# Algebraic Complexity Theory

## Lecture 1: Course overview; Arithmetic Circuits

Department of Computer Science,
Indian Institute of Science

# About the course

- The broad area of Algorithms & Complexity has two complementary facets: designing efficient algorithms for computational problems (upper bounds) and proving hardness results by studying suitable models of computation (lower bounds).

# About the course

- The broad area of Algorithms & Complexity has two complementary facets: designing efficient algorithms for computational problems (upper bounds) and proving hardness results by studying suitable models of computation (lower bounds).

- When computational problems have algebraic, linear algebraic or number theoretic flavor, the two facets are known as Computer Algebra and Algebraic Complexity Theory (ACT).

# Computer Algebra

- Linear algebraic problems:

  1. Computing determinant of a matrix
  2. Computing inverse of a matrix
  3. Solving a system of linear equations
  4. Computing characteristic polynomial
  5. Matrix multiplication

# Computer Algebra

- Computation with polynomials:

1. Computing GCD of polynomials
2. Polynomial interpolation & (multi-point) evaluation
3. Polynomial factoring
4. Polynomial multiplication
5. Solving a polynomial system
6. Computing Gröbner basis of a polynomial ideal

# Computer Algebra

- Computation with numbers:

1. Computing GCD of integers
2. Integer factoring
3. Integer multiplication
4. Testing primality
5. Finding short vectors in an integer lattice

# Computer Algebra

- References:
  1. Modern Computer Algebra by von zur Gathen & Gerhard (1999, 2003)
  2. A Computational Introduction to Number Theory and Algebra by Victor Shoup (2005, 2008)
  3. Algebra and Computation by Madhu Sudan (1999)
  4. A Survey of Techniques used in Algebraic and Number Theoretic Algorithms by Manindra Agrawal (2005)
  5. Topics in Algebra and Computation by me (2013)

# Towards Algebraic Complexity Theory

- To get a sense of what **model of computation** we should study in ACT for examining algebraic problems & algorithms, let us focus on matrix multiplication.

- Linear algebraic problems:
  1. Computing determinant of a matrix
  2. Computing inverse of a matrix
  3. Solving a system of linear equations
  4. Computing characteristic polynomial
  5. Matrix multiplication

(1–4) Reduce to matrix multiplication

# Towards Algebraic Complexity Theory

- To get a sense of what **model of computation** we should study in ACT for examining algebraic problems & algorithms, let us focus on matrix multiplication.

- Linear algebraic problems:
    1. Computing determinant of a matrix
    2. Computing inverse of a matrix
    3. Solving a system of linear equations
    4. Computing characteristic polynomial
    5. Matrix multiplication

Ref.: A survey on "*Computation of the Inverse and Determinant of a Matrix*" by Villard (2002). See also the wiki page on "*Computational Complexity of Mathematical operations*".

# Towards Algebraic Complexity Theory

- To get a sense of what **model of computation** we should study in ACT for examining algebraic problems & algorithms, let us focus on matrix multiplication.

- Linear algebraic problems:
  1. Computing determinant of a matrix
  2. Computing inverse of a matrix
  3. Solving a system of linear equations
  4. Computing characteristic polynomial → Reference: *"Fast algorithms for the characteristic polynomial"* by Keller-Gehrig (1985)
  5. Matrix multiplication

# Matrix Multiplication

- Input: Two matrices $A = (x_{ij})_{i,j \in [n]}$ and $B = (y_{kl})_{k,l \in [n]}$.
- Output: The matrix $C = AB = (z_{pq})_{p,q \in [n]}$.

- Easy to see that $O(n^3)$ additions and multiplications are sufficient to compute $C$.
- Is $O(n^3)$ *arithmetic operations* necessary?

# Matrix Multiplication

- Input: Two matrices $A = (x_{ij})_{i,j \in [n]}$ and $B = (y_{kl})_{k,l \in [n]}$.
- Output: The matrix $C = AB = (z_{pq})_{p,q \in [n]}$.

- Easy to see that $O(n^3)$ additions and multiplications are sufficient to compute $C$.
- Is $O(n^3)$ *arithmetic operations* necessary? No! *(Strassen'69)*

- Let's focus on the $n = 2$ case. A trivial algorithm uses 8 multiplications and 4 additions.

# Matrix Multiplication

- Let $A = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $B = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$ and $C = AB = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$

- Strassen (1969). Using 7 multiplications and 18 additions/subtractions, we can compute $C$.

# Matrix Multiplication

- Let $A = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $B = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$ and $C = AB = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$

Suppose,

- $m_1 = x_{11} \cdot (y_{12} - y_{22})$
- $m_2 = (x_{11} + x_{12}) \cdot y_{22}$
- $m_3 = (x_{21} + x_{22}) \cdot y_{11}$
- $m_4 = x_{22} \cdot (y_{21} - y_{11})$
- $m_5 = (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$
- $m_6 = (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$
- $m_7 = (x_{11} - x_{21}) \cdot (y_{11} + y_{12})$

Then,

- $z_{11} = - m_2 + m_4 + m_5 + m_6$
- $z_{12} = m_1 + m_2$
- $z_{21} = m_3 + m_4$
- $z_{22} = m_1 - m_3 + m_5 - m_7$

- Strassen (1969). Using <u>7 multiplications</u> and 18 additions/subtractions, we can compute C.

# Matrix Multiplication

- Let $A = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $B = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$ and $C = AB = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$

Suppose,

- $m_1 = x_{11} \cdot (y_{12} - y_{22})$
- $m_2 = (x_{11} + x_{12}) \cdot y_{22}$
- $m_3 = (x_{21} + x_{22}) \cdot y_{11}$
- $m_4 = x_{22} \cdot (y_{21} - y_{11})$
- $m_5 = (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$
- $m_6 = (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$
- $m_7 = (x_{11} - x_{21}) \cdot (y_{11} + y_{12})$

Then,

- $z_{11} = - m_2 + m_4 + m_5 + m_6$
- $z_{12} = m_1 + m_2$
- $z_{21} = m_3 + m_4$
- $z_{22} = m_1 - m_3 + m_5 - m_7$

- Why does this help? Because, the above identities hold even if $x_{ij}$, $y_{kl}$ and $z_{pq}$ are matrices!

# Matrix Multiplication

- Let $A = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $B = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$ and $C = AB = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$

Suppose,

- $m_1 = x_{11} \cdot (y_{12} - y_{22})$
- $m_2 = (x_{11} + x_{12}) \cdot y_{22}$
- $m_3 = (x_{21} + x_{22}) \cdot y_{11}$
- $m_4 = x_{22} \cdot (y_{21} - y_{11})$
- $m_5 = (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$
- $m_6 = (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$
- $m_7 = (x_{11} - x_{21}) \cdot (y_{11} + y_{12})$

Then,

- $z_{11} = - m_2 + m_4 + m_5 + m_6$
- $z_{12} = m_1 + m_2$
- $z_{21} = m_3 + m_4$
- $z_{22} = m_1 - m_3 + m_5 - m_7$

- So we can apply recursion to multiply two $2^k \times 2^k$ matrices, where $x_{ij}$, $y_{kl}$ and $z_{pq}$ are $2^{k-1} \times 2^{k-1}$ matrices.

# Matrix Multiplication

- Let $A = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $B = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$ and $C = AB = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$

Suspose,

- $m_1 = x_{11} \cdot (y_{12} - y_{22})$
- $m_2 = (x_{11} + x_{12}) \cdot y_{22}$
- $m_3 = (x_{21} + x_{22}) \cdot y_{11}$
- $m_4 = x_{22} \cdot (y_{21} - y_{11})$
- $m_5 = (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$
- $m_6 = (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$
- $m_7 = (x_{11} - x_{21}) \cdot (y_{11} + y_{12})$

Then,

- $z_{11} = - m_2 + m_4 + m_5 + m_6$
- $z_{12} = m_1 + m_2$
- $z_{21} = m_3 + m_4$
- $z_{22} = m_1 - m_3 + m_5 - m_7$

- Solving the recursion $M(2^k) = 18 \cdot 2^{2(k-1)} + 7 \cdot M(2^{k-1})$, we get $M(2^k) = O(7^k)$. Hence, $M(n) = O(n^{\lg 7}) = O(n^{2.807..})$.

# Matrix Multiplication

- Open question: How many *arithmetic operations* (multiplications, divisions, additions, subtractions) are *necessary & sufficient* to multiply two n x n matrices?
- *Winograd (1971)* showed that 7 multiplications are required for multiplying two 2 x 2 matrices.

# Matrix Multiplication

- Open question: How many *arithmetic operations* (multiplications, divisions, additions, subtractions) are ***necessary & sufficient*** to multiply two n x n matrices?
- *Winograd (1971)* showed that 7 multiplications are required for multiplying two 2 x 2 matrices.

| References | M(n) |
|---|---|
| Strassen (1969) | $O(n^{2.81})$ |
| Schönhage (1981) | $O(n^{2.55})$ |
| Coppersmith & Winograd (1987) | $O(n^{2.376})$ |
| Optimized CW (Stothers, Vassilevska Williams, Le Gall..) | ⋮ |
| Alman, Duan, Vassilevska Williams, Xu, Xu, Zhou (2024) | $O(n^{2.37134})$ |

Ref. "*Algebraic Complexity Theory and Matrix Multiplication*" by Le Gall (a tutorial)

# Matrix Multiplication

- Open question: How many *arithmetic operations* (multiplications, divisions, additions, subtractions) are *necessary & sufficient* to multiply two n x n matrices?
- *Winograd (1971)* showed that 7 multiplications are required for multiplying two 2 x 2 matrices.

| **References** | **M(n)** |
|---|---|
| Strassen (1969) | $O(n^{2.81})$ |
| Schönhage (1981) | $O(n^{2.55})$ |
| Coppersmith & Winograd (1987) | $O(n^{2.376})$ |
| Optimized CW (Stothers, Vassilevska Williams, Le Gall..) | ⋮ |
| Alman, Duan, Vassilevska Williams, Xu, Xu, Zhou (2024) | $O(n^{2.37134})$ |

Ref. See also Lec 3 of "*Topics in Complexity Theory*" course (2015)

# Matrix Multiplication

- Open question: How many *arithmetic operations* (multiplications, divisions, additions, subtractions) are *necessary & sufficient* to multiply two n x n matrices?
- *Winograd (1971)* showed that 7 multiplications are required for multiplying two 2 x 2 matrices.

- Questions and results of the above kind are studied in Algebraic Complexity Theory using a model a computation known as ***arithmetic circuits***.

# Models of Computation in ACT

- Given the nature of the aforementioned problems, the basic operations involved in algorithms for these problems are arithmetic operations such as **addition** (+), **subtraction** (-), **multiplication** (x), **division** (÷), $k^{th}$ **root finding**, and **comparison**.

# Models of Computation in ACT

- Given the nature of the aforementioned problems, the basic operations involved in algorithms for these problems are arithmetic operations such as **addition** (+), **subtraction** (-), **multiplication** (x), **division** (÷), $k^{th}$ **root finding**, and **comparison**.
- Computation models, like TMs and Boolean Circuits, have been defined to analyze such algorithms, e.g.,

  a) real RAM model

Ref. "*Computational Geometry*" by Shamos (1978)

# Models of Computation in ACT

- Given the nature of the aforementioned problems, the basic operations involved in algorithms for these problems are arithmetic operations such as **addition** (+), **subtraction** (-), **multiplication** (x), **division** (÷), $k^{th}$ **root finding**, and **comparison**.
- Computation models, like TMs and Boolean Circuits, have been defined to analyze such algorithms, e.g.,

  a) real RAM model

  b) BSS model

Ref. *"On a theory of Computation and Complexity over the real numbers: NP-completeness, recursive functions, and Universal Machines"* by Blum, Shub, Smale (1989)

# Models of Computation in ACT

- Given the nature of the aforementioned problems, the basic operations involved in algorithms for these problems are arithmetic operations such as **addition** (+), **subtraction** (-), **multiplication** (x), **division** (÷), $k^{th}$ **root finding**, and **comparison**.
- Computation models, like TMs and Boolean Circuits, have been defined to analyze such algorithms, e.g.,

  a) real RAM model

  b) BSS model

  c) Arithmetic Circuits

↓

"Simplest" ***non-uniform*** version of the first two models

# Models of Computation in ACT

- Given the nature of the aforementioned problems, the basic operations involved in algorithms for these problems are arithmetic operations such as **addition** (+), **subtraction** (-), **multiplication** (x), **division** (÷), $k^{th}$ **root finding**, and **comparison**.
- Computation models, like TMs and Boolean Circuits, have been defined to analyze such algorithms, e.g.,

a) real RAM model

b) BSS model

c) Arithmetic Circuits

root finding and comparison **not** allowed

↑

"Simplest" *non-uniform* version of the first two models

# Arithmetic Circuits

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations ($+, \times, \div$) or input variables $x_1, \ldots, x_n$ or field constants.

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations ($+$,$\times$,$\div$) or input variables $x_1, \ldots, x_n$ or field constants.

  The *input nodes* labelled by $x_1, \ldots, x_n$ have in-degree $0$.

  Nodes labelled by $\mathbb{F}$ elements also have in-degree $0$.

  Nodes with out-degree $0$ are the *output nodes*.

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An ***arithmetic circuit*** over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations $(+,\times,\div)$ or input variables $x_1, \ldots, x_n$ or field constants.

  The ***input nodes*** labelled by $x_1, \ldots, x_n$ have in-degree $0$.

  Nodes labelled by $\mathbb{F}$ elements also have in-degree $0$.

  Nodes with out-degree $0$ are the ***output nodes***.

  Nodes labelled by $\div$ have fan-in two.

  Edges are labelled by $\mathbb{F}$ elements.

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations $(+,\times,\div)$ or input variables $x_1, \ldots, x_n$ or field constants.

A node labelled by $x_i$ (similarly, $\alpha\in\mathbb{F}$) *computes* $x_i$ (respectively, $\alpha$). A node labelled by an operation $*$ with inputs from nodes computing $f_1, \ldots, f_m$ *computes*

$$\alpha_1 f_1 * \ldots * \alpha_m f_m$$

where $\alpha_1, \ldots, \alpha_m \in \mathbb{F}$ are the corresponding edge labels.

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations $(+,\times,\div)$ or input variables $x_1, \ldots, x_n$ or field constants.

A node labelled by $x_i$ (similarly, $\alpha\in\mathbb{F}$) *computes* $x_i$ (respectively, $\alpha$). A node labelled by an operation $*$ with inputs from nodes computing $f_1, \ldots, f_m$ *computes*

$$\alpha_1 f_1 * \ldots * \alpha_m f_m$$

where $\alpha_1, \ldots, \alpha_m \in \mathbb{F}$ are the corresponding edge labels.

Division by $0$ is forbidden.

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations ($+, \times, \div$) or input variables $x_1, \ldots, x_n$ or field constants.

Naturally, an arithmetic circuit computes a set of *rational functions* over $\mathbb{F}$. A rational function is a ratio of two polynomials.

# Arithmetic Circuits

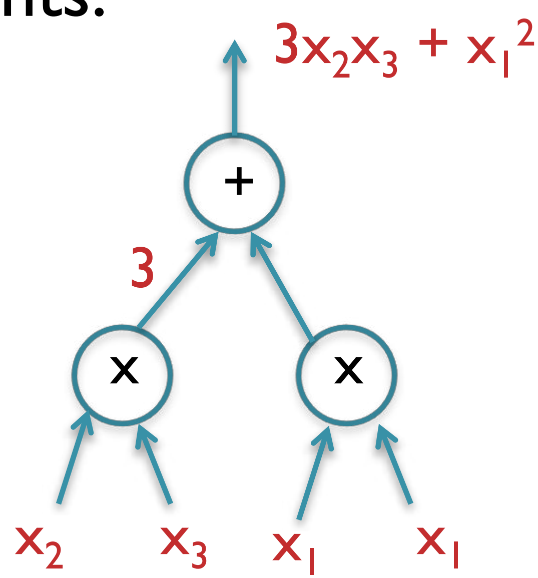
- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations $(+, \times, \div)$ or input variables $x_1, \ldots, x_n$ or field constants.

*Size* of a circuit is the number of edges in it.

*Depth* of a circuit is the length of the longest path from an input to an output node (gate).

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An ***arithmetic circuit*** over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations $(+,\times,\div)$ or input variables $x_1, \ldots, x_n$ or field constants.




Size = 7
Depth = 2

# Arithmetic Circuits

- Arith. circuits are *algebraic analogs* of Bool. circuits.
- Definition. An *arithmetic circuit* over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by arithmetic operations ($+, \times, \div$) or input variables $x_1, \ldots, x_n$ or field constants.

The number of $\times, \div$ gates with at least two children not labelled by field constants is called the *non-scalar complexity* of the circuit.

When there are no $\div$ gates, non-scalar complexity is also called *multiplicative complexity*.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 1. (Determinant computation)

Let $X = (x_{ij})_{i,j\in[n]}$ . Then,

$$\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)} ,$$

which is a degree-$n$ polynomial in $n^2$ variables with $\pm 1$ nonzero coefficients.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 2. (Matrix multiplication)

Let $A = (x_{ij})_{i,j \in [n]}$, $B = (y_{kl})_{k,l \in [n]}$, $C = AB = (z_{pq})_{p,q \in [n]}$.

Then, each $z_{pq} = \sum_{k \in [n]} x_{pk} \cdot y_{kq}$ is a quadratic form in the **x** and **y** variables. All nonzero coefficients are 1.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 3. (Solving a linear system)

$$\left.\begin{array}{c} x_{11}y_1 + x_{12}y_2 + \ldots + x_{1n}y_n = z_1 \\ \vdots \\ x_{n1}y_1 + x_{n2}y_2 + \ldots + x_{nn}y_n = z_n \end{array}\right\}$$

a linear system in **y** variables

Here the inputs are $\{x_{ij} : i,j \in [n]\}$ and $z_1, \ldots, z_n$.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 3. (Solving a linear system)

$$\left.\begin{array}{c} x_{11}y_1 + x_{12}y_2 + \ldots + x_{1n}y_n = z_1 \\ \vdots \\ x_{n1}y_1 + x_{n2}y_2 + \ldots + x_{nn}y_n = z_n \end{array}\right\}$$

a linear system in **y** variables

Let $X = (x_{ij})_{i,j\in[n]}$ and $Z_\ell$ be an $n \times n$ matrix whose $\ell^{th}$ column is $(z_1 \ldots z_n)^T$ and any other $(i,j)$-th entry is $x_{ij}$. Then, by Cramer's rule, $y_\ell = \det(Z_\ell)/\det(X)$.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 4. (Polynomial interpolation)

  Input. Points $(x_1,z_1), \ldots, (x_n, z_n) \in \mathbb{F}^2$.

  Output. $f(y) \in \mathbb{F}[y]$ s.t. $f(x_i) = z_i$ for all $i \in [n]$.

Lagrange interpolation.

$$f(y) = \sum_{i \in [n]} z_i \cdot \frac{\prod_{j \in [n]\setminus\{i\}} (y - x_j)}{\prod_{j \in [n]\setminus\{i\}} (x_i - x_j)}$$

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 4. (Polynomial interpolation)

  Input. Points $(x_1,z_1), \ldots, (x_n, z_n) \in \mathbb{F}^2$.

  Output. $f(y) \in \mathbb{F}[y]$ s.t. $f(x_i) = z_i$ for all $i \in [n]$.

Lagrange interpolation.

$$f(y) = \sum_{i \in [n]} z_i \cdot \frac{\prod_{j \in [n]\setminus\{i\}} (y - x_j)}{\prod_{j \in [n]\setminus\{i\}} (x_i - x_j)}$$

Obs. The coefficients of $f$ are rational functions in $\mathbf{x}$ and $\mathbf{z}$ variables of degree $O(n^2)$.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 4. (Polynomial interpolation)

  Lagrange interpolation.

$$f(y) = \sum_{i \in [n]} z_i \cdot \frac{\prod_{j \in [n]\setminus\{i\}} (y - x_j)}{\prod_{j \in [n]\setminus\{i\}} (x_i - x_j)}$$

Define the <u>*elementary symmetric polynomial*</u> as

$$\mathrm{ESym}_{n,d}(\mathbf{x}) := \sum_{S \in \binom{[n]}{d}} \prod_{i \in S} x_i$$

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 4. (Polynomial interpolation)

  Lagrange interpolation.

$$f(y) = \sum_{i \in [n]} z_i \cdot \frac{\prod_{j \in [n]\setminus\{i\}} (y - x_j)}{\prod_{j \in [n]\setminus\{i\}} (x_i - x_j)}$$

Obs. $(y - x_1)\cdot\ldots\cdot(y - x_n) = \sum_{d \in [0,n]} (-1)^d \cdot ESym_{n,d}(\mathbf{x}) \cdot y^{n-d}$

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 4. (Polynomial interpolation)

  Lagrange interpolation.

$$f(y) = \sum_{i \in [n]} z_i \cdot \frac{\prod_{j \in [n]\setminus\{i\}} (y - x_j)}{\prod_{j \in [n]\setminus\{i\}} (x_i - x_j)}$$

Obs. $(y - x_1)\cdot\ldots\cdot(y - x_n) = \sum_{d \in [0,n]} (-1)^d \cdot \mathrm{ESym}_{n,d}(\mathbf{x}) \cdot y^{n-d}$

More on ESym in later lectures…

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 5. (Polynomial multiplication)

  Input. $f = x_n y^n + \ldots + x_0$ and $g = z_n y^n + \ldots + z_0 \in \mathbb{F}[y]$

  Output. $h = f \cdot g = \sum_{i \in [0,2n]} \left( \sum_{j \in [0,i]} x_j z_{i-j} \right) y^i$

  Obs. The coefficients of $h$ are polynomials in **x** and **z** variables of degree 2.

# Why care about arithmetic circuits?

- Reason 1. For several of the aforementioned problems, the output is a rational function (often a *polynomial*) in the input variables.
- Example 6. (Checking coprimality of polynomials)

  Input. $f = x_n y^n + \ldots + x_0$ and $g = z_m y^m + \ldots + z_0 \in \mathbb{F}[y]$;

  the leading coefficients $x_n, z_m \neq 0$.

  Output. 1 if $\gcd_y(f, g) = 1$, else o/p 0.

Lemma. The $\gcd_y(f, g) \neq 1$ iff the *resultant* of f and g is 0.

(see Lec 6 of "Topics in Algebra & Computation" (2013))

# Sylvester Matrix and the Resultant

- $f = x_n y^n + \ldots + x_0$ , $g = z_m y^m + \ldots + z_0 \in \mathbb{F}[y]$; $x_n, z_m \neq 0$.
- Definition. The Sylvester matrix $S_y(f, g)$ is as follows:

$S_y(f, g)$ :=

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_n$ | 0 | | 0 | $z_m$ | 0 | | 0 |
| $x_{n-1}$ | $x_n$ | | $\vdots$ | $z_{m-1}$ | $z_m$ | | $\vdots$ |
| $\vdots$ | $x_{n-1}$ | | $\vdots$ | $\vdots$ | $z_{m-1}$ | | $\vdots$ |
| $x_0$ | $\vdots$ | | 0 | $z_0$ | $\vdots$ | | 0 |
| 0 | $x_0$ | $\ddots$ | $x_n$ | 0 | $z_0$ | $\ddots$ | $z_m$ |
| $\vdots$ | 0 | | $x_{n-1}$ | $\vdots$ | 0 | | $z_{m-1}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| 0 | 0 | | $x_0$ | 0 | 0 | | $z_0$ |

$(m+n) \times (m+n)$

m (first m columns), n (last n columns)

# Sylvester Matrix and the Resultant

- $f = x_n y^n + \ldots + x_0$ , $g = z_m y^m + \ldots + z_0 \in \mathbb{F}[y]$; $x_n, z_m \neq 0$.
- Definition. The Sylvester matrix $S_y(f, g)$ is as follows:

$S_y(f, g)$ :=

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_n$ | 0 | | 0 | $z_m$ | 0 | | 0 |
| $x_{n-1}$ | $x_n$ | | $\vdots$ | $z_{m-1}$ | $z_m$ | | $\vdots$ |
| $\vdots$ | $x_{n-1}$ | | $\vdots$ | $\vdots$ | $z_{m-1}$ | | $\vdots$ |
| $x_0$ | $\vdots$ | | 0 | $z_0$ | $\vdots$ | | 0 |
| 0 | $x_0$ | $\ddots$ | $x_n$ | 0 | $z_0$ | $\ddots$ | $z_m$ |
| $\vdots$ | 0 | | $x_{n-1}$ | $\vdots$ | 0 | | $z_{m-1}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| 0 | 0 | | $x_0$ | 0 | 0 | | $z_0$ |

m (first four columns) n (last four columns)

(m+n) x (m+n)

Defn. The *resultant*
$Res_y(f, g) := \det(S_y(f, g)) \in \mathbb{F}[\mathbf{x}, \mathbf{z}]$

$\deg(Res_y(f, g)) \leq m+n$

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. For every Boolean function $\mathscr{f}: \{0,1\}^n \longrightarrow \{0,1\}$, there is a *unique multilinear* polynomial $f \in \mathbb{F}[x_1,\ldots,x_n]$ s.t. $\mathscr{f}(\mathbf{a}) = f(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. For every Boolean function $\mathscr{f}: \{0,1\}^n \longrightarrow \{0,1\}$, there is a *unique multilinear* polynomial $f \in \mathbb{F}[x_1,\ldots,x_n]$ s.t. $\mathscr{f}(\mathbf{a}) = f(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- Proof sketch. Let $\mathscr{f} = \mathcal{T}_1 \vee \ldots \vee \mathcal{T}_m$ be a DNF representation of $\mathscr{f}$, where each term $\mathcal{T}_i$ has n literals. With every $\mathcal{T}_i$ associate a multilinear polynomial in a natural way; e.g., if $\mathcal{T}_i = x_1 \wedge \neg x_2 \wedge \neg x_3$, then $T_i = x_1(1-x_2)(1-x_3)$. Finally, $f = \sum_{i \in [n]} T_i$.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is <u>*necessary*</u> to prove arithmetic circuit lower bounds first!
- Obs. For every Boolean function $\mathscr{f}: \{0,1\}^n \longrightarrow \{0,1\}$, there is a *unique multilinear* polynomial $f \in \mathbb{F}[x_1,\ldots,x_n]$ s.t. $\mathscr{f}(\mathbf{a}) = f(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- Proof sketch. Uniqueness of $f$ can be shown using induction. It also follows from <u>*Combinatorial Nullstellensatz.*</u>

# Combinatorial Nullstellensatz

- Theorem *(Alon '99)*. Let $f \in \mathbb{F}[x_1,\ldots,x_n]$ and $\deg(f) = t_1+\ldots+t_n$, where $t_i \geq 0$. Suppose the coefficient of the monomial $x_1^{t_1}\cdot\ldots\cdot x_n^{t_n}$ in $f$ is nonzero.

  Then, if $S_1,\ldots, S_n$ are subsets of $\mathbb{F}$ with $|S_i| > t_i$, there is a point $\mathbf{a} \in S_1 \times \ldots \times S_n$ s.t. $f(\mathbf{a}) \neq 0$.

- That is, $S_1 \times \ldots \times S_n$ is a *hitting-set* for $f$. More on hitting-sets and Polynomial Identity Testing (PIT) later.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. If f is computable by a size-s arithmetic circuit over any *fixed* finite field, or over $\mathbb{Z}$ with poly(s) bit integers as edge labels, then $f$ is computable by a Boolean circuit of size poly(s).
- Assume that the arithmetic circuit has no ÷ gates.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. If $f$ is computable by a size-$s$ arithmetic circuit over any *fixed* finite field, or over $\mathbb{Z}$ with poly(s) bit integers as edge labels, then $f$ is computable by a Boolean circuit of size poly(s).
- Proof sketch. Over finite fields, replace every field operation by a constant sized Boolean circuit. Over $\mathbb{Z}$, reduce the integers labelling the edges modulo 2, replace a $+$ gate by a $\oplus$ gate, and a $\times$ gate by a $\wedge$ gate.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. If $f$ is computable by a size-$s$ arithmetic circuit over any *fixed* finite field, or over $\mathbb{Z}$ with poly(s) bit integers as edge labels, then $f$ is computable by a Boolean circuit of size poly(s).
- Corollary. A super-polynomial (i.e., $n^{\omega(1)}$) lower bound for Boolean circuits computing $f$ *implies* a super-polynomial lower bound for arithmetic circuits computing $f$.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is *necessary* to prove arithmetic circuit lower bounds first!
- Obs. If $f$ is computable by a size-$s$ arithmetic circuit over any *fixed* finite field, or over $\mathbb{Z}$ with poly(s) bit integers as edge labels, then $f$ is computable by a Boolean circuit of size poly(s).
- In this sense, proving arithmetic circuit lower bound is a stepping-stone to proving Boolean circuit lower bound.

# Why care about arithmetic circuits?

- Reason 2. Boolean circuit lower bounds imply arithmetic circuit lower bounds. So, it is <u>*necessary*</u> to prove arithmetic circuit lower bounds first!
- Open question. *(converse)* Does arithmetic circuit lower bound imply Boolean circuit lower bound?

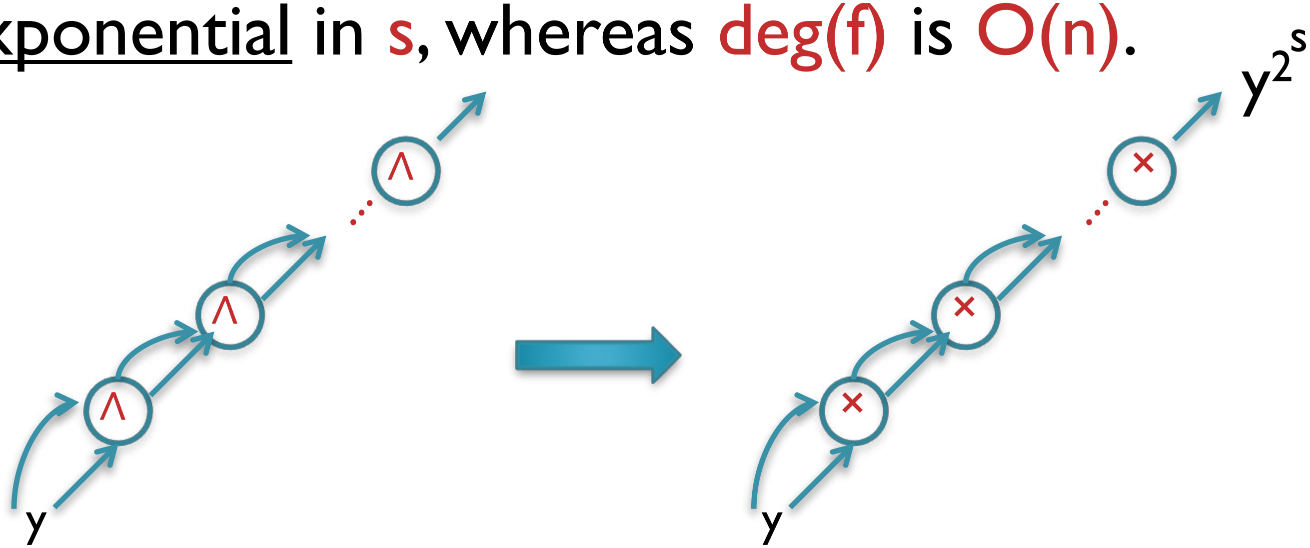
- We don't know!
- Caution. Do not interpret this as "*arithmetic circuits cannot simulate Boolean circuits.*"

# Arithmetization

- Obs. If $f$: $\{0,1\}^n \longrightarrow \{0,1\}$ is computable by a Boolean circuit of size $s$, then there's an arithmetic circuit (over *any* field) of size $O(s)$ computing a polynomial $h$ s.t. $f(\mathbf{a}) = h(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- *Proof sketch.* Replace $x_1 \wedge x_2$ by $x_1 x_2$, $x_1 \vee x_2$ by $1-(1-x_1)(1-x_2)$, and $\bar{x}_1$ by $1-x_1$.

# Arithmetization

- Obs. If $f$: $\{0,1\}^n \longrightarrow \{0,1\}$ is computable by a Boolean circuit of size s, then there's an arithmetic circuit (over *any* field) of size O(s) computing a polynomial h s.t. $f(\mathbf{a}) = h(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- Note. Polynomial h needn't be f (the unique multilinear polynomial for $f$). In particular, deg(h) can be <u>exponential</u> in s, whereas deg(f) is O(n).



# Arithmetization

- Obs. If $f$: $\{0,1\}^n \longrightarrow \{0,1\}$ is computable by a Boolean circuit of size $s$, then there's an arithmetic circuit (over *any* field) of size $O(s)$ computing a polynomial $h$ s.t. $f(\mathbf{a}) = h(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- Note. Polynomial $h$ needn't be $f$ (the unique multilinear polynomial for $f$). In particular, $\deg(h)$ can be exponential in $s$, whereas $\deg(f)$ is $O(n)$.
- The absence of poly-size circuits for $f$ doesn't necessarily rule out poly-size circuits for a polynomial $h$ satisfying $f(\mathbf{a}) = h(\mathbf{a}) = f(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.

# Arithmetization

- Obs. If $f: \{0,1\}^n \longrightarrow \{0,1\}$ is computable by a Boolean circuit of size $s$, then there's an arithmetic circuit (over *any* field) of size $O(s)$ computing a polynomial $h$ s.t. $f(\mathbf{a}) = h(\mathbf{a})$ for all $\mathbf{a} \in \{0,1\}^n$.
- Note. Polynomial $h$ needn't be $f$ (the unique multilinear polynomial for $f$). In particular, $\deg(h)$ can be exponential in $s$, whereas $\deg(f)$ is $O(n)$.
- So, we're unable to conclude that a super-poly lower bound for arithmetic circuits computing $f$ implies super-polynomial lower bound for Boolean circuits computing $f$.

# Why prove arithmetic circuit LB?

- As mentioned before, it is a necessary step for Boolean circuit lower bounds.

- Moreover, proving arithmetic circuit size upper and lower bounds is an important goal in its own right from the viewpoint of understanding the complexity of algebraic problems.

# Reading materials

- Matrix multiplication
  - *Fast Matrix Multiplication (survey)* by Bläser (2013)

- Matrix multiplication & models of computations
  - *Algebraic Complexity Theory (book)* by Bürgisser, Clausen, and Shokrollahi (1997)
  - *Algebraic Complexity Theory (survey)* by von zur Gathen (1988)
  - *Algebraic Complexity Theory (survey)* by Pippenger (1981)

# Reading materials

- Algebraic Complexity Classes
  - *Completeness and Reductions in Algebraic Complexity Theory (habilitation)* by Bürgisser (2000)
  - *Algebraic Complexity Classes* by Mahajan (2013)
  - *Completeness Classes in Algebraic Complexity Theory* by Bürgisser (2024)

# Reading materials

- Lower bounds and algorithms for arithmetic circuits
  - *Partial Derivatives in Arithmetic Complexity and beyond (survey)* by Chen, Kayal, and Wigderson (2010)
  - *Arithmetic Circuits: A survey of recent results and Open Questions* by Shpilka and Yehudayoff (2009)

# Reading materials

- Lower bounds for arithmetic circuits
  - *A survey of lower bounds in Arithmetic Circuit Complexity* by Saptharishi (and other contributors) (2021)

- Polynomial Identity Testing
  - *Progress on Polynomial Identity Testing: Part 1 and 2* by Saxena (2009, 2014)
  - *Recent advances in Polynomial Identity Testing* by Dutta and Ghosh (2024)