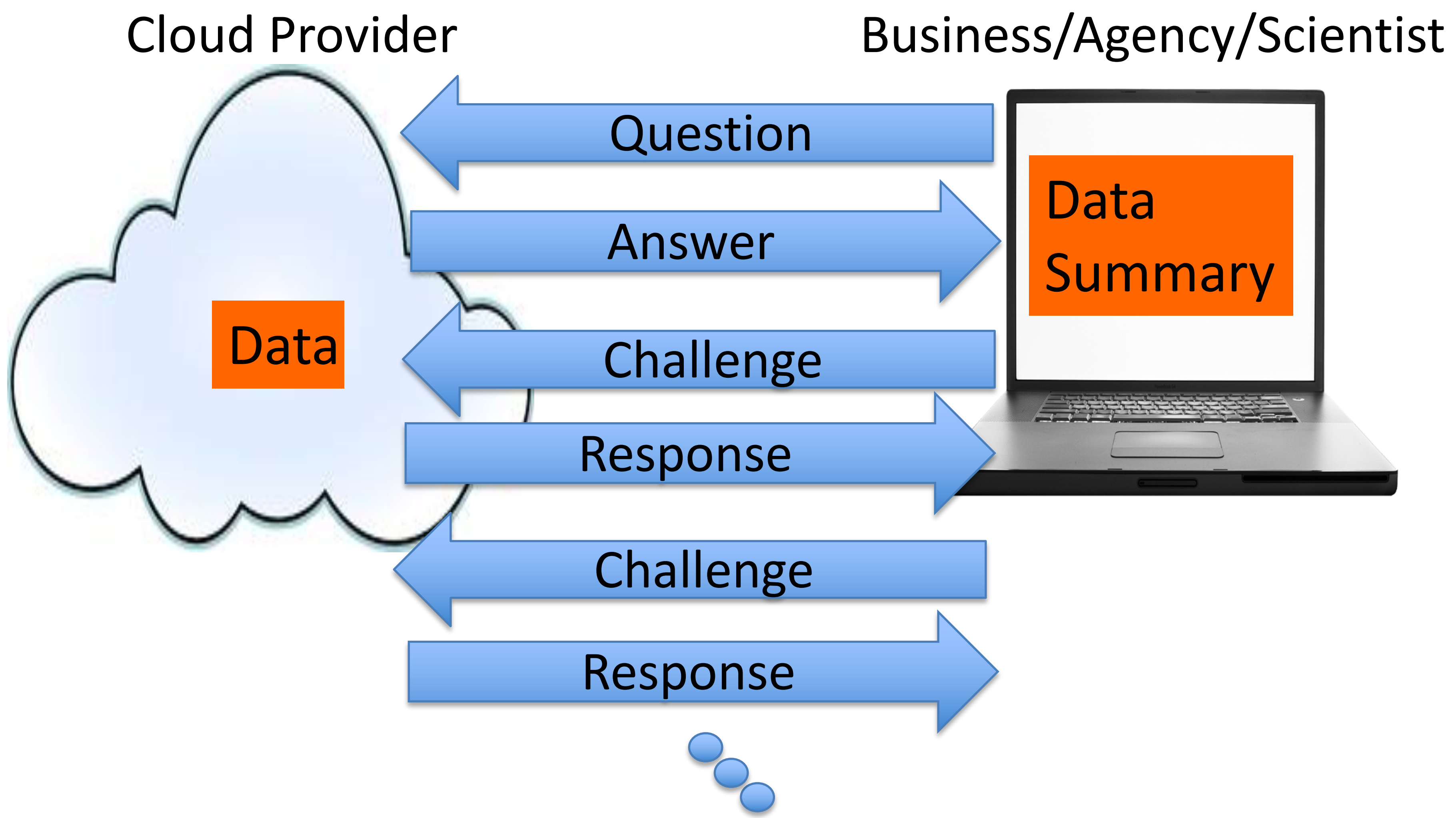
Interactive Proofs



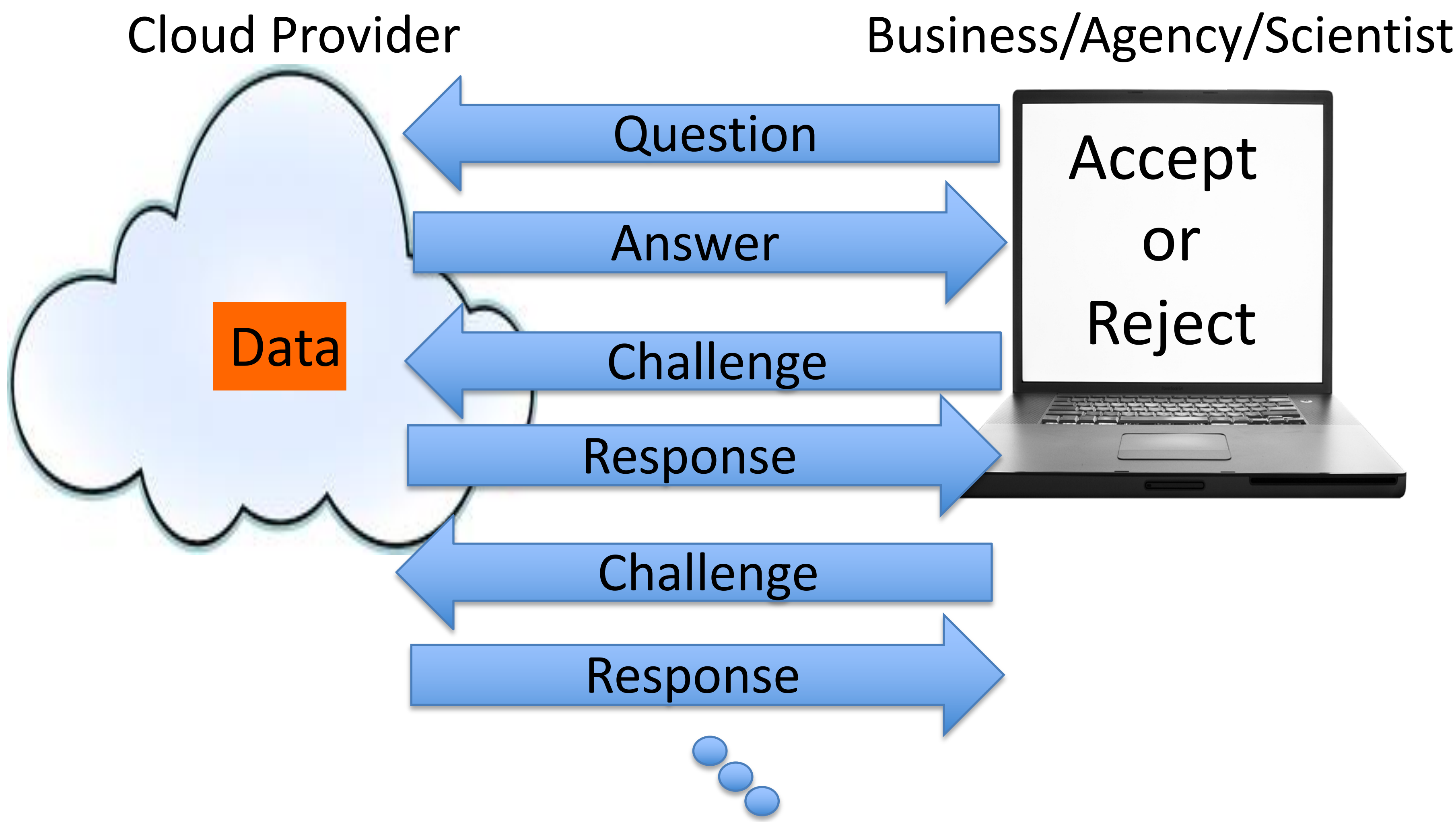
Interactive Proofs



Interactive Proofs



Interactive Proofs



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.

The Power of Randomness: A Demonstration

EQUALITY Testing

Alice

Bob

$\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$

$\boldsymbol{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$

Alice and Bob's Goal: Determine whether $\boldsymbol{a} = \boldsymbol{b}$, while exchanging as few bits as possible.

EQUALITY Testing

Alice

Bob

$\boldsymbol{a} = (a_1, \dots, a_n) \in \{0,1\}^n$

$\boldsymbol{b} = (b_1, \dots, b_n) \in \{0,1\}^n$

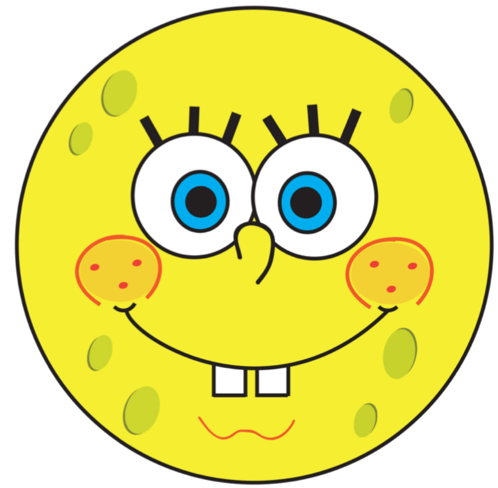
Trivial solution: Alice sends $\boldsymbol{a}$ to Bob, who checks whether $\boldsymbol{a} = \boldsymbol{b}$.
Communication cost is n.

EQUALITY Testing

Alice

$\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$

Bob



$\boldsymbol{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$

Fact: Trivial solution is optimal amongst deterministic protocols.

A Logarithmic Cost Randomized Solution

Randomized EQUALITY Testing Protocol

- Notation:
  - Let $\boldsymbol{F}$ be any finite field with $|\boldsymbol{F}| \geq n^2$.
  - Interpret each a_i, b_i as elements of $\boldsymbol{F}$.
  - Let $p(x) = \sum_{i=1}^{n} a_i\, x^i$ and $q(x) = \sum_{i=1}^{n} b_i\, x^i$.

Randomized EQUALITY Testing Protocol

- Notation:
  - Let $\boldsymbol{F}$ be any finite field with $|\boldsymbol{F}| \geq n^2$.
  - Interpret each a_i, b_i as elements of $\boldsymbol{F}$.
  - Let $p(x) = \sum_{i=1}^{n} a_i\, x^i$ and $q(x) = \sum_{i=1}^{n} b_i\, x^i$.

- The Protocol:
  - Alice picks a random $r \in \boldsymbol{F}$ and sends $(r, p(r))$ to Bob.
  - Bob outputs EQUAL if $p(r) = q(r)$. Otherwise he outputs NOT-EQUAL.

Randomized EQUALITY Testing Protocol

- Notation:
  - Let $\boldsymbol{F}$ be any finite field with $|\boldsymbol{F}| \geq n^2$.
  - Interpret each a_i, b_i as elements of $\boldsymbol{F}$.
  - Let $p(x) = \sum_{i=1}^{n} a_i\, x^i$ and $q(x) = \sum_{i=1}^{n} b_i\, x^i$.

- The Protocol:
  - Alice picks a random $r \in \boldsymbol{F}$ and sends $(r, p(r))$ to Bob.
  - Bob outputs EQUAL if $p(r) = q(r)$. Otherwise he outputs NOT-EQUAL.

- Total communication: $O(\log |\boldsymbol{F}|) = O(\log n)$ bits.

Randomized EQUALITY Testing Protocol

- Notation:
  - Let $\boldsymbol{F}$ be any finite field with $|\boldsymbol{F}| \geq n^2$.
  - Interpret each a_i, b_i as elements of $\boldsymbol{F}$.
  - Let $p(x) = \sum_{i=1}^{n} a_i\, x^i$ and $q(x) = \sum_{i=1}^{n} b_i\, x^i$.

- The Protocol:
  - Alice picks a random $r \in \boldsymbol{F}$ and sends $(r, p(r))$ to Bob.
  - Bob outputs EQUAL if $p(r) = q(r)$. Otherwise he outputs NOT-EQUAL.

- Total communication: $O(\log|\boldsymbol{F}|) = O(\log n)$ bits.
- Call $p(r)$ the *Reed-Solomon fingerprint* of the vector $\boldsymbol{a}$ at r.

Correctness Analysis

- Claim 1: if $\boldsymbol{a} = \boldsymbol{b}$, then Bob outputs EQUAL with probability 1.

- Claim 2: $\boldsymbol{a} \neq \boldsymbol{b}$, then Bob outputs NOT-EQUAL with probability at least $1 - \frac{1}{n}$ over the choice of $r \in \boldsymbol{F}$.

Correctness Analysis

- Claim 1: if $\boldsymbol{a} = \boldsymbol{b}$, then Bob outputs EQUAL with probability 1.
  - Proof: Since $\boldsymbol{a} = \boldsymbol{b}$, p and q are the same polynomial, so $p(r) = q(r)$ for all $r \in \boldsymbol{F}$.

- Claim 2: $\boldsymbol{a} \neq \boldsymbol{b}$, then Bob outputs NOT-EQUAL with probability at least $1 - \frac{1}{n}$ over the choice of $r \in \boldsymbol{F}$.

Correctness Analysis

- Claim 2: $\boldsymbol{a} \neq \boldsymbol{b}$, then Bob outputs NOT-EQUAL with probability at least $1 - \frac{1}{n}$ over the choice of $r \in \boldsymbol{F}$.

Correctness Analysis

- Claim 2: $\boldsymbol{a} \neq \boldsymbol{b}$, then Bob outputs NOT-EQUAL with probability at least $1 - \frac{1}{n}$ over the choice of $r \in \boldsymbol{F}$.

FACT: Let $p \neq q$ be univariate polynomials of degree at most n. Then p and q agree on at most n inputs. Equivalently:

$$\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{n}{|\boldsymbol{F}|}.$$

Correctness Analysis

- Claim 2: $\boldsymbol{a} \neq \boldsymbol{b}$, then Bob outputs NOT-EQUAL with probability at least $1 - \frac{1}{n}$ over the choice of $r \in \boldsymbol{F}$.

FACT: Let $p \neq q$ be univariate polynomials of degree at most n. Then p and q agree on at most n inputs. Equivalently:

$$\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{n}{|\boldsymbol{F}|}.$$

- If $\boldsymbol{a} \neq \boldsymbol{b}$, then p and q are **not** the same polynomial. By **FACT**, the probability Alice picks an r such that $p(r) = q(r)$ is at most $\frac{n}{|\boldsymbol{F}|} \leq \frac{n}{n^2} \leq \frac{1}{n}$.

Main Takeaways

1. Any two distinct low-degree polynomials differ almost everywhere: if $p \neq q$ then $\Pr_{r \in \boldsymbol{F}}[p(r) = q(r)] \leq \frac{n}{|\boldsymbol{F}|}$ where n bounds the degree of p and q.
   - Corollary: If two low-degree polynomials agree at a randomly chosen input, it is "safe" to believe they are the **same** polynomial.

2. Interpreting inputs as low-degree polynomials is powerful.
   - If two inputs differ **at all**, then once interpreted as polynomials, they differ **almost everywhere**.

Freivalds' Protocol for Verifying Matrix Products

Demonstrating the Power of Randomness in Verifiable Computing

Verifying Matrix Multiplication

- Input is two matrices $A, B \in \boldsymbol{F}^{n\times n}$. Goal is to compute $A \cdot B$.
- Fastest known algorithm runs in time about $n^{2.37}$.

Verifying Matrix Multiplication

- Input is two matrices $\mathrm{A}, \mathrm{B} \in \boldsymbol{F}^{n \times n}$. Goal is to compute $\mathrm{A} \cdot \mathrm{B}$.
- Fastest known algorithm runs in time about $n^{2.37}$.
- What if an untrusted prover P claims that the answer is a matrix C? Can V **verify** that $C = \mathrm{A} \cdot \mathrm{B}$ in $O(n^2)$ time?

Verifying Matrix Multiplication

- Input is two matrices $\mathrm{A}, \mathrm{B} \in \boldsymbol{F}^{n \times n}$. Goal is to compute $\mathrm{A} \cdot \mathrm{B}$.
- Fastest known algorithm runs in time about $n^{2.37}$.
- What if an untrusted prover P claims that the answer is a matrix C? Can V **verify** that $C = \mathrm{A} \cdot \mathrm{B}$ in $O(n^2)$ time?
- Yes!

Verifying Matrix Multiplication

- **The Protocol:**
  1. V picks a random $r \in \boldsymbol{F}$ and lets $\boldsymbol{x} = (r, r^2, \ldots, r^n)$.
  2. V computes $C \cdot \boldsymbol{x}$ and $(\text{AB}) \cdot \boldsymbol{x}$, accepting iff they are equal.

Verifying Matrix Multiplication

- **The Protocol:**
  1. V picks a random $r \in \boldsymbol{F}$ and lets $\boldsymbol{x} = (r, r^2, \ldots, r^n)$.
  2. V computes $C \cdot \boldsymbol{x}$ and $(\mathrm{AB}) \cdot \boldsymbol{x}$, accepting iff they are equal.

- Runtime Analysis:
  - V's runtime dominated by computing 3 matrix-vector products, each of which takes $O(n^2)$ time.
    - $C \cdot \boldsymbol{x}$ is one matrix-vector multiplication.
    - $(\mathrm{AB}) \cdot \boldsymbol{x} = \mathrm{A} \cdot (\mathrm{B} \cdot \boldsymbol{x})$ takes two matrix-vector multiplications.

Correctness Analysis

- Claim 1: If $C = \mathrm{A} \cdot \mathrm{B}$ then V accepts with probability 1.
- Claim 2: If $C \neq \mathrm{A} \cdot \mathrm{B}$, then V rejects with probability at least

$$1 - \frac{n}{|\boldsymbol{F}|} \geq 1 - 1/n.$$

Correctness Analysis

- Claim 1: If $C = A \cdot B$ then V accepts with probability 1.
- Claim 2: If $C \neq A \cdot B$, then V rejects with probability at least

$$1 - \frac{n}{|\boldsymbol{F}|} \geq 1 - 1/n.$$

- Proof of Claim 2:
  - Recall that $\boldsymbol{x} = (r, r^2, \dots, r^n)$.
  - $(C \cdot \boldsymbol{x})_i = \sum_{j=1}^{n} C_{ij} r^j$ is the Reed-Solomon fingerprint at r of the ith row of C.

Correctness Analysis

- Claim 1: If $C = \mathrm{A} \cdot \mathrm{B}$ then V accepts with probability 1.
- Claim 2: If $C \neq \mathrm{A} \cdot \mathrm{B}$, then V rejects with probability at least

$$1 - \frac{n}{|\boldsymbol{F}|} \geq 1 - 1/n.$$

- Proof of Claim 2:
  - Recall that $\boldsymbol{x} = (r, r^2, \ldots, r^n)$.
  - $(C \cdot \boldsymbol{x})_i = \sum_{j=1}^{n} C_{ij} r^j$ is the Reed-Solomon fingerprint at r of the ith row of C.
  - Similarly, $((\mathrm{AB}) \cdot \boldsymbol{x})_i$ is the Reed-Solomon fingerprint at r of the ith row of AB.

Correctness Analysis

- Claim 1: If $C = \mathrm{A} \cdot \mathrm{B}$ then V accepts with probability 1.
- Claim 2: If $C \neq \mathrm{A} \cdot \mathrm{B}$, then V rejects with probability at least
$$1 - \frac{n}{|\boldsymbol{F}|} \geq 1 - 1/n.$$
  - Proof of Claim 2:
    - Recall that $\boldsymbol{x} = (r, r^2, \ldots, r^n)$.
    - $(C \cdot \boldsymbol{x})_i = \sum_{j=1}^{n} C_{ij} r^j$ is the Reed-Solomon fingerprint at r of the ith row of C.
    - Similarly, $((\mathrm{AB}) \cdot \boldsymbol{x})_i$ is the Reed-Solomon fingerprint at r of the ith row of AB.
    - So if even one row of C does not equal the corresponding row of AB, the fingerprints for that row will differ with probability at least $1 - 1/n$, causing V to reject.

Interactive Proof Techniques: Preliminaries

Schwartz-Zippel Lemma

- Recall **FACT:** Let $p \neq q$ be univariate polynomials 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 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 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 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 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 $f: \{0,1\}^\ell \to \boldsymbol{F}$, 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 $\tilde{f}$.

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.

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

1	2
8	10

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

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

$$\tilde{f}(x_1, x_2) = (1 - x_1)(1 - x_2) + 2(1 - x_1)x_2 + 8x_1(1 - x_2) + 10x_1x_2$$

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

Can check:

$\tilde{f}(0,0) = 1$

$\tilde{f}(0,1) = 2$

$\tilde{f}(1,0) = 8$

$\tilde{f}(1,1) = 10$

Another (non-multilinear) extension of f:

$$g(x_1, x_2) = -x_1^2 + x_1 x_2 + 8\,x_1 + x_2 + 1$$

1	2	3	4	5	6	…
8	10	12	14	16	18	…
13	16	19	22	25	28	…
16	20	24	28	32	36	…
17	22	27	32	37	42	…
16	22	28	34	40	44	…
⋮	⋮	⋮	⋮	⋮	⋮	

Can check:

$g(0,0) = 1$

$g(0,1) = 2$

$g(1,0) = 8$

$g(1,1) = 10$

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 $\tilde{f}(r)$.

- Note: If f is "structured", there may extensions g for which $g(r)$ can be evaluated **much** faster than $O(2^{\ell})$-time.

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 $\tilde{f}(r)$.

- Note: If f is "structured", there may extensions g for which $g(r)$ can be evaluated **much** faster than $O(2^\ell)$-time.
  - We will see an example later when covering arithmetization of Boolean formulae.

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:

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

- **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.

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 $1 - \frac{\ell \cdot d}{|\boldsymbol{F}|}$, where d is the maximum degree of g in any variable.
- Proof is by induction on the number of variables ℓ.
  - Base case: $\ell = 1$. In this case, P sends a single message $s_1(X_1)$ claimed to equal $g(X_1)$. V picks r_1 at random, checks that $s_1(r_1) = g(r_1)$.
  - By **Fact**, if $s_1 \neq g$, then $\Pr_{r_1 \in \boldsymbol{F}}[s_1(r_1) = g(r_1)] \leq \frac{d}{|\boldsymbol{F}|}$.

Soundness: Inductive Case

- Inductive case: $\ell > 1$.
  - Recall: P's first message $s_1(X_1)$ is claimed to equal

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

  - Then V picks a random r_1 and sends r_1 to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.

Soundness: Inductive Case

- Inductive case: $\ell > 1$.
  - Recall: P's first message $s_1(X_1)$ is claimed to equal

    $$H_1\,(X_1) \coloneqq \sum_{b_2\in\{0,1\}} \dots \sum_{b_\ell\in\{0,1\}} g(X_1, b_2, \dots, b_\ell).$$
  - Then V picks a random r_1 and sends r_1 to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.
- By **Fact**, if $s_1 \neq H_1$, then $\Pr_{r_1\in \boldsymbol{F}}[s_1(r_1) = H(r_1)] \leq \frac{d}{|\boldsymbol{F}|}$.

Soundness: Inductive Case

- Inductive case: $\ell > 1$.
  - Recall: P's first message $s_1(X_1)$ is claimed to equal
    $$H_1\ (X_1) \coloneqq \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell).$$
  - Then V picks a random r_1 and sends r_1 to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.
- By **Fact**, if $s_1 \neq H_1$, then $\Pr_{r_1 \in \boldsymbol{F}}[s_1(r_1) = H(r_1)] \leq \frac{d}{|\boldsymbol{F}|}$.
- If $s_1(r_1) \neq H(r_1)$, P is left to prove a false claim in the recursive call.

Soundness: Inductive Case

- Inductive case: $\ell > 1$.
  - Recall: P's first message $s_1(X_1)$ is claimed to equal
    $$H_1\ (X_1) \coloneqq \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell).$$
  - Then V picks a random r_1 and sends r_1 to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.
- By **Fact**, if $s_1 \neq H_1$, then $\Pr_{r_1 \in \boldsymbol{F}}[s_1(r_1) = H(r_1)] \leq \frac{d}{|\boldsymbol{F}|}$.
- If $s_1(r_1) \neq H(r_1)$, P is left to prove a false claim in the recursive call.
  - The recursive call applies sum-check to $g(r_1, X_2, \dots, X_\ell)$, which is ℓ-1 variate.
  - By induction, P fails to convince V in the recursive call with probability at least $1 - \frac{d(\ell-1)}{|\boldsymbol{F}|}$.

Soundness: Inductive Case

- Inductive case: $\ell > 1$.
  - Recall: P's first message $s_1(X_1)$ is claimed to equal

    $$H_1(X_1) \coloneqq \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \dots, b_\ell).$$
  - Then V picks a random r_1 and sends r_1 to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.
- By **Fact**, if $s_1 \neq H_1$, then $\Pr_{r_1 \in \boldsymbol{F}}[s_1(r_1) = H(r_1)] \leq \frac{d}{|\boldsymbol{F}|}$.
- If $s_1(r_1) \neq H(r_1)$, P is left to prove a false claim in the recursive call.
  - The recursive call applies sum-check to $g(r_1, X_2, \dots, X_\ell)$, which is ℓ-1 variate.
  - By induction, P fails to convince V in the recursive call with probability at least $1 - \frac{d(\ell-1)}{|\boldsymbol{F}|}$.
- **Summary:** if $s_1 \neq H_1$, the probability V accepts is at most:

$$\Pr_{r_1 \in \boldsymbol{F}}[s_1(r_1) = H(r_1)] + \Pr_{r_2, \dots, r_\ell \in \boldsymbol{F}}[\text{V accepts} | s_1(r_1) \neq H(r_1)]$$

$$\leq \frac{d}{|\boldsymbol{F}|} + \frac{d(\ell-1)}{|\boldsymbol{F}|} \leq \frac{d\ell}{|\boldsymbol{F}|}.$$

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 elements.

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 elements.
- V's runtime is:

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

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 elements.
- V's runtime is:

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

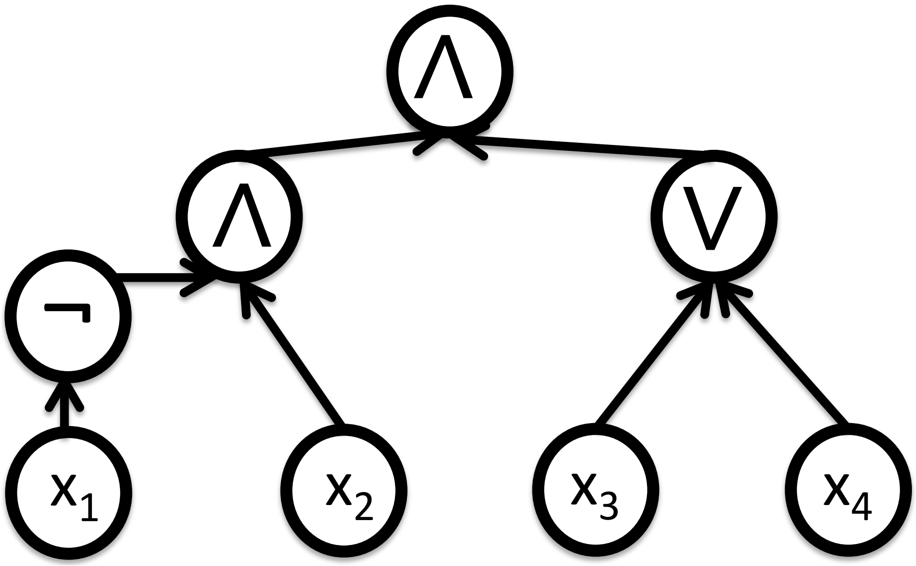
- P's runtime is at most:

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

First Application of Sum-Check: An IP For #SAT [LFKN]

#SAT Problem

- Let φ be a Boolean formula of size S over n variables.

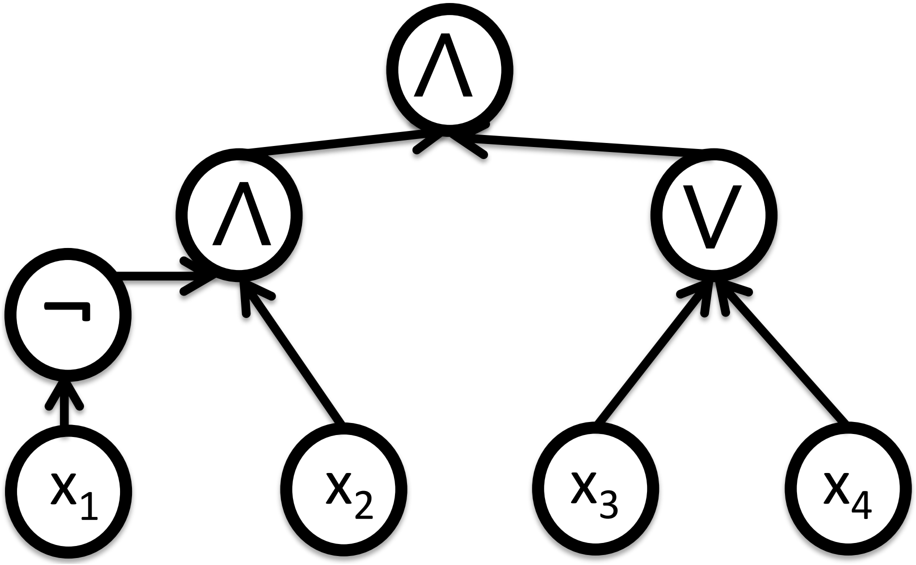


#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \rightarrow \{0, 1\}$. ($0$ interpreted as FALSE, 1 as TRUE).

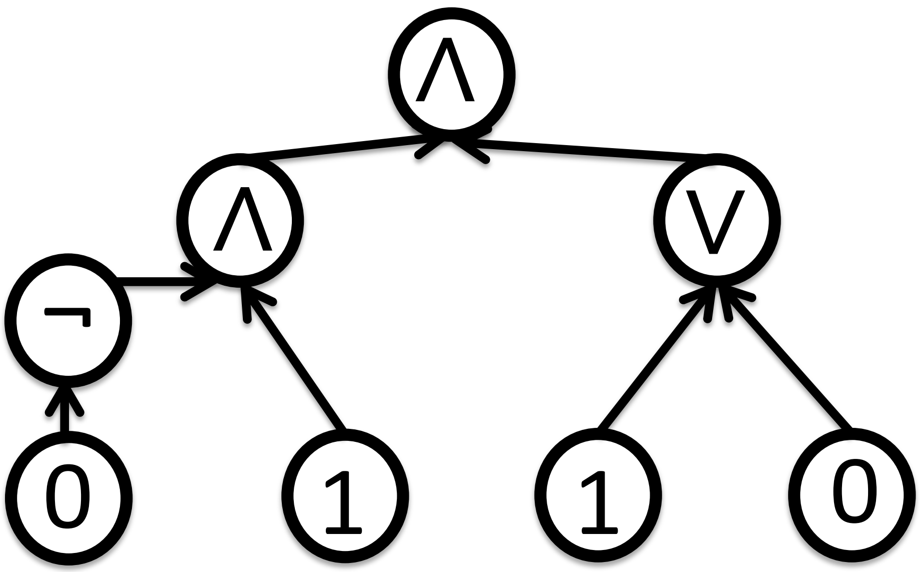
#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \rightarrow \{0,1\}$. (0 interpreted as FALSE, 1 as TRUE).



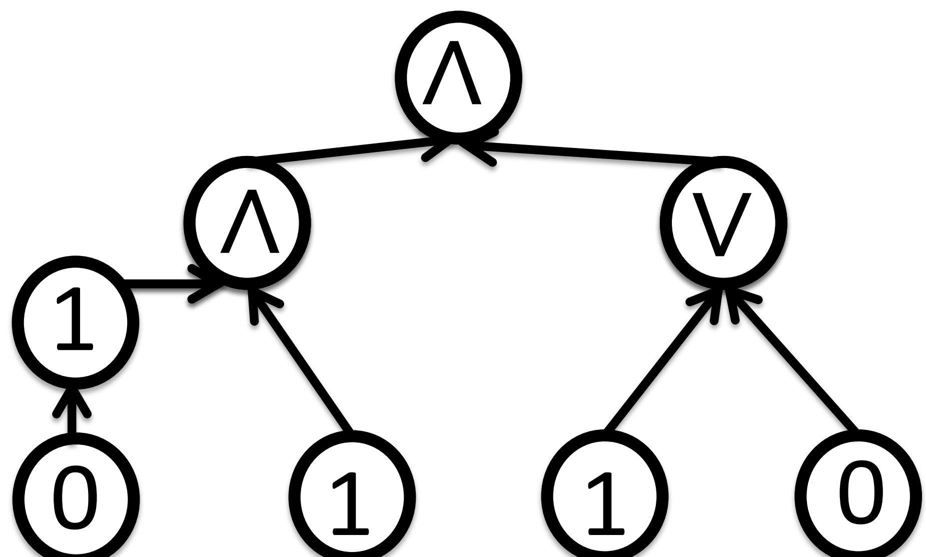
#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x \in \{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \rightarrow \{0,1\}$. (0 interpreted as FALSE, 1 as TRUE).



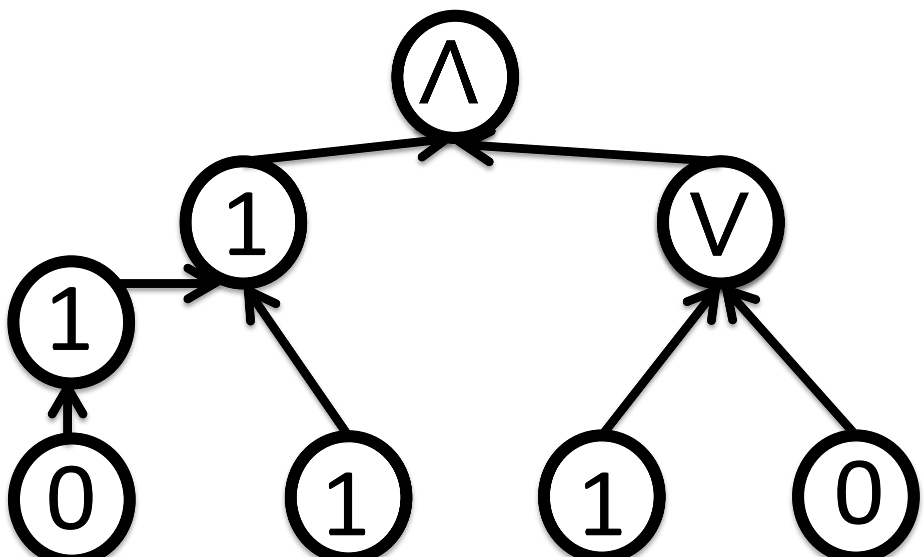
#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \rightarrow \{0,1\}$. (0 interpreted as FALSE, 1 as TRUE).



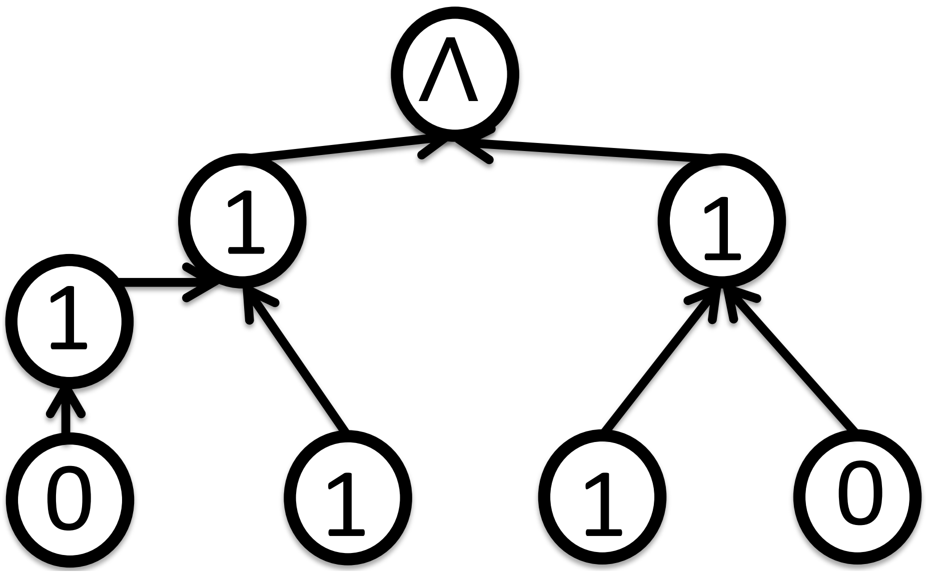
#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x \in \{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \to \{0, 1\}$. (0 interpreted as FALSE, 1 as TRUE).



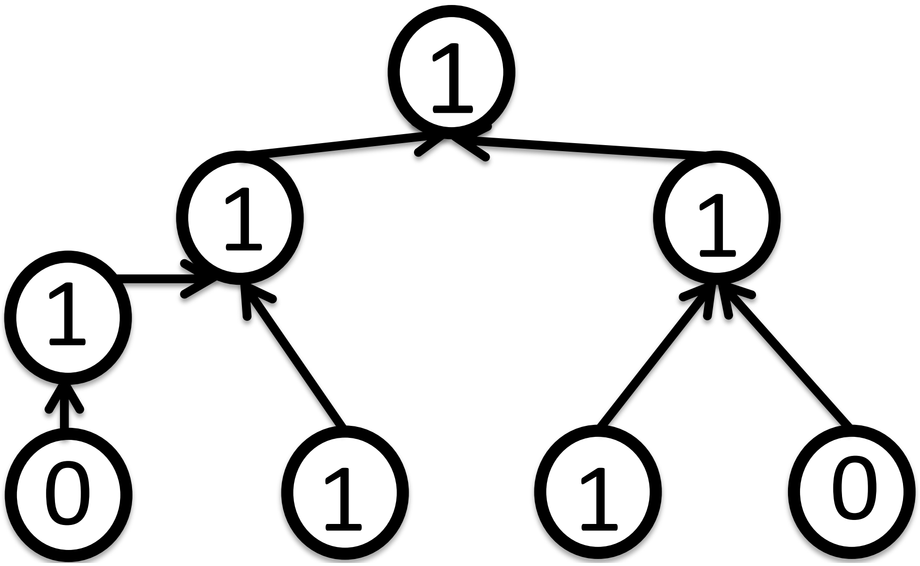
#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \rightarrow \{0,1\}$. (0 interpreted as FALSE, 1 as TRUE).



#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: count the number of satisfying assignments of φ.
- i.e., Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.
- In the sum above, we are viewing φ as a function mapping $\{0,1\}^n \to \{0,1\}$. (0 interpreted as FALSE, 1 as TRUE).



#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.

#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^n} \varphi(x)$.

- Protocol:
- Let g be an extension polynomial of φ.
- Apply the sum-check protocol to compute $\sum_{x \in \{0,1\}^n} g(x)$.

#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^n} \varphi(x)$.

- Protocol:
- Let g be an extension polynomial of $\boldsymbol{\varphi}$.
- Apply the sum-check protocol to compute $\sum_{x \in \{0,1\}^n} g(x)$.
  - Note: in final round of sum-check, V needs to compute $g(r)$ for some randomly chosen r in $\boldsymbol{F^n}$.
    - To control V's runtime, we need this to be fast.

#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^n} \varphi(x)$.

- Protocol:
- Let g be an extension polynomial of φ.
- Apply the sum-check protocol to compute $\sum_{x \in \{0,1\}^n} g(x)$.
  - Note: in final round of sum-check, V needs to compute $g(r)$ for some randomly chosen r in $\boldsymbol{F^n}$.
    - To control V's runtime, we need this to be fast.
  - To control communication and P and V's runtime, we need g to be "low-degree".

#SAT Problem

- Let φ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x\in\{0,1\}^n} \varphi(x)$.

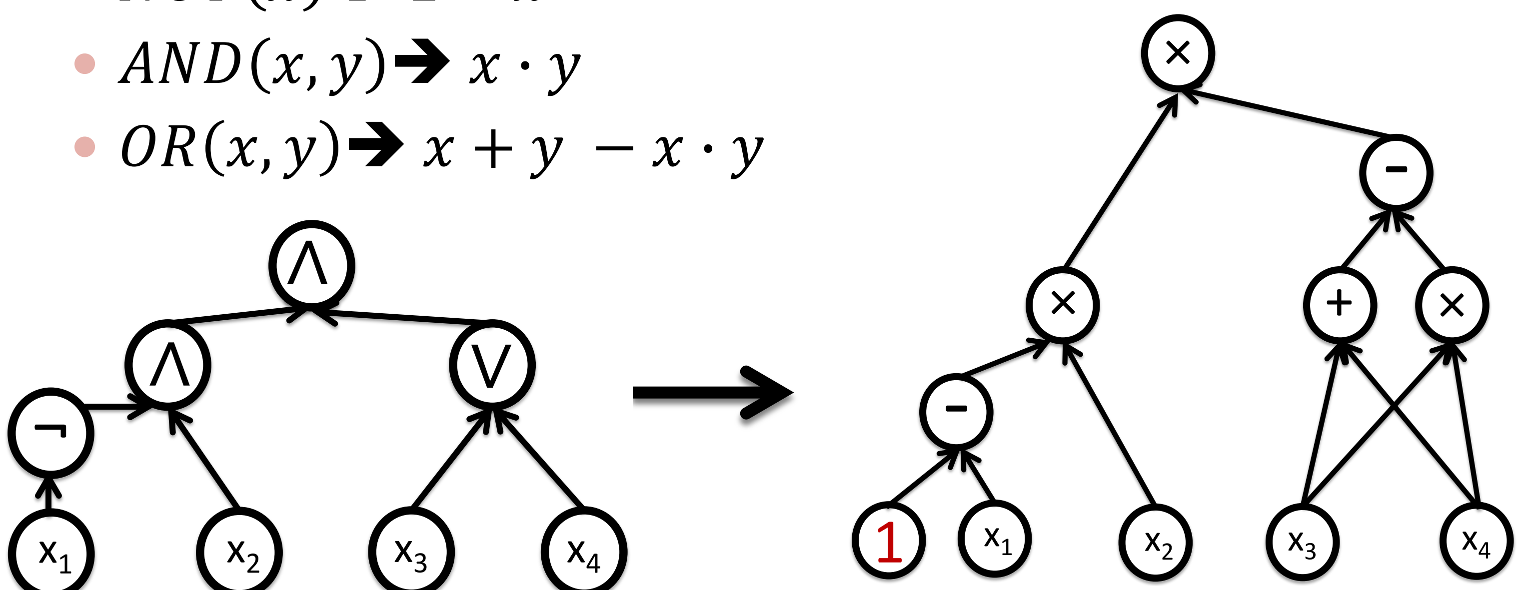
- Protocol:
- Let g be an extension polynomial of $\boldsymbol{\varphi}$.
- Apply the sum-check protocol to compute $\sum_{x\in\{0,1\}^n} g(x)$.
  - Note: in final round of sum-check, V needs to compute $g(r)$ for some randomly chosen r in $\boldsymbol{F^n}$.
    - To control V's runtime, we need this to be fast.
  - To control communication and P and V's runtime, we need g to be "low-degree".
  - Key question: how to construct the extension polynomial g?

Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$

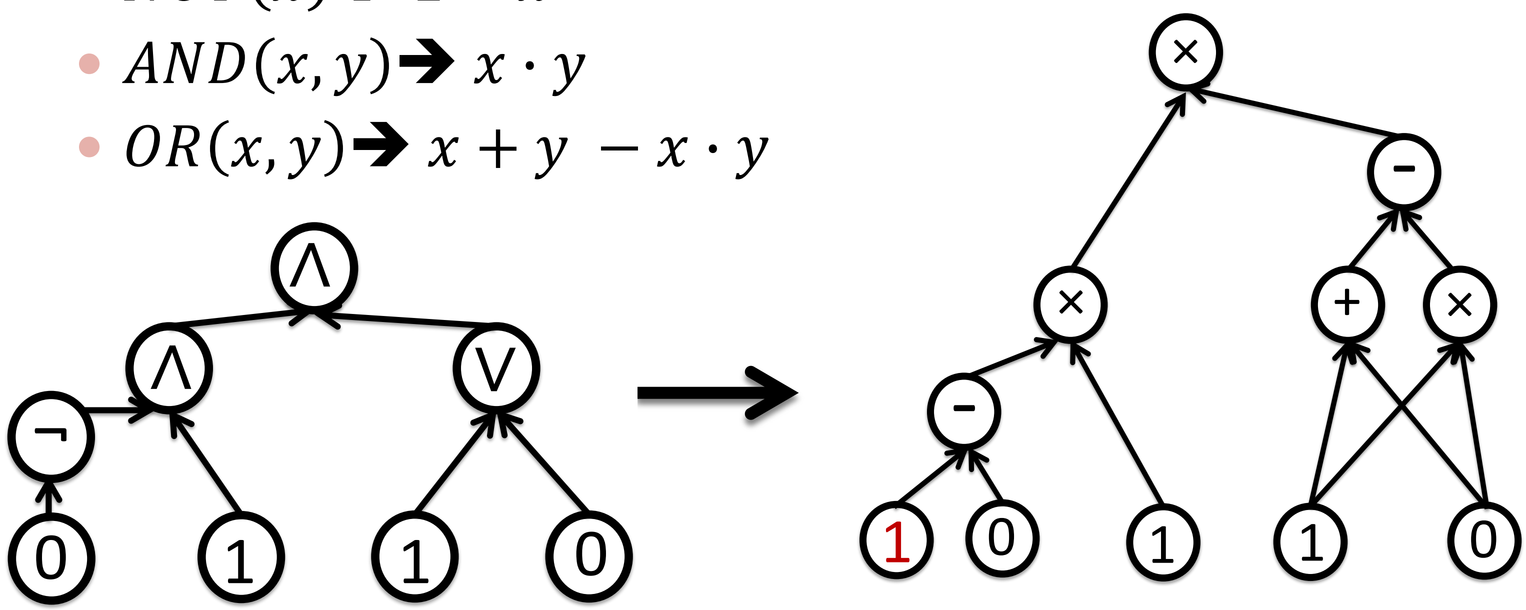
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x)$ ➔ $1 - x$
    - $AND(x, y)$ ➔ $x \cdot y$
    - $OR(x, y)$ ➔ $x + y - x \cdot y$



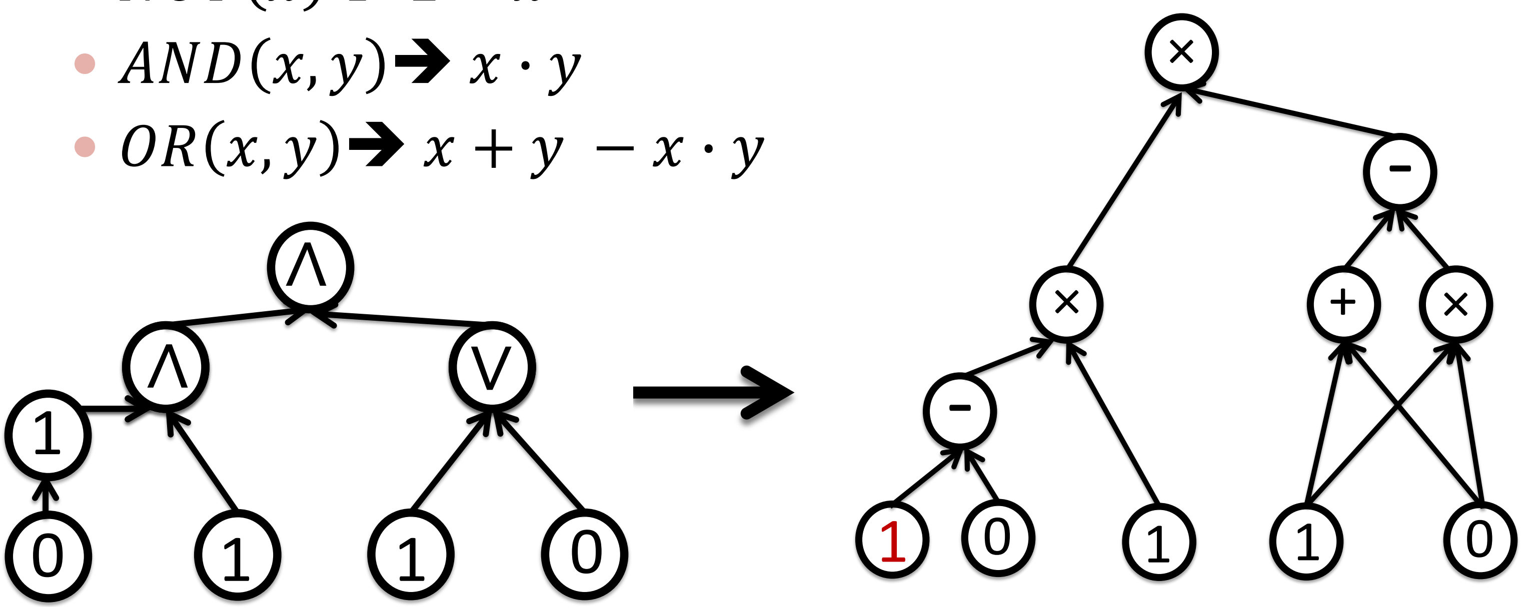
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



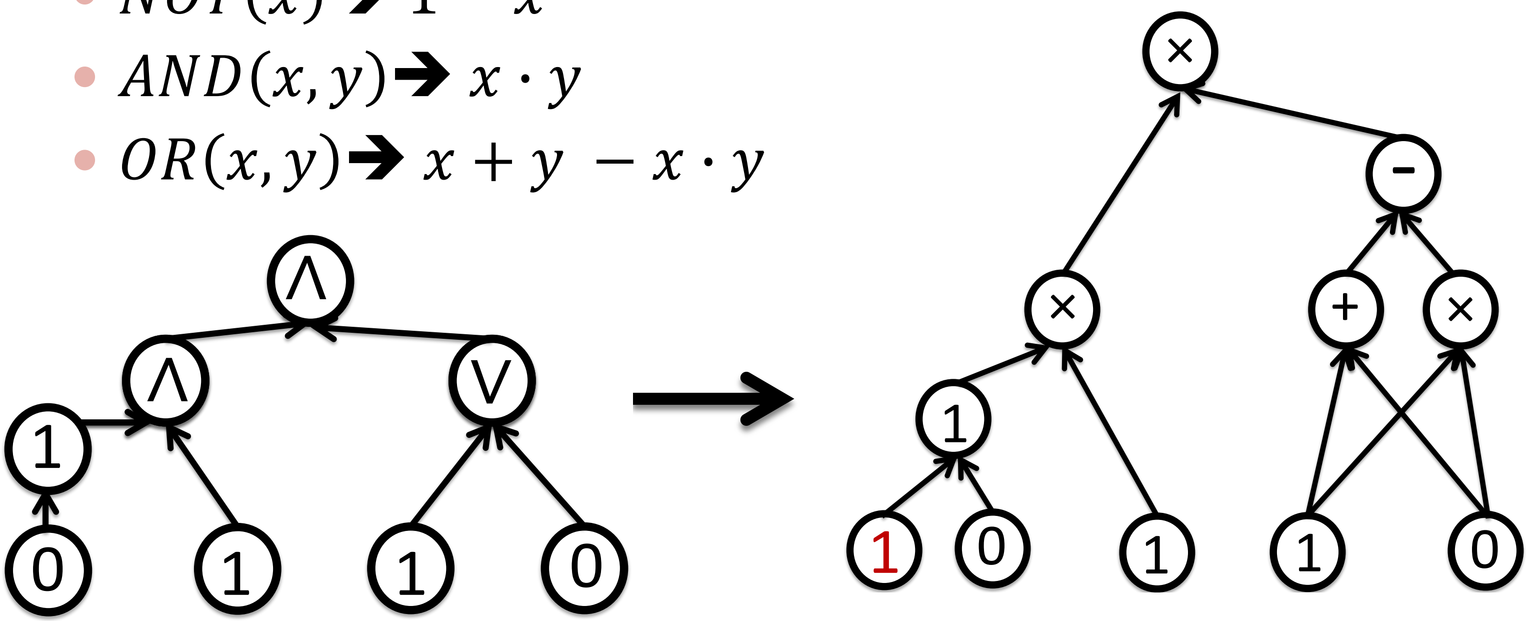
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x,y) \rightarrow x \cdot y$
    - $OR(x,y) \rightarrow x + y - x \cdot y$



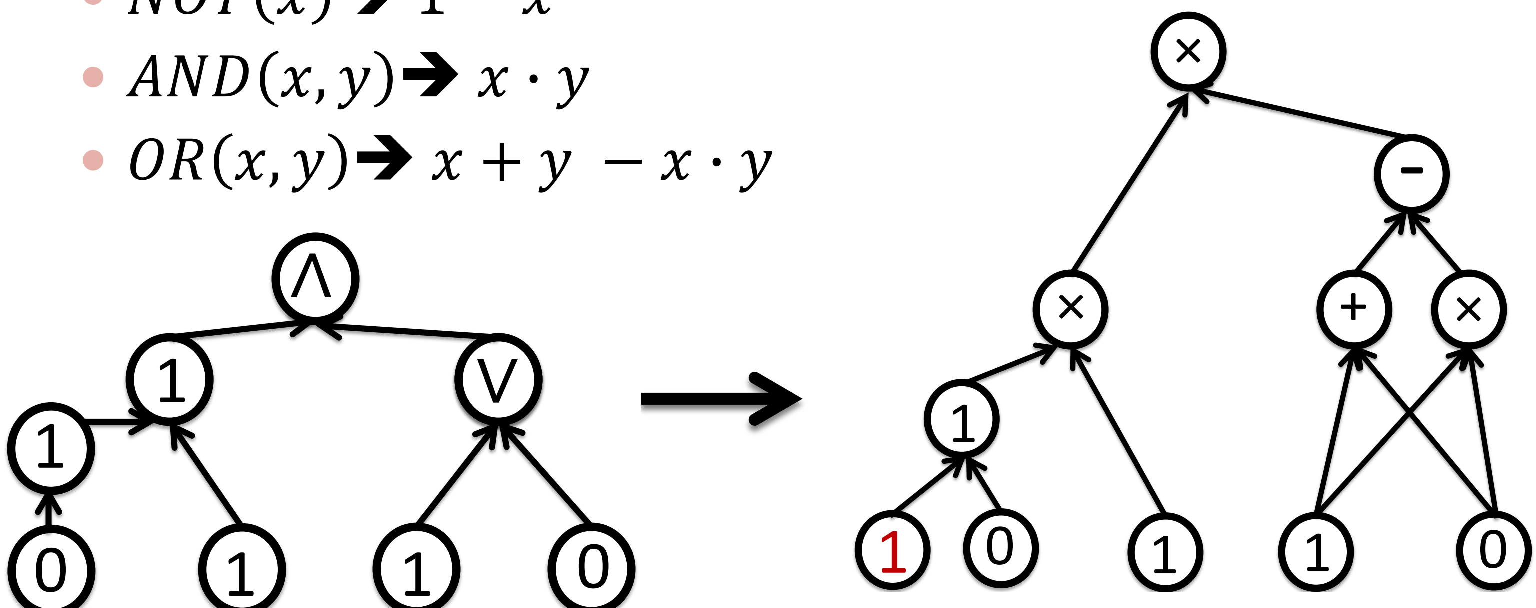
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x)$ ➔ $1 - x$
    - $AND(x, y)$ ➔ $x \cdot y$
    - $OR(x, y)$ ➔ $x + y - x \cdot y$



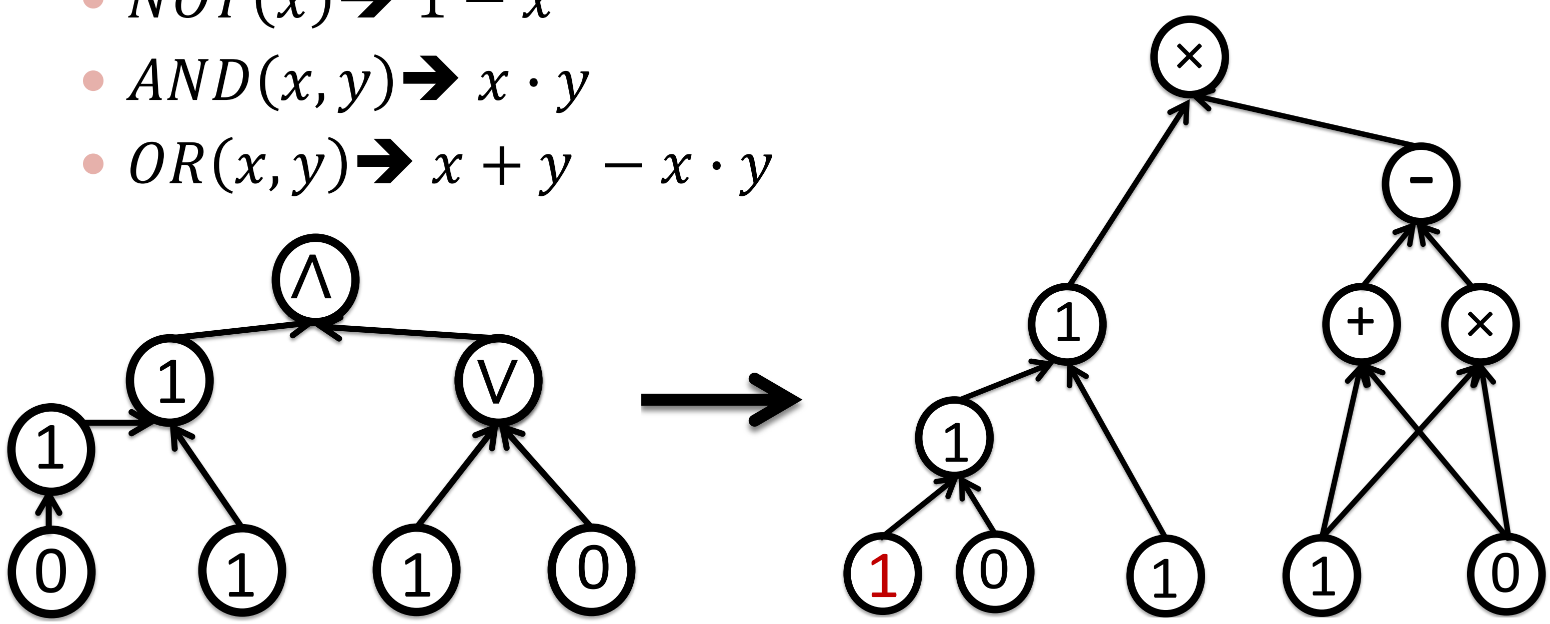
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



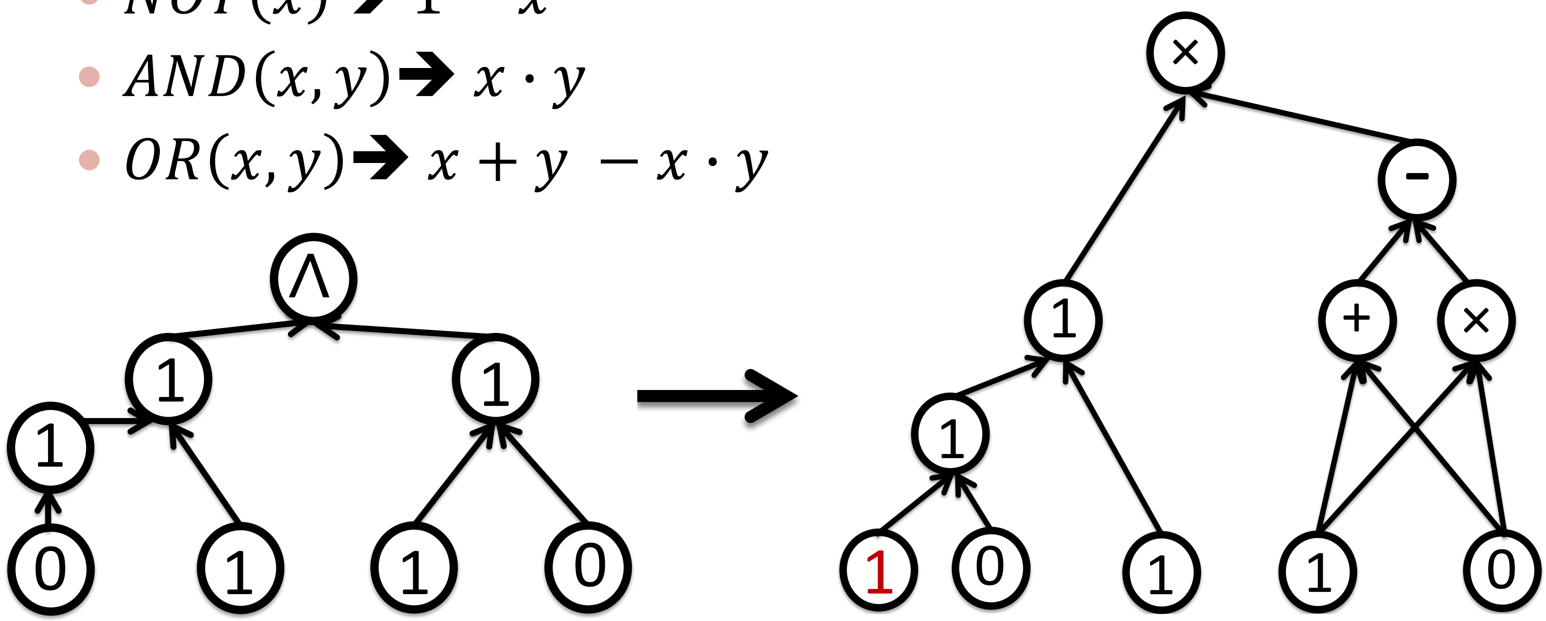
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



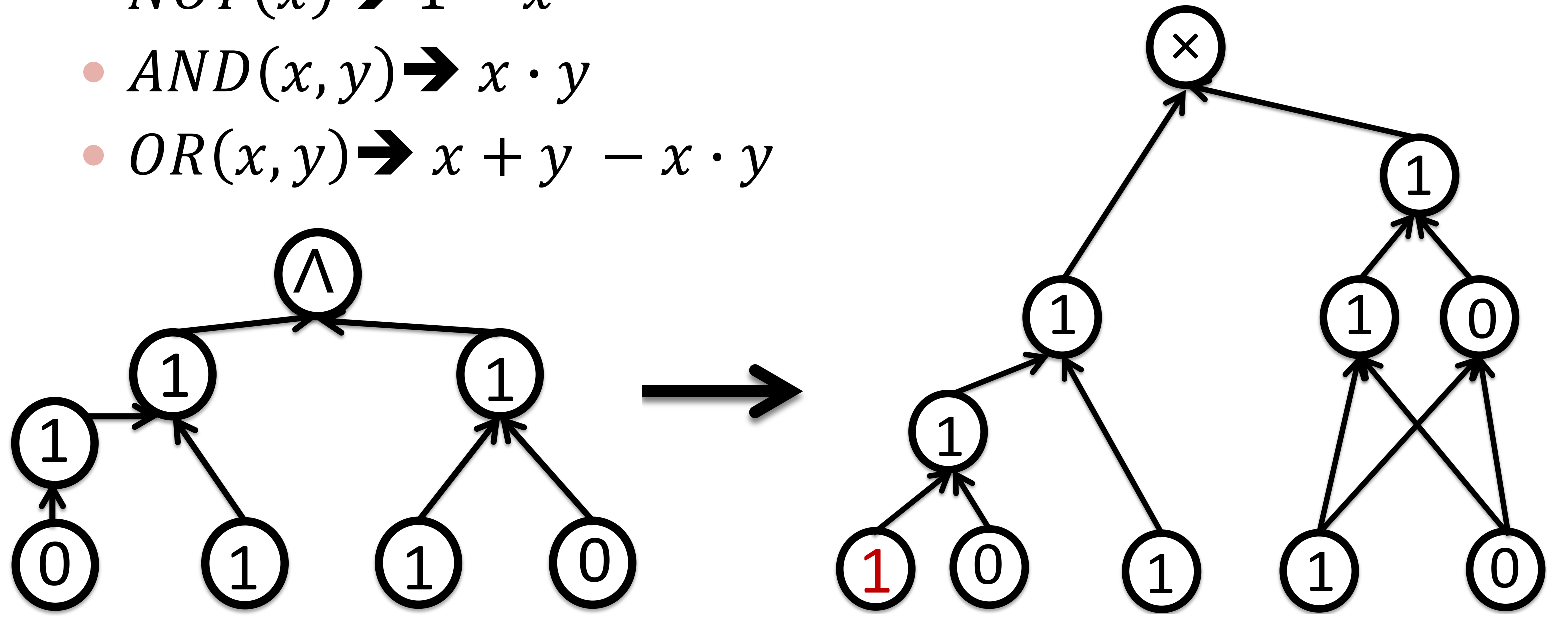
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



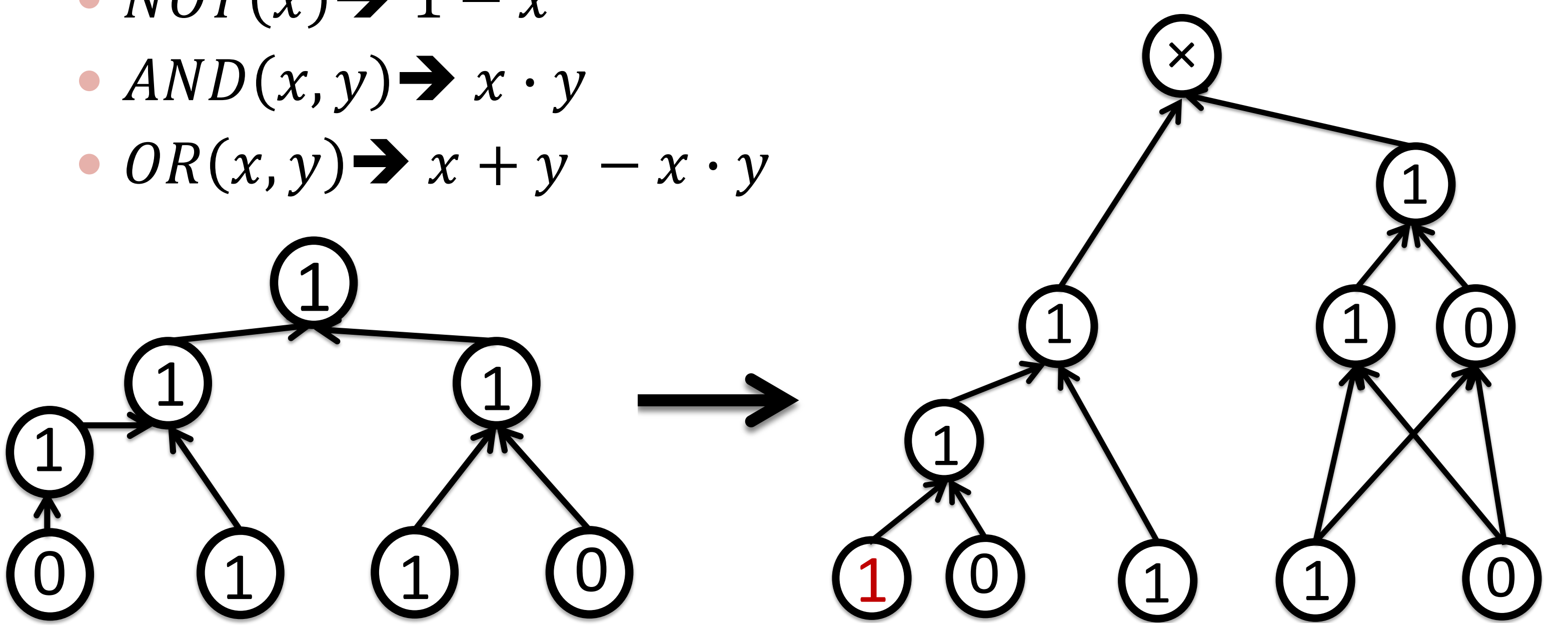
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



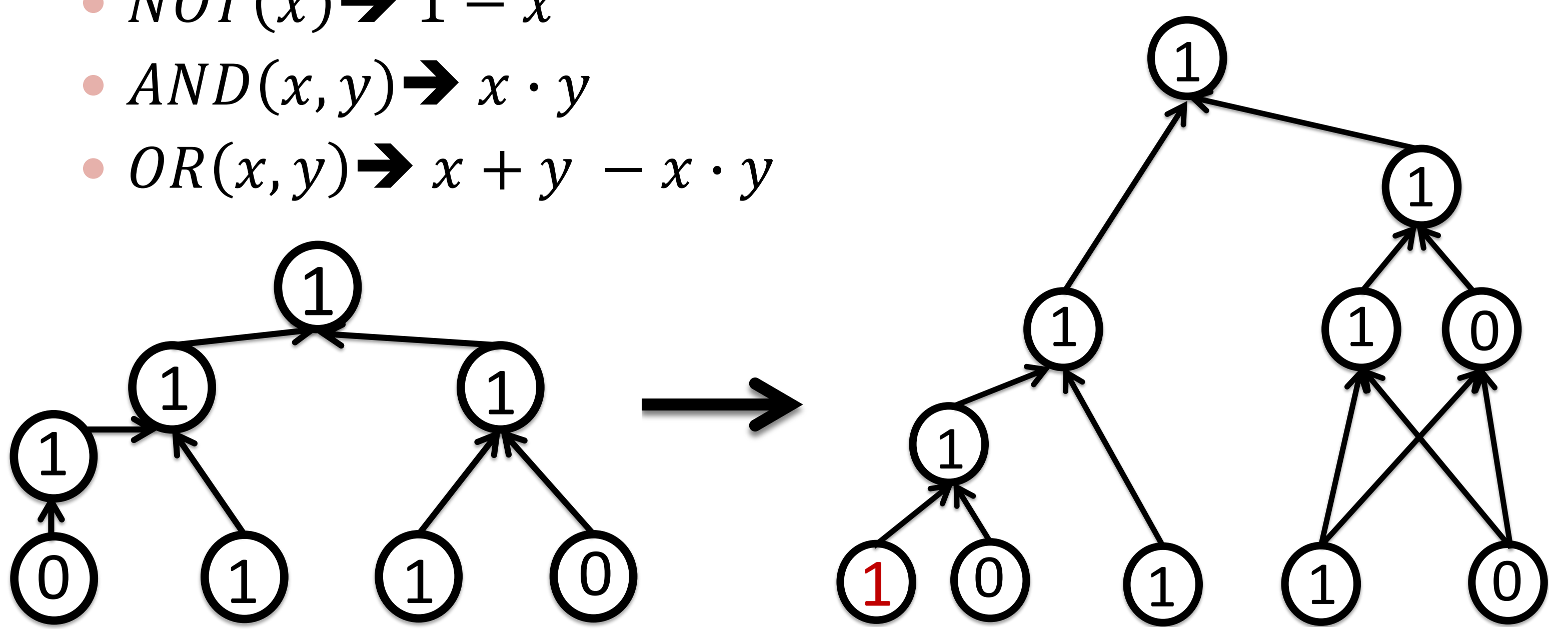
Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$

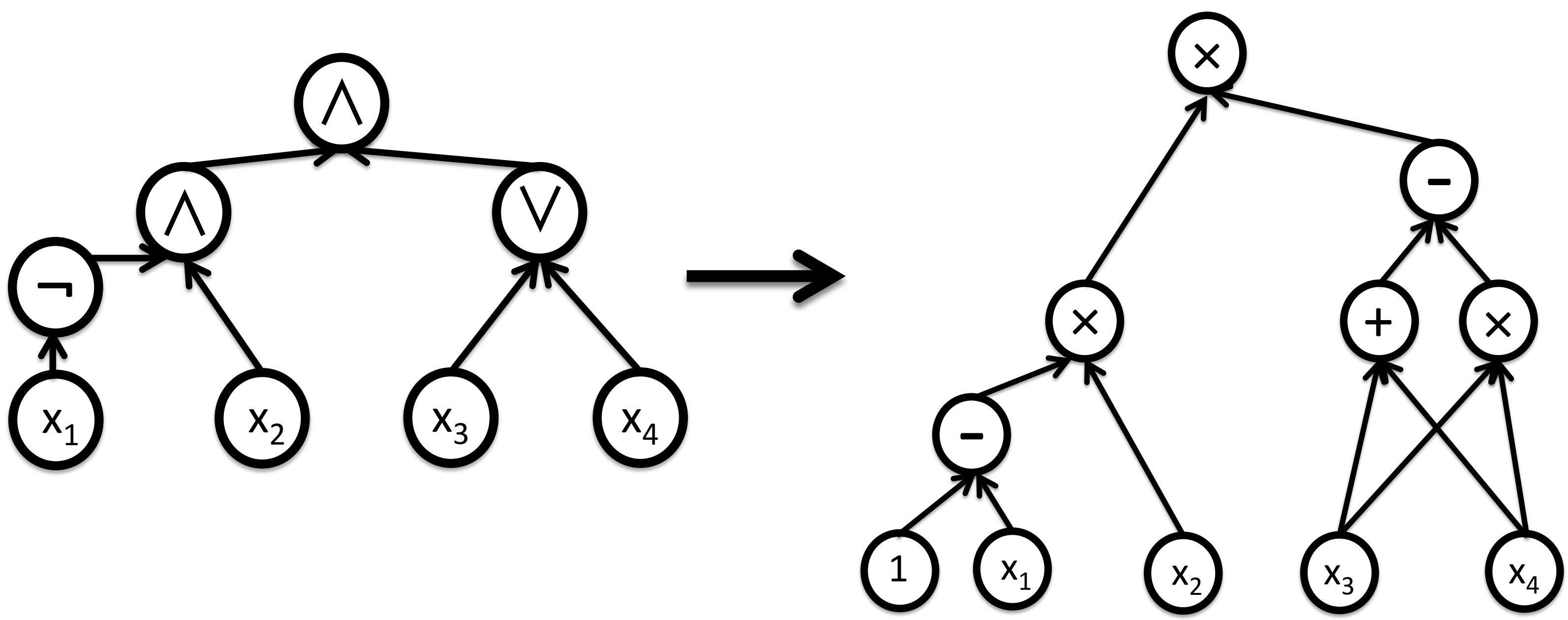


Arithmetization

- Key question: how to construct the extension polynomial g?
- Answer: Arithmetize φ
  - i.e., replace φ with an **arithmetic** circuit computing extension g
    - Go gate-by-gate through φ, replacing each gate with the gate's multilinear extension.
    - $NOT(x) \rightarrow 1 - x$
    - $AND(x, y) \rightarrow x \cdot y$
    - $OR(x, y) \rightarrow x + y - x \cdot y$



Summary of Arithmetization



Transforming a Boolean formula φ of size S into an arithmetic circuit computing an extension g of φ.

Note: $\deg(g) \leq S$, and g can be evaluated at any input, gate by gate, in time $O(S)$.

Costs of #SAT Protocol Applied to g

- Let φ be a Boolean formula of size S over n variables, g the extension obtained by arithmetizing φ.

Rounds	Communication	V Time	P Time
n	P sends a degree S polynomial in reach round, V sends one field element in each round $\Longrightarrow$ $O(S \cdot n)$ field elements sent in total.	• $O(S)$ time to process each of the n messages of P • $O(S)$ time to evaluate $g(r)$ $\Longrightarrow$ $O(S \cdot n)$ time total	P evaluates g at $O(S \cdot 2^n)$ points to determine each message $\Longrightarrow$ $O(S \cdot n \cdot 2^n)$ time in total.

IP=PSPACE

- #SAT is a **#P**-complete problem.
  - Hence, the protocol we just saw implies **every** problem in **#P** has an interactive proof with a polynomial time verifier.
- It is not much harder to show that this in fact holds for every problem in **PSPACE** [LFKN, Shamir].

IP=PSPACE

- #SAT is a **#P**-complete problem.
  - Hence, the protocol we just saw implies **every** problem in **#P** has an interactive proof with a polynomial time verifier.
- It is not much harder to show that this in fact holds for every problem in **PSPACE** [LFKN, Shamir].
- But is this a **practical** result?

IP=PSPACE

- #SAT is a **#P**-complete problem.
  - Hence, the protocol we just saw implies **every** problem in **#P** has an interactive proof with a polynomial time verifier.
- It is not much harder to show that this in fact holds for every problem in **PSPACE** [LFKN, Shamir].
- But is this a **practical** result?
  - No. The main reason: P's runtime.

IP=PSPACE

- #SAT is a **#P**-complete problem.
  - Hence, the protocol we just saw implies **every** problem in **#P** has an interactive proof with a polynomial time verifier.
- It is not much harder to show that this in fact holds for every problem in **PSPACE** [LFKN, Shamir].
- But is this a **practical** result?
  - No. The main reason: P's runtime.
  - When applying the protocols of [LFKN, Shamir] even to very simple problems, the honest prover would require **superpolynomial** time.

IP=PSPACE

- #SAT is a **#P**-complete problem.
  - Hence, the protocol we just saw implies **every** problem in **#P** has an interactive proof with a polynomial time verifier.
- It is not much harder to show that this in fact holds for every problem in **PSPACE** [LFKN, Shamir].
- But is this a **practical** result?
  - No. The main reason: P's runtime.
  - When applying the protocols of [LFKN, Shamir] even to very simple problems, the honest prover would require **superpolynomial** time.
  - The #SAT prover took time at least 2^n.
    - This seems unavoidable for #SAT, since we don't know how to even solve the problem in less than 2^n time.
    - But we can hope to solve "easier" problems without turning those problems into #SAT instances.

Doubly-Efficient Interactive Proofs

Doubly-Efficient Interactive Proof

- A doubly-efficient interactive proof for a problem is one where:
  - V runs in time linear in the input size.
  - P runs in polynomial time.

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: $\frac{1}{6} \cdot \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik}$.
- 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:

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

The GKR Protocol

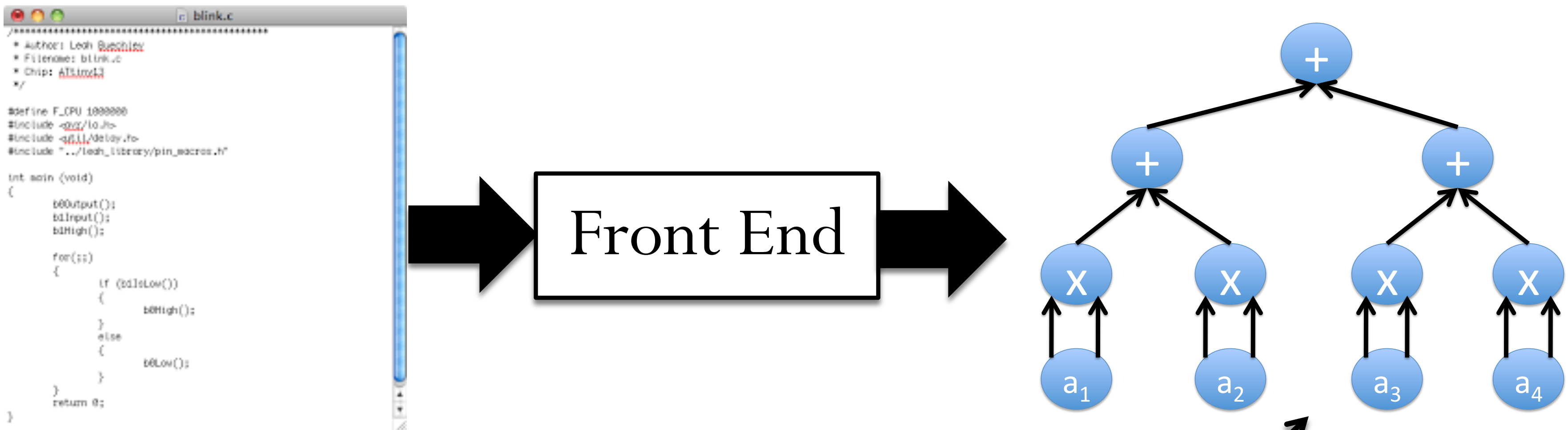
A General-Purpose Doubly-Efficient Interactive Proof

General-Purpose Doubly-Efficient Interactive Proofs

- [GKR 2008] gave a doubly-efficient interactive proof for any function computed by an efficient **parallel** algorithm.

General-Purpose Doubly-Efficient Protocols

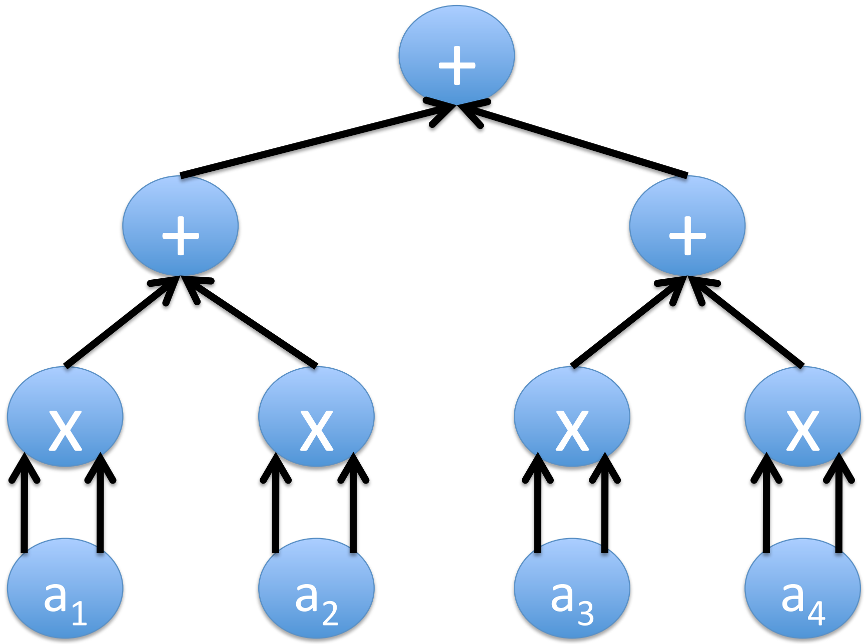
- Start with a computer program written in high-level programming language (C, Java, etc.)
- Step 1: Turn the program into an equivalent model amenable to probabilistic checking.
  - Typically some type of arithmetic circuit.
  - Called the **Front End** of the system.
- Step 2: Run an interactive proof or argument on the circuit.
  - Called the **Back End** of the system.



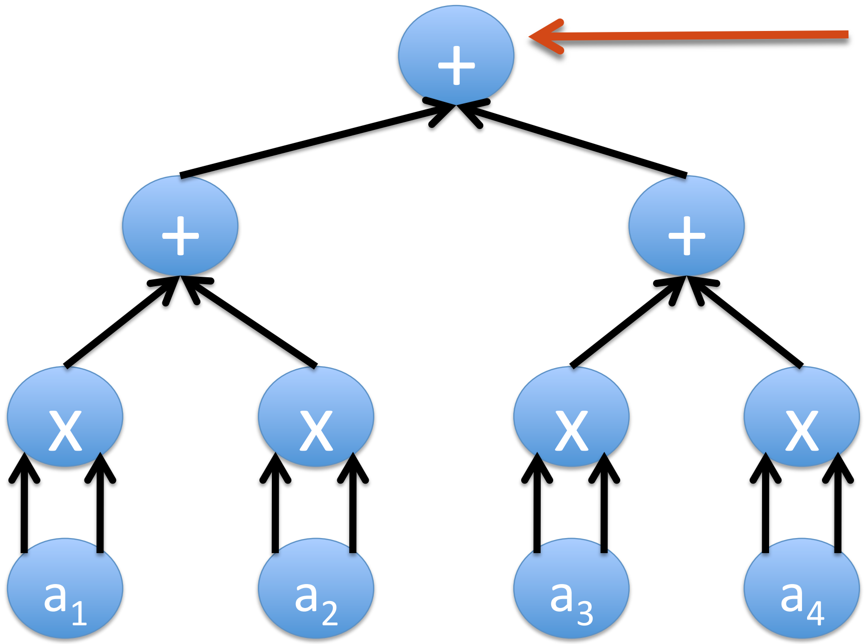
P and V run interactive proof (back end) on circuit.

Note: if the program is an efficient **parallel** algorithm, then the circuit can be small-depth.

The GKR Protocol: Overview

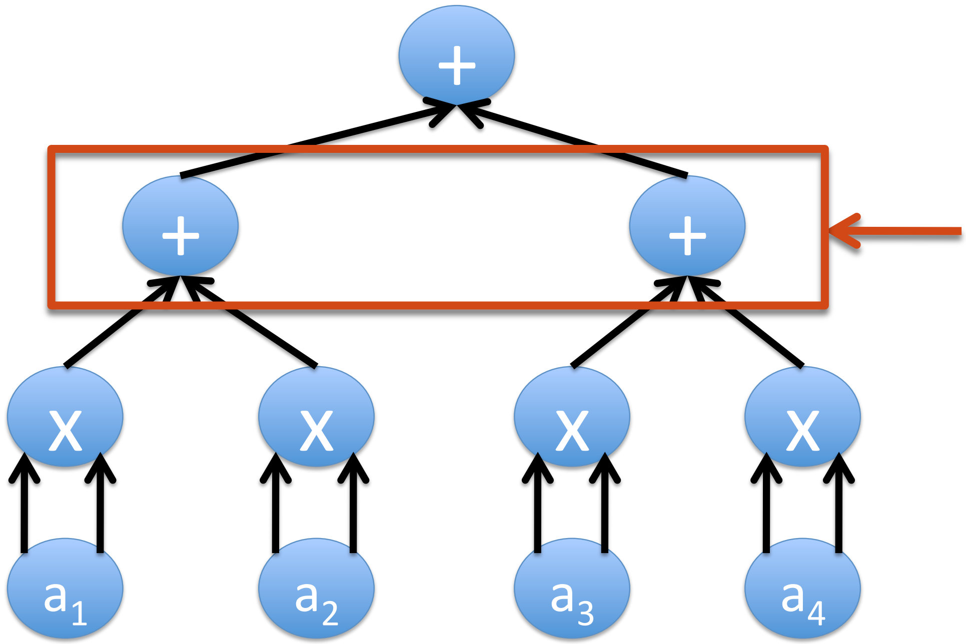


The GKR Protocol: Overview



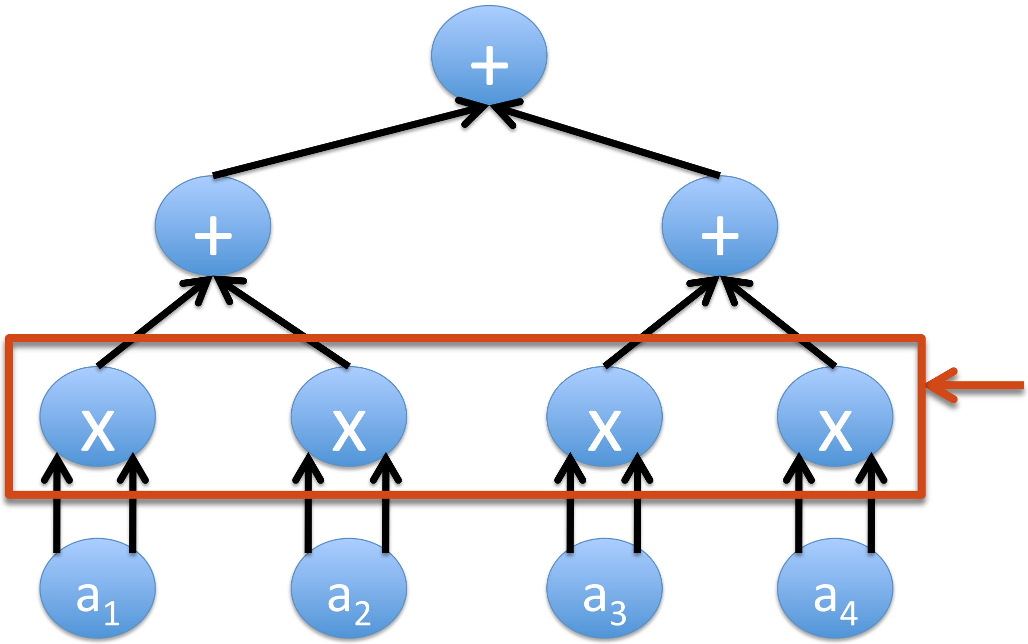
P starts the conversation with an answer (output).

The GKR Protocol: Overview



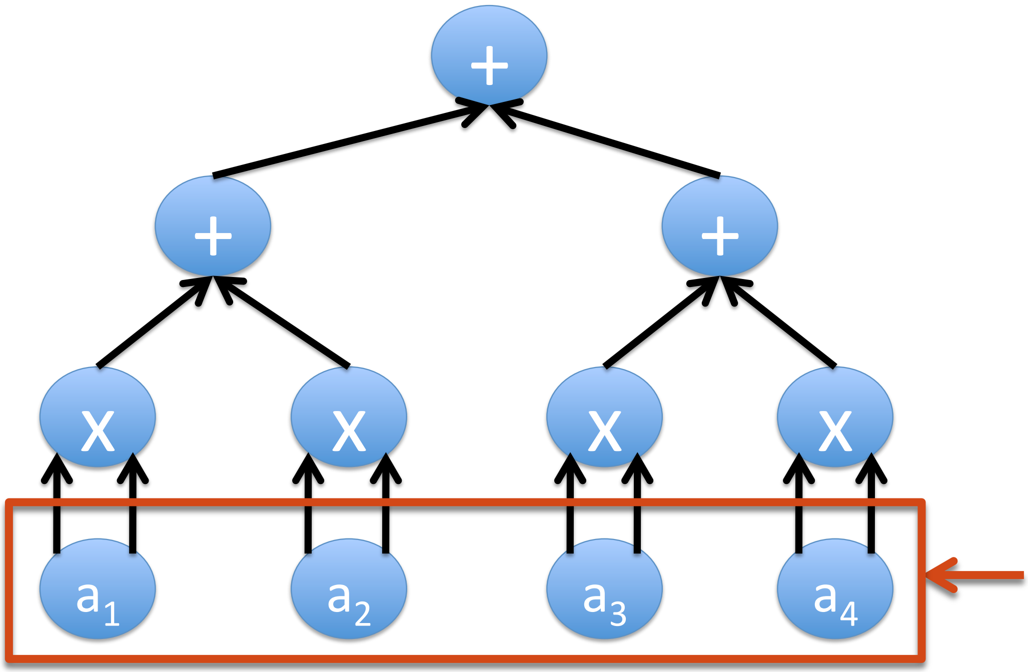
V sends series of challenges. P responds with info about next circuit level.

The GKR Protocol: Overview



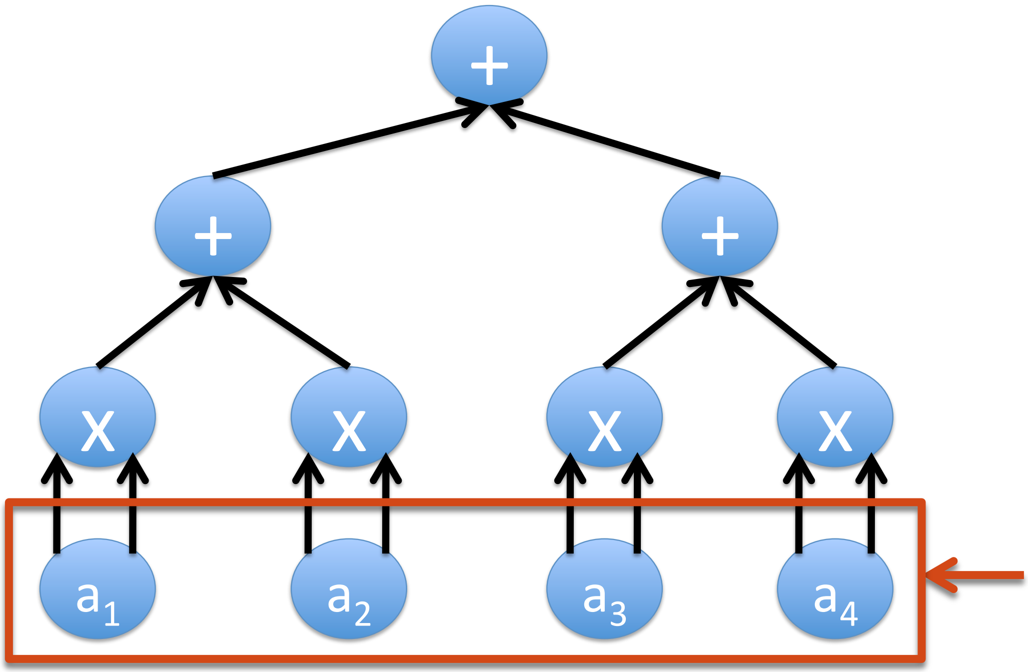
Challenges continue, layer by layer down to the the input.

The GKR Protocol: Overview



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

The GKR Protocol: Overview



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

V sees input directly, so can check P's final statement directly.

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.
- 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.
- Communication cost is $O(D \log S)$.

- P time is $O(S)$.
  - A naïve implementation of the prover in the GKR protocol with take $\Omega(S^4)$ time, where S is circuit size.
  - A sequence of works has brought this down to $O(S)$, for arbitrary circuits! [CMT12, Thaler13, WBSTWW17, WTSTW18, XZZPS19]

[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 linear 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 linear 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 $\tilde{O}(n + T^{\varepsilon} \cdot \text{poly}(s))$.
  - P runs in time $\tilde{O}\,(T^{1+\varepsilon} \cdot \text{poly}(s))$.
- In particular, if $T = \text{poly}(n)$ 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 $\tilde{O}(n + T^{\varepsilon} \cdot \text{poly}(s))$ 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 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: 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!