Primality Testing, Randomness & Derandomization

Nitin Saxena

Centrum voor Wiskunde en Informatica
Amsterdam

Bonn, May 2007

Outline

Primality: The Problem

- Given an integer n, test whether it is prime.
- Easy Solution: Divide n by all numbers between 2 and $(n-1)$.
- But we would like to do this in time polynomial in $\log n$.
- First asked by Kurt Gödel in a letter to John von Neumann in 1956.

Primality: The Problem

- Given an integer n, test whether it is prime.
- Easy Solution: Divide n by all numbers between 2 and $(n-1)$.
- But we would like to do this in time polynomial in $\log n$.
- First asked by Kurt Gödel in a letter to John von Neumann in 1956.

Primality: The Problem

- Given an integer n, test whether it is prime.
- Easy Solution: Divide n by all numbers between 2 and $(n-1)$.
- But we would like to do this in time polynomial in $\log n$.
- First asked by Kurt Gödel in a letter to John von Neumann in 1956.

Primality: The Problem

- Given an integer n, test whether it is prime.
- Easy Solution: Divide n by all numbers between 2 and $(n-1)$.
- But we would like to do this in time polynomial in $\log n$.
- First asked by Kurt Gödel in a letter to John von Neumann in 1956.

Primality: First Attempts

- Ancient Chinese (500 B.C.) and Greek (Eratosthenes, 200 B.C.) methods.
- Lucas proved in 1876 that $2^{127} - 1$ is prime; this would remain the largest known Mersenne prime for three-quarters of a century. Method generalizes to n with smooth $(n+1)$ (Lucas-Lehmer Test 1930s).
- Pépin's Test (1877): specialized for Fermat numbers $F_k = 2^{2^k} + 1$.
- Lucas Test (1891): When $(n-1)$ is smooth.
- Pocklington-Lehmer Test (1914): When we know distinct primes $p_1, \ldots, p_t | (n-1)$ such that $\prod_{i=1}^{t} p_t \geq \sqrt{n}$.

Primality: First Attempts

- Ancient Chinese (500 B.C.) and Greek (Eratosthenes, 200 B.C.) methods.
- Lucas proved in 1876 that $2^{127} - 1$ is prime; this would remain the largest known Mersenne prime for three-quarters of a century. Method generalizes to n with smooth $(n + 1)$ (Lucas-Lehmer Test 1930s).
- Pépin's Test (1877): specialized for Fermat numbers $F_k = 2^{2^k} + 1$.
- Lucas Test (1891): When $(n - 1)$ is smooth.
- Pocklington-Lehmer Test (1914): When we know distinct primes $p_1, \ldots, p_t | (n - 1)$ such that $\prod_{i=1}^{t} p_t \geq \sqrt{n}$.

Primality: First Attempts

- Ancient Chinese (500 B.C.) and Greek (Eratosthenes, 200 B.C.) methods.
- Lucas proved in 1876 that $2^{127} - 1$ is prime; this would remain the largest known Mersenne prime for three-quarters of a century. Method generalizes to n with smooth $(n+1)$ (Lucas-Lehmer Test 1930s).
- Pépin's Test (1877): specialized for Fermat numbers $F_k = 2^{2^k} + 1$.
- Lucas Test (1891): When $(n-1)$ is smooth.
- Pocklington-Lehmer Test (1914): When we know distinct primes $p_1, \ldots, p_t | (n-1)$ such that $\prod_{i=1}^{t} p_t \geq \sqrt{n}$.

Primality: First Attempts

- Ancient Chinese (500 B.C.) and Greek (Eratosthenes, 200 B.C.) methods.
- Lucas proved in 1876 that $2^{127} - 1$ is prime; this would remain the largest known Mersenne prime for three-quarters of a century. Method generalizes to n with smooth $(n+1)$ (Lucas-Lehmer Test 1930s).
- Pépin's Test (1877): specialized for Fermat numbers $F_k = 2^{2^k} + 1$.
- Lucas Test (1891): When $(n-1)$ is smooth.
- Pocklington-Lehmer Test (1914): When we know distinct primes $p_1, \ldots, p_t | (n-1)$ such that $\prod_{i=1}^{t} p_t \geq \sqrt{n}$.

Primality: First Attempts

- Ancient Chinese (500 B.C.) and Greek (Eratosthenes, 200 B.C.) methods.
- Lucas proved in 1876 that $2^{127}-1$ is prime; this would remain the largest known Mersenne prime for three-quarters of a century. Method generalizes to n with smooth $(n+1)$ (Lucas-Lehmer Test 1930s).
- Pépin's Test (1877): specialized for Fermat numbers $F_k = 2^{2^k}+1$.
- Lucas Test (1891): When $(n-1)$ is smooth.
- Pocklington-Lehmer Test (1914): When we know distinct primes $p_1, \ldots, p_t | (n-1)$ such that $\prod_{i=1}^{t} p_t \geq \sqrt{n}$.

Primality: Randomness Enters

Theorem (Solovay-Strassen, 1977)

An odd number n is prime iff for most $a \in \mathbb{Z}_n$, $a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$.

- Jacobi symbol $\left(\frac{a}{n}\right)$ is computable in time $O^{\sim}(\log^2 n)$.
- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{2}$.

Primality: Randomness Enters

Theorem (Solovay-Strassen, 1977)

An odd number n is prime iff for most $a \in \mathbb{Z}_n$, $a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$.

- Jacobi symbol $\left(\frac{a}{n}\right)$ is computable in time $O^{\sim}(\log^2 n)$.
- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{2}$.

Primality: Randomness Enters

Theorem (Solovay-Strassen, 1977)

An odd number n is prime iff for most $a \in \mathbb{Z}_n$, $a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$.

- Jacobi symbol $\left(\frac{a}{n}\right)$ is computable in time $O^{\sim}(\log^2 n)$.
- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{2}$.

Primality: Randomness Enters

Theorem (Solovay-Strassen, 1977)

An odd number n is prime iff for most $a \in \mathbb{Z}_n$, $a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$.

- Jacobi symbol $\left(\frac{a}{n}\right)$ is computable in time $O^{\sim}(\log^2 n)$.
- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{2}$.

Primality: Randomness Enters

Theorem (Solovay-Strassen, 1977)

An odd number n is prime iff for most $a \in \mathbb{Z}_n$, $a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$.

- Jacobi symbol $\left(\frac{a}{n}\right)$ is computable in time $O^{\sim}(\log^2 n)$.
- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{2}$.

Primality: Randomness Enters

Theorem (Miller-Rabin, 1980)

An odd number $n = 1 + 2^s \cdot t$ (odd t) is prime iff for most $a \in \mathbb{Z}_n$, the sequence $a^{2^{s-1}\cdot t}, a^{2^{s-2}\cdot t}, \ldots, a^t$ has either a -1 or all 1's.

- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{4}$.
- The most popular primality test!

Primality: Randomness Enters

Theorem (Miller-Rabin, 1980)

An odd number $n = 1 + 2^s \cdot t$ (odd t) is prime iff for most $a \in \mathbb{Z}_n$, the sequence $a^{2^{s-1}\cdot t}, a^{2^{s-2}\cdot t}, \ldots, a^t$ has either a -1 or all 1's.

- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{4}$.
- The most popular primality test!

Primality: Randomness Enters

Theorem (Miller-Rabin, 1980)

An odd number $n = 1 + 2^s \cdot t$ (odd t) is prime iff for most $a \in \mathbb{Z}_n$, the sequence $a^{2^{s-1}\cdot t}, a^{2^{s-2}\cdot t}, \ldots, a^t$ has either a -1 or all 1's.

- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{4}$.
- The most popular primality test!

Primality: Randomness Enters

Theorem (Miller-Rabin, 1980)

An odd number $n = 1 + 2^s \cdot t$ (odd t) is prime iff for most $a \in \mathbb{Z}_n$, the sequence $a^{2^{s-1}\cdot t}, a^{2^{s-2}\cdot t}, \ldots, a^t$ has either a -1 or all 1's.

- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{4}$.
- The most popular primality test!

Primality: Randomness Enters

Theorem (Miller-Rabin, 1980)

An odd number $n = 1 + 2^s \cdot t$ (odd t) is prime iff for most $a \in \mathbb{Z}_n$, the sequence $a^{2^{s-1} \cdot t}, a^{2^{s-2} \cdot t}, \ldots, a^t$ has either a -1 or all 1's.

- We check the above equation for a random a.
- This gives a randomized test that takes time $O^{\sim}(\log^2 n)$.
- It errs with probability at most $\frac{1}{4}$.
- The most popular primality test!

Outline

Randomized Algorithms

- These randomized primality tests began a vigorous study of randomness in computation.
- Deterministic Polynomial Time: P is the set of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $|x|^c$ many steps.
- Randomized Polynomial Time: BPP is the set of boolean functions f such that:

$$\exists g \in \mathrm{P},\ \exists d > 0,\ \forall x,\ \Pr_{r \in \{0,1\}^{|x|^d}}[g(x,r) = f(x)] \geq \frac{2}{3}$$

Randomized Algorithms

- These randomized primality tests began a vigorous study of randomness in computation.
- Deterministic Polynomial Time: P is the set of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $|x|^c$ many steps.
- Randomized Polynomial Time: BPP is the set of boolean functions f such that:

$$\exists g \in \mathrm{P},\ \exists d > 0,\ \forall x,\ \Pr_{r \in \{0,1\}^{|x|^d}}[g(x,r) = f(x)] \geq \frac{2}{3}$$

Randomized Algorithms

- These randomized primality tests began a vigorous study of randomness in computation.
- Deterministic Polynomial Time: P is the set of boolean functions $f:\{0,1\}^* \to \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $|x|^c$ many steps.
- Randomized Polynomial Time: BPP is the set of boolean functions f such that:

$$\exists g \in \mathrm{P},\ \exists d > 0,\ \forall x,\ \Pr_{r \in \{0,1\}^{|x|^d}}[g(x,r) = f(x)] \geq \frac{2}{3}$$

Randomized Algorithms: Examples

BPP contains some very well studied algebraic problems:

1. Primality testing.
2. Identity testing – given an arithmetic circuit $C(x_1, \ldots, x_n)$ computing a polynomial, test whether it is identically zero.
3. Polynomial factoring over finite fields – given a polynomial $f(x) \in \mathbb{F}_q[x]$, find a nontrivial factor.

Randomized Algorithms: Examples

BPP contains some very well studied algebraic problems:

1. Primality testing.
2. Identity testing – given an arithmetic circuit $C(x_1, \ldots, x_n)$ computing a polynomial, test whether it is identically zero.
3. Polynomial factoring over finite fields – given a polynomial $f(x) \in \mathbb{F}_q[x]$, find a nontrivial factor.

Randomized Algorithms: Examples

BPP contains some very well studied algebraic problems:

1. Primality testing.
2. Identity testing – given an arithmetic circuit $C(x_1, \ldots, x_n)$ computing a polynomial, test whether it is identically zero.
3. Polynomial factoring over finite fields – given a polynomial $f(x) \in \mathbb{F}_q[x]$, find a nontrivial factor.

Randomized Algorithms: Examples

BPP contains some very well studied algebraic problems:

1. Primality testing.
2. Identity testing – given an arithmetic circuit $C(x_1, \ldots, x_n)$ computing a polynomial, test whether it is identically zero.
3. Polynomial factoring over finite fields – given a polynomial $f(x) \in \mathbb{F}_q[x]$, find a nontrivial factor.

Determinism and Randomness: Different?

- Can we select the random bits carefully in a randomized algorithm such that there is no error?
- For example, if we assume GRH then the first $(2\log^2 n)$ a's suffice to test primality of n in Solovay-Strassen and Miller-Rabin tests.
- Can we derandomize any randomized polynomial time algorithm?
- Is BPP=P? or

"God does not play dice...."??

Determinism and Randomness: Different?

- Can we select the random bits carefully in a randomized algorithm such that there is no error?
- For example, if we assume GRH then the first $(2\log^2 n)$ a's suffice to test primality of n in Solovay-Strassen and Miller-Rabin tests.
- Can we derandomize any randomized polynomial time algorithm?
- Is BPP=P? or

"God does not play dice...."??

Determinism and Randomness: Different?

- Can we select the random bits carefully in a randomized algorithm such that there is no error?
- For example, if we assume GRH then the first $(2\log^2 n)$ a's suffice to test primality of n in Solovay-Strassen and Miller-Rabin tests.
- Can we derandomize any randomized polynomial time algorithm?
- Is BPP=P? or

"God does not play dice...."??

Determinism and Randomness: Different?

- Can we select the random bits carefully in a randomized algorithm such that there is no error?
- For example, if we assume GRH then the first $(2\log^2 n)$ a's suffice to test primality of n in Solovay-Strassen and Miller-Rabin tests.
- Can we derandomize any randomized polynomial time algorithm?
- Is BPP=P? or

"God does not play dice...."??

Determinism and Randomness: Different?

- Can we select the random bits carefully in a randomized algorithm such that there is no error?
- For example, if we assume GRH then the first $(2\log^2 n)$ a's suffice to test primality of n in Solovay-Strassen and Miller-Rabin tests.
- Can we derandomize any randomized polynomial time algorithm?
- Is BPP=P? or

"God does not play dice...."??

Determinism and Randomness: Hardness Enters

- In the 1990s it was observed that if there are hard problems then they can be used to derandomize.
- Specifically, Impagliazzo-Wigderson showed in 1997 that BPP=P if E requires exponential boolean circuits.
- E is the set of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $2^{c|x|}$ many steps.
- A boolean circuit comprises of AND, OR, NOT gates.

Determinism and Randomness: Hardness Enters

- In the 1990s it was observed that if there are hard problems then they can be used to derandomize.
- Specifically, Impagliazzo-Wigderson showed in 1997 that BPP=P if E requires exponential boolean circuits.
- E is the set of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $2^{c|x|}$ many steps.
- A boolean circuit comprises of AND, OR, NOT gates.

Determinism and Randomness: Hardness Enters

- In the 1990s it was observed that if there are hard problems then they can be used to derandomize.
- Specifically, Impagliazzo-Wigderson showed in 1997 that BPP=P if E requires exponential boolean circuits.
- E is the set of boolean functions $f : \{0,1\}^* \to \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $2^{c|x|}$ many steps.
- A boolean circuit comprises of AND, OR, NOT gates.

Determinism and Randomness: Hardness Enters

- In the 1990s it was observed that if there are hard problems then they can be used to derandomize.
- Specifically, Impagliazzo-Wigderson showed in 1997 that BPP=P if E requires exponential boolean circuits.
- E is the set of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ such that $f(x)$ is computable by a Turing machine in $2^{c|x|}$ many steps.
- A boolean circuit comprises of AND, OR, NOT gates.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose $f \in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose $f \in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose $f \in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose f $\in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose f $\in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

- *Proof Idea:* Suppose f $\in$BPP and g is the corresponding boolean function computable in n^c time and using $m := n^d$ random bits.
- Suppose we have a hard boolean function $h \in$ E that cannot be computed in randomized polynomial time.
- Then instead of "feeding" g purely random m bits we can feed $h(y^{(1)}), \ldots, h(y^{(m)})$. Where $y^{(i)}$'s are substrings of a string $y \in \{0,1\}^{c \log n}$.
- We intend to show that these m bits would be random enough since h is hard.
- Thus, computing $g(x, h(y^{(1)}) \cdots h(y^{(m)}))$ for all $y \in \{0,1\}^{c \log n}$ and taking the majority vote would give us a deterministic polynomial time way to compute f.

Determinism and Randomness: Hardness Enters

Proof that $h(y^{(1)}), \ldots, h(y^{(m)})$ *are pseudorandom:*

- Suppose $h(y^{(1)}), \ldots, h(y^{(m)})$ are not behaving randomly. Say:
  $$\left| \Pr_y \left[g(x, h(y^{(1)}) \cdots h(y^{(m)})) = f(x)\right] - \Pr_{r_1,\ldots,r_m} \left[g(x, r_1 \cdots r_m) = f(x)\right] \right| > \frac{1}{n}$$
  $$\Rightarrow \left| \sum_{i=0}^{m} \left(\Pr_{y,r_i,\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i-1)}) \cdot r_i \cdots r_m) = f(x)\right] - \Pr_{y,r_{i+1},\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i)}) \cdot r_{i+1} \cdots r_m) = f(x)\right]\right) \right| > \frac{1}{n}$$
- Therefore, $\exists j$ such that the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j-1)}) \cdot r_j \cdots r_m) = f(x)$ differs from the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j)}) \cdot r_{j+1} \cdots r_m) = f(x)$ by more than $\frac{1}{nm}$.
- Thus, g and f can be used to predict the value of $h(y^{(j)})$ which contradicts the hardness of h.

Determinism and Randomness: Hardness Enters

Proof that $h(y^{(1)}), \ldots, h(y^{(m)})$ are pseudorandom:

- Suppose $h(y^{(1)}), \ldots, h(y^{(m)})$ are not behaving randomly. Say:
  $| \Pr_y \left[g(x, h(y^{(1)}) \cdots h(y^{(m)})) = f(x)\right] - \Pr_{r_1,\ldots,r_m}\left[g(x, r_1 \cdots r_m) = f(x)\right] \ | > \frac{1}{n}$
  $\Rightarrow | \sum_{i=0}^{m} \left(\Pr_{y,r_i,\ldots,r_m}\left[g(x, h(y^{(1)}) \cdots h(y^{(i-1)}) \cdot r_i \cdots r_m) = f(x)\right] - \Pr_{y,r_{i+1},\ldots,r_m}\left[g(x, h(y^{(1)}) \cdots h(y^{(i)}) \cdot r_{i+1} \cdots r_m) = f(x)\right]\right) \ | > \frac{1}{n}$
- Therefore, $\exists j$ such that the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j-1)}) \cdot r_j \cdots r_m) = f(x)$ differs from the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j)}) \cdot r_{j+1} \cdots r_m) = f(x)$ by more than $\frac{1}{nm}$.
- Thus, g and f can be used to predict the value of $h(y^{(j)})$ which contradicts the hardness of h.

Determinism and Randomness: Hardness Enters

Proof that $h(y^{(1)}), \ldots, h(y^{(m)})$ *are pseudorandom:*

- Suppose $h(y^{(1)}), \ldots, h(y^{(m)})$ are not behaving randomly. Say:

  $$| \Pr_y \left[g(x, h(y^{(1)}) \cdots h(y^{(m)})) = f(x)\right] - \Pr_{r_1,\ldots,r_m} \left[g(x, r_1 \cdots r_m) = f(x)\right] | > \frac{1}{n}$$

  $$\Rightarrow | \sum_{i=0}^{m} \left(\Pr_{y,r_i,\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i-1)}) \cdot r_i \cdots r_m) = f(x)\right] - \Pr_{y,r_{i+1},\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i)}) \cdot r_{i+1} \cdots r_m) = f(x)\right]\right) | > \frac{1}{n}$$

- Therefore, $\exists j$ such that the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j-1)}) \cdot r_j \cdots r_m) = f(x)$ differs from the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j)}) \cdot r_{j+1} \cdots r_m) = f(x)$ by more than $\frac{1}{nm}$.
- Thus, g and f can be used to predict the value of $h(y^{(j)})$ which contradicts the hardness of h.

Determinism and Randomness: Hardness Enters

Proof that $h(y^{(1)}), \ldots, h(y^{(m)})$ *are pseudorandom:*

- Suppose $h(y^{(1)}), \ldots, h(y^{(m)})$ are not behaving randomly. Say:

  $$\left| \Pr_y \left[g(x, h(y^{(1)}) \cdots h(y^{(m)})) = f(x) \right] - \Pr_{r_1,\ldots,r_m} \left[g(x, r_1 \cdots r_m) = f(x) \right] \right| > \frac{1}{n}$$

  $$\Rightarrow \left| \sum_{i=0}^{m} \left(\Pr_{y,r_i,\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i-1)}) \cdot r_i \cdots r_m) = f(x) \right] - \Pr_{y,r_{i+1},\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i)}) \cdot r_{i+1} \cdots r_m) = f(x) \right] \right) \right| > \frac{1}{n}$$

- Therefore, $\exists j$ such that the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j-1)}) \cdot r_j \cdots r_m) = f(x)$ differs from the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j)}) \cdot r_{j+1} \cdots r_m) = f(x)$ by more than $\frac{1}{nm}$.

- Thus, g and f can be used to predict the value of $h(y^{(j)})$ which contradicts the hardness of h.

Determinism and Randomness: Hardness Enters

Proof that $h(y^{(1)}), \ldots, h(y^{(m)})$ are pseudorandom:

- Suppose $h(y^{(1)}), \ldots, h(y^{(m)})$ are not behaving randomly. Say:
  $$| \Pr_y \left[g(x, h(y^{(1)}) \cdots h(y^{(m)})) = f(x)\right] - \Pr_{r_1,\ldots,r_m} \left[g(x, r_1 \cdots r_m) = f(x)\right] | > \frac{1}{n}$$
  $$\Rightarrow | \sum_{i=0}^{m} \left(\Pr_{y,r_i,\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i-1)}) \cdot r_i \cdots r_m) = f(x)\right] - \Pr_{y,r_{i+1},\ldots,r_m} \left[g(x, h(y^{(1)}) \cdots h(y^{(i)}) \cdot r_{i+1} \cdots r_m) = f(x)\right]\right) | > \frac{1}{n}$$
- Therefore, $\exists j$ such that the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j-1)}) \cdot r_j \cdots r_m) = f(x)$ differs from the prob of $g(x, h(y^{(1)}) \cdots h(y^{(j)}) \cdot r_{j+1} \cdots r_m) = f(x)$ by more than $\frac{1}{nm}$.
- Thus, g and f can be used to predict the value of $h(y^{(j)})$ which contradicts the hardness of h.

Derandomization

- We saw heuristic evidence that randomized polynomial time algorithms can always be made deterministic polynomial time.
- But such a general derandomization seems tied up with proving lower bounds.
- How about derandomizing concrete algebraic problems?

Derandomization

- We saw heuristic evidence that randomized polynomial time algorithms can always be made deterministic polynomial time.
- But such a general derandomization seems tied up with proving lower bounds.
- How about derandomizing concrete algebraic problems?

Derandomization

- We saw heuristic evidence that randomized polynomial time algorithms can always be made deterministic polynomial time.
- But such a general derandomization seems tied up with proving lower bounds.
- How about derandomizing concrete algebraic problems?

Outline

1. Prologue
2. Randomness
3. Derandomizing Primality Testing
4. Derandomizing Identity Testing
   - Depth 3 Circuits: Algorithm I
   - Depth 3 Circuits: Algorithm II
5. Epilogue

Agrawal-Kayal-S (AKS) Test

Theorem (A Generalization of FLT)

If n is a prime then for all $a \in \mathbb{Z}_n$, $(x+a)^n = (x^n + a) \pmod{n, x^r - 1}$.

- This was the basis of the AKS test proposed in 2002.
- It was the first unconditional, deterministic and polynomial time primality test.

Agrawal-Kayal-S (AKS) Test

Theorem (A Generalization of FLT)

If n is a prime then for all $a \in \mathbb{Z}_n$, $(x + a)^n = (x^n + a)$ $(mod\ n, x^r - 1)$.

- This was the basis of the AKS test proposed in 2002.
- It was the first unconditional, deterministic and polynomial time primality test.

Agrawal-Kayal-S (AKS) Test

Theorem (A Generalization of FLT)

If n is a prime then for all $a \in \mathbb{Z}_n$, $(x+a)^n = (x^n + a) \pmod{n, x^r - 1}$.

- This was the basis of the AKS test proposed in 2002.
- It was the first unconditional, deterministic and polynomial time primality test.

AKS Test

1. If n is a prime power, it is composite.
2. Select an r such that $\mathrm{ord}_r(n) > 4\log^2 n$ and work in the ring $R := \mathbb{Z}_n[x]/(x^r - 1)$.
3. For each a, $1 \leq a \leq \ell := \lceil 2\sqrt{r}\log n \rceil$, check if $(x+a)^n = (x^n + a)$.
4. If yes then n is prime else composite.

AKS Test

1. If n is a prime power, it is composite.
2. Select an r such that $\text{ord}_r(n) > 4\log^2 n$ and work in the ring $R := \mathbb{Z}_n[x]/(x^r - 1)$.
3. For each a, $1 \le a \le \ell := \lceil 2\sqrt{r}\log n \rceil$, check if $(x+a)^n = (x^n + a)$.
4. If yes then n is prime else composite.

AKS Test

1. If n is a prime power, it is composite.
2. Select an r such that $\text{ord}_r(n) > 4\log^2 n$ and work in the ring $R := \mathbb{Z}_n[x]/(x^r - 1)$.
3. For each a, $1 \leq a \leq \ell := \lceil 2\sqrt{r}\log n \rceil$, check if $(x+a)^n = (x^n + a)$.
4. If yes then n is prime else composite.

AKS Test

1. If n is a prime power, it is composite.
2. Select an r such that $\mathrm{ord}_r(n) > 4\log^2 n$ and work in the ring $R := \mathbb{Z}_n[x]/(x^r - 1)$.
3. For each a, $1 \leq a \leq \ell := \lceil 2\sqrt{r}\log n \rceil$, check if $(x+a)^n = (x^n + a)$.
4. If yes then n is prime else composite.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r} \rangle$. $t := \#I \geq \text{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r} \rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.

  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r} \rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.

  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r} \rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod r\rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)}\rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r}\rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)}\rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod{r} \rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

- Suppose all the congruences hold and p is a prime factor of n.
- The group $I := \langle n, p \pmod r \rangle$. $t := \#I \geq \mathrm{ord}_r(n) \geq 4\log^2 n$.
- The group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ where $h(x)$ is an irreducible factor of $\frac{x^r-1}{x-1}$ modulo p.
  $\#J \geq 2^{\min\{t,\ell\}} > 2^{2\sqrt{t}\log n} \geq n^{2\sqrt{t}}$.
- *Proof:* Let $f(x), g(x)$ be two different products of $(x+a)$'s, having degree $< t$. Suppose $f(x) = g(x) \pmod{p, h(x)}$.
- The test tells us that $f(x^{n^i \cdot p^j}) = g(x^{n^i \cdot p^j}) \pmod{p, h(x)}$.
- But this means that $f(z) - g(z)$ has atleast t roots in the field $\mathbb{F}_p[x]/(h(x))$, which is a contradiction.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.
Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i,j,i',j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.
Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i,j,i',j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod r \rangle$ is of size $t > 4 \log^2 n$.
Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i, j) \neq (i', j')$ such that $0 \leq i, j, i', j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod r$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.

Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i, j, i', j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.

Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i, j, i', j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.
Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i,j,i',j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: The Proof

The Two Groups

Group $I := \langle n, p \pmod{r} \rangle$ is of size $t > 4\log^2 n$.
Group $J := \langle (x+1), \ldots, (x+\ell) \pmod{p, h(x)} \rangle$ is of size $> n^{2\sqrt{t}}$.

- There exist tuples $(i,j) \neq (i',j')$ such that $0 \leq i, j, i', j' \leq \sqrt{t}$ and $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{r}$.
- The test tells us that for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x^{n^i \cdot p^j})$ and $f(x)^{n^{i'} \cdot p^{j'}} = f(x^{n^{i'} \cdot p^{j'}})$.
- Thus, for all $f(x) \in J$, $f(x)^{n^i \cdot p^j} = f(x)^{n^{i'} \cdot p^{j'}}$.
- As J is a cyclic group: $n^i \cdot p^j \equiv n^{i'} \cdot p^{j'} \pmod{\#J}$.
- As $\#J$ is large, $n^i \cdot p^j = n^{i'} \cdot p^{j'}$. Hence, $n = p$ a prime.

AKS Test: Time Complexity

- Recall that r is the least number such that $\mathrm{ord}_r(n) > 4\log^2 n$.
- Prime number theorem gives $r = O(\log^5 n)$ and the algorithm takes time $O^{\sim}(\log^{10.5} n)$.
- Lenstra and Pomerance (2003) further reduced the time complexity to $O^{\sim}(\log^6 n)$.

AKS Test: Time Complexity

- Recall that r is the least number such that $\text{ord}_r(n) > 4\log^2 n$.
- Prime number theorem gives $r = O(\log^5 n)$ and the algorithm takes time $O^{\sim}(\log^{10.5} n)$.
- Lenstra and Pomerance (2003) further reduced the time complexity to $O^{\sim}(\log^6 n)$.

AKS Test: Time Complexity

- Recall that r is the least number such that $\text{ord}_r(n) > 4\log^2 n$.
- Prime number theorem gives $r = O(\log^5 n)$ and the algorithm takes time $O^{\sim}(\log^{10.5} n)$.
- Lenstra and Pomerance (2003) further reduced the time complexity to $O^{\sim}(\log^6 n)$.

Outline

Identities

- High School algebra teaches us lots of useful algebraic identities.
- For example,
  $x^3 + y^3 + z^3 - 3xyz = (x + y + z)(x^2 + y^2 + z^2 - xy - yz - zx)$.
- Lebesgue identity:

$$(a^2 + b^2 + c^2 + d^2)^2 = (a^2 + b^2 - c^2 - d^2)^2 + (2ac + 2bd)^2 + (2ad - 2bc)^2$$

Identities

- High School algebra teaches us lots of useful algebraic identities.
- For example,
  $x^3 + y^3 + z^3 - 3xyz = (x + y + z)(x^2 + y^2 + z^2 - xy - yz - zx)$.
- Lebesgue identity:

$$(a^2 + b^2 + c^2 + d^2)^2 = (a^2 + b^2 - c^2 - d^2)^2 + (2ac + 2bd)^2 + (2ad - 2bc)^2$$

Identities

- High School algebra teaches us lots of useful algebraic identities.
- For example,
  $x^3 + y^3 + z^3 - 3xyz = (x+y+z)(x^2+y^2+z^2-xy-yz-zx)$.
- Lebesgue identity:

$$(a^2+b^2+c^2+d^2)^2 = (a^2+b^2-c^2-d^2)^2 + (2ac+2bd)^2 + (2ad-2bc)^2$$

Identities

- Identity communicated by Euler in a letter to Goldbach on April 15, 1750:

$$
\begin{aligned}
&(a_1^2+a_2^2+a_3^2+a_4^2)(b_1^2+b_2^2+b_3^2+b_4^2) \;=\\
&\qquad (a_1b_1-a_2b_2-a_3b_3-a_4b_4)^2 \;+\; (a_1b_2+a_2b_1+a_3b_4-a_4b_3)^2 \;+\\
&\qquad (a_1b_3-a_2b_4+a_3b_1+a_4b_2)^2 \;+\; (a_1b_4+a_2b_3-a_3b_2+a_4b_1)^2
\end{aligned}
$$

- All these can be checked by expansion.

Identities

- Identity communicated by Euler in a letter to Goldbach on April 15, 1750:

$$(a_1^2 + a_2^2 + a_3^2 + a_4^2)(b_1^2 + b_2^2 + b_3^2 + b_4^2) = (a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4)^2 + (a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3)^2 + (a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2)^2 + (a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1)^2$$

- All these can be checked by expansion.

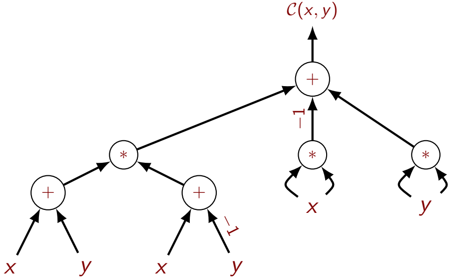
Formalizing Identity Testing

- We can assume that our polynomial expression is given in the form of an arithmetic circuit $\mathcal{C}$ over a field $\mathbb{F}$:

- Identity testing is the problem of checking whether a given circuit is zero or not.

Formalizing Identity Testing

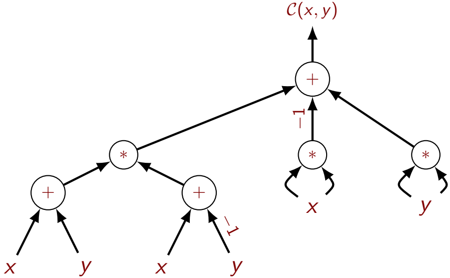
- We can assume that our polynomial expression is given in the form of an arithmetic circuit $\mathcal{C}$ over a field $\mathbb{F}$:



- Identity testing is the problem of checking whether a given circuit is zero or not.

Formalizing Identity Testing

- We can assume that our polynomial expression is given in the form of an arithmetic circuit $\mathcal{C}$ over a field $\mathbb{F}$:



- Identity testing is the problem of checking whether a given circuit is zero or not.

Identity Testing is Important

Identity testing is instrumental in many complexity theory results:

- Graph matching is in RNC (Lovasz '79).
- PSPACE=IP (Shamir '90).
- NEXP=MIP (Babai-Fortnow-Lund '90).
- Even AKS Primality test is based on checking whether $(x+1)^n - (x^n + 1) = 0 \pmod{n}$ (Agrawal-Kayal-S '02).

Identity Testing is Important

Identity testing is instrumental in many complexity theory results:

- Graph matching is in RNC (Lovasz '79).
- PSPACE=IP (Shamir '90).
- NEXP=MIP (Babai-Fortnow-Lund '90).
- Even AKS Primality test is based on checking whether $(x+1)^n - (x^n+1) = 0 \pmod{n}$ (Agrawal-Kayal-S '02).

Identity Testing is Important

Identity testing is instrumental in many complexity theory results:

- Graph matching is in RNC (Lovasz '79).
- PSPACE=IP (Shamir '90).
- NEXP=MIP (Babai-Fortnow-Lund '90).
- Even AKS Primality test is based on checking whether $(x+1)^n - (x^n+1) = 0 \pmod{n}$ (Agrawal-Kayal-S '02).

Identity Testing is Important

Identity testing is instrumental in many complexity theory results:

- Graph matching is in RNC (Lovasz '79).
- PSPACE=IP (Shamir '90).
- NEXP=MIP (Babai-Fortnow-Lund '90).
- Even AKS Primality test is based on checking whether $(x+1)^n - (x^n+1) = 0 \pmod{n}$ (Agrawal-Kayal-S '02).

Identity Testing is Important

Identity testing is instrumental in many complexity theory results:

- Graph matching is in RNC (Lovasz '79).
- PSPACE=IP (Shamir '90).
- NEXP=MIP (Babai-Fortnow-Lund '90).
- Even AKS Primality test is based on checking whether $(x+1)^n - (x^n+1) = 0 \pmod{n}$ (Agrawal-Kayal-S '02).

A Randomized Solution

- (Schwartz '80, Zippel '79) gave a randomized algorithm for identity testing.
- Given an arithmetic circuit $C(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$:
  - Pick a random tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$.
  - Return YES iff $C(\alpha_1, \ldots, \alpha_n) = 0$.
- Clearly, this can be done in time polynomial in the size of C.

A Randomized Solution

- (Schwartz '80, Zippel '79) gave a randomized algorithm for identity testing.
- Given an arithmetic circuit $C(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$:
  - Pick a random tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$.
  - Return YES iff $C(\alpha_1, \ldots, \alpha_n) = 0$.
- Clearly, this can be done in time polynomial in the size of C.

A Randomized Solution

- (Schwartz '80, Zippel '79) gave a randomized algorithm for identity testing.
- Given an arithmetic circuit $C(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$:
  - Pick a random tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$.
  - Return YES iff $C(\alpha_1, \ldots, \alpha_n) = 0$.
- Clearly, this can be done in time polynomial in the size of C.

A Randomized Solution

- (Schwartz '80, Zippel '79) gave a randomized algorithm for identity testing.
- Given an arithmetic circuit $C(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$:
  - Pick a random tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$.
  - Return YES iff $C(\alpha_1, \ldots, \alpha_n) = 0$.
- Clearly, this can be done in time polynomial in the size of C.

A Randomized Solution

- (Schwartz '80, Zippel '79) gave a randomized algorithm for identity testing.
- Given an arithmetic circuit $C(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$:
  - Pick a random tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$.
  - Return YES iff $C(\alpha_1, \ldots, \alpha_n) = 0$.
- Clearly, this can be done in time polynomial in the size of C.

The Question

Big question here: Can we do identity testing in deterministic polynomial time?

Consequences of Derandomization

- (Impagliazzo-Kabanets '03) showed that a derandomized identity test would imply circuit lower bounds for permanent.
- Thus, a derandomization of identity testing would both:
  - provide evidence that randomization in algorithms is dispensable, and
  - give lower bounds.

Consequences of Derandomization

- (Impagliazzo-Kabanets '03) showed that a derandomized identity test would imply circuit lower bounds for permanent.
- Thus, a derandomization of identity testing would both:
  - provide evidence that randomization in algorithms is dispensable, and
  - give lower bounds.

Consequences of Derandomization

- (Impagliazzo-Kabanets '03) showed that a derandomized identity test would imply circuit lower bounds for permanent.
- Thus, a derandomization of identity testing would both:
  - provide evidence that randomization in algorithms is dispensable, and
  - give lower bounds.

Consequences of Derandomization

- (Impagliazzo-Kabanets '03) showed that a derandomized identity test would imply circuit lower bounds for permanent.
- Thus, a derandomization of identity testing would both:
  - provide evidence that randomization in algorithms is dispensable, and
  - give lower bounds.

Progress

- Some progress has been made for restricted circuits.
- Noncommutative formulas: (Raz-Shpilka '04) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded top fanin: (Kayal-S '06) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded many distinct linear forms in each multiplication gate: (S '07) gives a deterministic polynomial time identity test.

Progress

- Some progress has been made for restricted circuits.
- Noncommutative formulas: (Raz-Shpilka '04) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded top fanin: (Kayal-S '06) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded many distinct linear forms in each multiplication gate: (S '07) gives a deterministic polynomial time identity test.

Progress

- Some progress has been made for restricted circuits.
- Noncommutative formulas: (Raz-Shpilka '04) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded top fanin: (Kayal-S '06) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded many distinct linear forms in each multiplication gate: (S '07) gives a deterministic polynomial time identity test.

Progress

- Some progress has been made for restricted circuits.
- Noncommutative formulas: (Raz-Shpilka '04) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded top fanin: (Kayal-S '06) gave a deterministic polynomial time identity test.
- Circuits of depth 3 with bounded many distinct linear forms in each multiplication gate: (S '07) gives a deterministic polynomial time identity test.

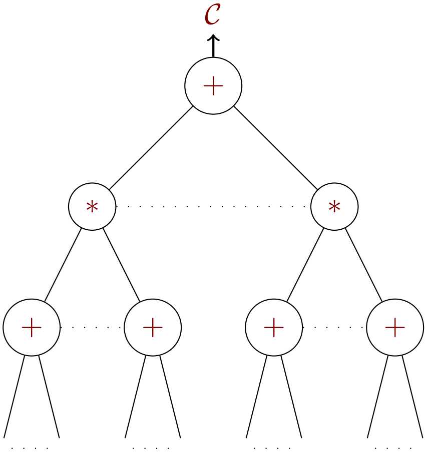
Outline

Depth 3 Circuits: The Setting

- For identity testing, it is sufficient to consider a "sum of product of linear functions" ($\Sigma\Pi\Sigma$ *circuit*).

Depth 3 Circuits: The Setting

- For identity testing, it is sufficient to consider a "sum of product of linear functions" ($\Sigma\Pi\Sigma$ *circuit*).



The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$
  where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \geq d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$
  where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \geq d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$ where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \geq d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$ where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \geq d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$ where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \geq d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1,\ldots,x_n) = T_1 + \cdots + T_k$
  where $T_i = L_{i,1}\cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1},\ldots,p_\ell^{e_\ell}$ of coprime linear functions $p_1,\ldots,p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \;\geq\; d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1,\ldots,x_n$.
  Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1),\ldots,\tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1),\ldots,\tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$
  where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \ \geq \ d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$.
  Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$ where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \ \geq \ d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$
  where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \ \geq \ d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$.
  Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

The Idea of Chinese Remaindering

- Let $\mathcal{C}$ be: $\mathcal{C}(x_1, \ldots, x_n) = T_1 + \cdots + T_k$ where $T_i = L_{i,1} \cdots L_{i,d}$ product of d linear functions.
- Pick powers $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ of coprime linear functions $p_1, \ldots, p_\ell$ such that,
  1. every $p_i^{e_i}$ divides some T_j.
  2. $e_1 + \cdots + e_\ell \;\geq\; d$.
- $\mathcal{C} = 0$ **iff** for all $i \in [\ell]$, $\mathcal{C} = 0 \pmod{p_i^{e_i}}$.
- We transform $p_i \mapsto x_1$ by applying an invertible map τ on $x_1, \ldots, x_n$. Then $\mathcal{C} = 0 \pmod{p_i^{e_i}}$ **iff**

$$\mathcal{C}(\tau(x_1), \ldots, \tau(x_n)) = 0 \text{ over } \mathbb{F}[x_1]/(x_1^{e_i}).$$

- Note that $\mathcal{C}(\tau(x_1), \ldots, \tau(x_n))$ modulo $x_1^{e_i}$ has fanin atmost $(k-1)$.
- Thus, we recursively solve identity testing over "bigger" rings.

Time Complexity

- Note that in each recursive call:
  1. Fanin k reduces by atleast 1
  2. Dimension of the base ring increases atmost d times.
- The computations that we do are on rings of dimension atmost d^k.
- Identity testing for depth 3 circuits over n variables, total degree d and top fanin k can be done in time $poly(d^k, n)$.

Time Complexity

- Note that in each recursive call:
  1. Fanin k reduces by atleast 1
  2. Dimension of the base ring increases atmost d times.
- The computations that we do are on rings of dimension atmost d^k.
- Identity testing for depth 3 circuits over n variables, total degree d and top fanin k can be done in time $poly(d^k, n)$.

Time Complexity

- Note that in each recursive call:
  1. Fanin k reduces by atleast 1
  2. Dimension of the base ring increases atmost d times.
- The computations that we do are on rings of dimension atmost d^k.
- Identity testing for depth 3 circuits over n variables, total degree d and top fanin k can be done in time $poly(d^k, n)$.

Time Complexity

- Note that in each recursive call:
  1. Fanin k reduces by atleast 1
  2. Dimension of the base ring increases atmost d times.
- The computations that we do are on rings of dimension atmost d^k.
- Identity testing for depth 3 circuits over n variables, total degree d and top fanin k can be done in time $poly(d^k, n)$.

Time Complexity

- Note that in each recursive call:
  1. Fanin k reduces by atleast 1
  2. Dimension of the base ring increases atmost d times.
- The computations that we do are on rings of dimension atmost d^k.
- Identity testing for depth 3 circuits over n variables, total degree d and top fanin k can be done in time $poly(d^k, n)$.

Outline

Idea for Diagonal Circuits

- Suppose the depth 3 circuit is a sum of powers of linear functions:

$$\mathcal{C}(x_1, \ldots, x_n) = \ell_1^d + \cdots + \ell_k^d$$

- The idea is to transform it into:
  $\sum_{i=1}^{k}(b_{i,1,0}+b_{i,1,1}x_1+\cdots+b_{i,1,d}x_1^d)\cdots(b_{i,n,0}+b_{i,n,1}x_n+\cdots+b_{i,n,d}x_n^d)$
- The above circuit can be viewed as a noncommutative circuit (*i.e.* $x_1, \ldots, x_n$ do not commutate).
- Thus, we can use the known results to identity test $\mathcal{C}$.
- As a by-product we also get exponential lower bounds for permanent and determinant.

Idea for Diagonal Circuits

- Suppose the depth 3 circuit is a sum of powers of linear functions:

$$\mathcal{C}(x_1, \ldots, x_n) = \ell_1^d + \cdots + \ell_k^d$$

- The idea is to transform it into:
  $\sum_{i=1}^{k} (b_{i,1,0} + b_{i,1,1}x_1 + \cdots + b_{i,1,d}x_1^d) \cdots (b_{i,n,0} + b_{i,n,1}x_n + \cdots + b_{i,n,d}x_n^d)$
- The above circuit can be viewed as a noncommutative circuit (*i.e.* $x_1, \ldots, x_n$ do not commutate).
- Thus, we can use the known results to identity test $\mathcal{C}$.
- As a by-product we also get exponential lower bounds for permanent and determinant.

Idea for Diagonal Circuits

- Suppose the depth 3 circuit is a sum of powers of linear functions:

$$\mathcal{C}(x_1, \ldots, x_n) = \ell_1^d + \cdots + \ell_k^d$$

- The idea is to transform it into: $\sum_{i=1}^{k}(b_{i,1,0}+b_{i,1,1}x_1+\cdots+b_{i,1,d}x_1^d)\cdots(b_{i,n,0}+b_{i,n,1}x_n+\cdots+b_{i,n,d}x_n^d)$
- The above circuit can be viewed as a noncommutative circuit (*i.e.* $x_1, \ldots, x_n$ do not commutate).
- Thus, we can use the known results to identity test $\mathcal{C}$.
- As a by-product we also get exponential lower bounds for permanent and determinant.

Idea for Diagonal Circuits

- Suppose the depth 3 circuit is a sum of powers of linear functions:

$$\mathcal{C}(x_1,\ldots,x_n) = \ell_1^d + \cdots + \ell_k^d$$

- The idea is to transform it into:
$\sum_{i=1}^{k}(b_{i,1,0}+b_{i,1,1}x_1+\cdots+b_{i,1,d}x_1^d)\cdots(b_{i,n,0}+b_{i,n,1}x_n+\cdots+b_{i,n,d}x_n^d)$
- The above circuit can be viewed as a noncommutative circuit (*i.e.* $x_1,\ldots,x_n$ do not commutate).
- Thus, we can use the known results to identity test $\mathcal{C}$.
- As a by-product we also get exponential lower bounds for permanent and determinant.

Idea for Diagonal Circuits

- Suppose the depth 3 circuit is a sum of powers of linear functions:

$$\mathcal{C}(x_1, \ldots, x_n) = \ell_1^d + \cdots + \ell_k^d$$

- The idea is to transform it into:
$\sum_{i=1}^{k}(b_{i,1,0}+b_{i,1,1}x_1+\cdots+b_{i,1,d}x_1^d)\cdots(b_{i,n,0}+b_{i,n,1}x_n+\cdots+b_{i,n,d}x_n^d)$
- The above circuit can be viewed as a noncommutative circuit (*i.e.* $x_1, \ldots, x_n$ do not commutate).
- Thus, we can use the known results to identity test $\mathcal{C}$.
- As a by-product we also get exponential lower bounds for permanent and determinant.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} z^{-dj} \cdot \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} z^{-dj} \cdot \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} z^{-dj} \cdot \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} z^{-dj} \cdot \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1,\ldots,x_n) = \sum_{i=1}^{k}(a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1}\cdot\mathcal{C}(x_1,\ldots,x_n) =$ coefficient of z^d in $\sum_{i=1}^{k}\exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k}\exp(a_{i,1}x_1z)\cdots\exp(a_{i,n}x_nz)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k}\prod_{v=1}^{n}\left(1 + a_{i,v}x_vz + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k}\sum_{j=1}^{dn+1} \quad z^{-dj} \cdot \prod_{v=1}^{n}\left(1 + a_{i,v}x_vz^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1}-1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} z^{-dj} \cdot \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: The Transformation

- We will present the circuit transformation over a field $\mathbb{F}$ of zero characteristic.
- Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} (a_{i,1}x_1 + \cdots + a_{i,n}x_n)^d$. Recall $\exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$
- $\Rightarrow \quad (d!)^{-1} \cdot \mathcal{C}(x_1, \ldots, x_n) =$ coefficient of z^d in $\sum_{i=1}^{k} \exp\left((a_{1,1}x_1 + \cdots + a_{1,n}x_n)z\right)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \exp(a_{i,1}x_1 z) \cdots \exp(a_{i,n}x_n z)$.
- $=$ coefficient of z^d in $\sum_{i=1}^{k} \prod_{v=1}^{n} \left(1 + a_{i,v}x_v z + \cdots + \frac{a_{i,v}^d x_v^d z^d}{d!}\right)$.
- $= \sum_{i=1}^{k} \sum_{j=1}^{dn+1} \quad z^{-dj} \cdot$ $\prod_{v=1}^{n} \left(1 + a_{i,v}x_v z^j + \cdots + \frac{a_{i,v}^d x_v^d z^{dj}}{d!}\right) \pmod{z^{dn+1} - 1}$.

Idea for Diagonal Circuits: Generalized

- We transformed a diagonal circuit $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i \in [k]} \ell_i^d$ to a noncommutative circuit over the ring $\mathbb{F}[z]/(z^{dn+1} - 1)$.
- This transformation on the multiplication gates generalizes:

Theorem [S '07]

Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ where $\ell_{i,1}, \ldots, \ell_{i,c}$ are linear functions and say $(e_{1,1} + \cdots + e_{1,c}) =: d$ which is the total degree of the polynomial $\mathcal{C}(x_1, \ldots, x_n)$. Then identity testing of $\mathcal{C}$ can be done in time $poly((e_{1,1} + 1) \cdots (e_{1,c} + 1), k, n)$.

- This immediately gives a $poly(2^d, k, n)$ time identity test for general depth 3 circuits.

Idea for Diagonal Circuits: Generalized

- We transformed a diagonal circuit $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i \in [k]} \ell_i^d$ to a noncommutative circuit over the ring $\mathbb{F}[z]/(z^{dn+1} - 1)$.
- This transformation on the multiplication gates generalizes:

Theorem [S '07]

Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ where $\ell_{i,1}, \ldots, \ell_{i,c}$ are linear functions and say $(e_{1,1} + \cdots + e_{1,c}) =: d$ which is the total degree of the polynomial $\mathcal{C}(x_1, \ldots, x_n)$. Then identity testing of $\mathcal{C}$ can be done in time $poly((e_{1,1}+1)\cdots(e_{1,c}+1), k, n)$.

- This immediately gives a $poly(2^d, k, n)$ time identity test for general depth 3 circuits.

Idea for Diagonal Circuits: Generalized

- We transformed a diagonal circuit $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i \in [k]} \ell_i^d$ to a noncommutative circuit over the ring $\mathbb{F}[z]/(z^{dn+1} - 1)$.
- This transformation on the multiplication gates generalizes:

Theorem [S '07]

Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ where $\ell_{i,1}, \ldots, \ell_{i,c}$ are linear functions and say $(e_{1,1} + \cdots + e_{1,c}) =: d$ which is the total degree of the polynomial $\mathcal{C}(x_1, \ldots, x_n)$. Then identity testing of $\mathcal{C}$ can be done in time $poly((e_{1,1}+1) \cdots (e_{1,c}+1), k, n)$.

- This immediately gives a $poly(2^d, k, n)$ time identity test for general depth 3 circuits.

Idea for Diagonal Circuits: Generalized

- We transformed a diagonal circuit $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i \in [k]} \ell_i^d$ to a noncommutative circuit over the ring $\mathbb{F}[z]/(z^{dn+1} - 1)$.
- This transformation on the multiplication gates generalizes:

Theorem [S '07]

Let $\mathcal{C}(x_1, \ldots, x_n) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ where $\ell_{i,1}, \ldots, \ell_{i,c}$ are linear functions and say $(e_{1,1} + \cdots + e_{1,c}) =: d$ which is the total degree of the polynomial $\mathcal{C}(x_1, \ldots, x_n)$. Then identity testing of $\mathcal{C}$ can be done in time $poly((e_{1,1}+1) \cdots (e_{1,c}+1), k, n)$.

- This immediately gives a $poly(2^d, k, n)$ time identity test for general depth 3 circuits.

Idea for Diagonal Circuits proves Lower Bounds

- (Nisan '91) had proved exponential lower bounds on the size of noncommutative circuits computing determinant or permanent.
- Thus, we get that:

[S '07]

If $\mathcal{C}(x_1, \ldots, x_{n^2}) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ of degree d is computing the determinant or permanent of an $n \times n$ matrix then:

1. either $c = \Omega\left(\frac{n}{\log d}\right)$.
2. or $k = 2^{\Omega(n)}$.

Idea for Diagonal Circuits proves Lower Bounds

- (Nisan '91) had proved exponential lower bounds on the size of noncommutative circuits computing determinant or permanent.
- Thus, we get that:

[S '07]

If $\mathcal{C}(x_1, \ldots, x_{n^2}) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ of degree d is computing the determinant or permanent of an $n \times n$ matrix then:

1. either $c = \Omega\left(\frac{n}{\log d}\right)$.
2. or $k = 2^{\Omega(n)}$.

Idea for Diagonal Circuits proves Lower Bounds

- (Nisan '91) had proved exponential lower bounds on the size of noncommutative circuits computing determinant or permanent.
- Thus, we get that:

[S '07]

If $\mathcal{C}(x_1, \ldots, x_{n^2}) = \sum_{i=1}^{k} \ell_{i,1}^{e_{i,1}} \cdots \ell_{i,c}^{e_{i,c}}$ of degree d is computing the determinant or permanent of an $n \times n$ matrix then:

1. either $c = \Omega\left(\frac{n}{\log d}\right)$.
2. or $k = 2^{\Omega(n)}$.

Outline

Epilogue

- Derandomization can be done under hardness assumptions.
- Derandomization (of identity testing) will prove some hardness results.
- For concrete algebraic problems derandomization may be done if we understand the underlying structure well enough.
- Identity testing and polynomial factoring are waiting to be derandomized!

Epilogue

- Derandomization can be done under hardness assumptions.
- Derandomization (of identity testing) will prove some hardness results.
- For concrete algebraic problems derandomization may be done if we understand the underlying structure well enough.
- Identity testing and polynomial factoring are waiting to be derandomized!

Epilogue

- Derandomization can be done under hardness assumptions.
- Derandomization (of identity testing) will prove some hardness results.
- For concrete algebraic problems derandomization may be done if we understand the underlying structure well enough.
- Identity testing and polynomial factoring are waiting to be derandomized!

Epilogue

- Derandomization can be done under hardness assumptions.
- Derandomization (of identity testing) will prove some hardness results.
- For concrete algebraic problems derandomization may be done if we understand the underlying structure well enough.
- Identity testing and polynomial factoring are waiting to be derandomized!