CS244: Theory of Computation

Fu Song
ShanghaiTech University

Fall 2022

Outline

Outline

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**$=$ **NP**

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**= **NP**
- In practice, we may not need the absolute best or optimal solution to a problem

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**$=$ **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**$=$ **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find
- An approximation algorithm is designed to find such approximately optimal solutions

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**= **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find
- An approximation algorithm is designed to find such approximately optimal solutions
  - Minimization problem is to find a minimal solution

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**$=$ **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find
- An approximation algorithm is designed to find such approximately optimal solutions
  - Minimization problem is to find a minimal solution
  - Maximization problem is to find a maximal solution

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**= **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find
- An approximation algorithm is designed to find such approximately optimal solutions
  - Minimization problem is to find a minimal solution
  - Maximization problem is to find a maximal solution
  - For a minimization problem, a k-optimal approximation algorithm always finds a solution that is not more than k times optimal

Approximation Algorithm

- Optimization problems aim at finding the best solution among a collection of possible solutions
- When optimization problem is **NP**-hard or even worse, no polynomial time algorithm exists that finds the best solution unless **P**$=$ **NP**
- In practice, we may not need the absolute best or optimal solution to a problem
- A solution that is nearly optimal may be good enough and may be much easier to find
- An approximation algorithm is designed to find such approximately optimal solutions
  - Minimization problem is to find a minimal solution
  - Maximization problem is to find a maximal solution
  - For a minimization problem, a k-optimal approximation algorithm always finds a solution that is not more than k times optimal
  - For a maximization problem, a k-optimal approximation algorithm always finds a solution that is at least $\frac{1}{k}$ times the size of the optimal

MIN-VERTEX-COVER

If G is an undirected graph, a vertex cover of G is a subset S of the nodes where every edge of G touches one of those nodes,

MIN-VERTEX-COVER

If G is an undirected graph, a vertex cover of G is a subset S of the nodes where every edge of G touches one of those nodes, formally, for every edge (x, y) in G, $x \in S \vee y \in S$.

MIN-VERTEX-COVER

If G is an undirected graph, a vertex cover of G is a subset S of the nodes where every edge of G touches one of those nodes, formally, for every edge (x, y) in G, $x \in S \vee y \in S$.

$$\mathit{VERTEX\text{-}COVER} = \{\langle G, k\rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}.$$

MIN-VERTEX-COVER

If G is an undirected graph, a vertex cover of G is a subset S of the nodes where every edge of G touches one of those nodes, formally, for every edge (x, y) in G, $x \in S \vee y \in S$.

$$\mathit{VERTEX\text{-}COVER} = \{\langle G, k\rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}.$$

Theorem

VERTEX-COVER is **NP***-complete.*

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes,

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes, then X is twice as large as H, i.e., $|X| \leq 2|H|$

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes, then X is twice as large as H, i.e., $|X| \leq 2|H|$
- Let Y be the a minimal vertex cover of G

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes, then X is twice as large as H, i.e., $|X| \leq 2|H|$
- Let Y be the a minimal vertex cover of G
- $|Y|$ is equal to or lager than $|H|$, i.e., $2|Y| \geq 2|H|$

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge $[O(|E| \times |V|)$ time$]$
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes, then X is twice as large as H, i.e., $|X| \leq 2|H|$
- Let Y be the a minimal vertex cover of G
- $|Y|$ is equal to or lager than $|H|$, i.e., $2|Y| \geq 2|H| \geq |X|$

MIN-VERTEX-COVER

A On input $\langle G \rangle$, where G is an undirected graph

1. Repeat the following until all edges in G touch a marked edge:
2. Find an edge in G untouched by any marked edge
3. Mark that edge [$O(|E| \times |V|)$ time]
4. Output all nodes that are endpoints of marked edges

Theorem

A is a polynomial time algorithm that produces a vertex cover of G that is no more than twice as large as a minimal vertex cover

- H denotes marked edges and X denoted outputted nodes, then X is twice as large as H, i.e., $|X| \leq 2|H|$
- Let Y be the a minimal vertex cover of G
- $|Y|$ is equal to or lager than $|H|$, i.e., $2|Y| \geq 2|H| \geq |X|$
- This algorithm is a 2-optimal approximation algorithm

MAX-CUT

- A cut in an undirected graph is a separation of the vertices V into two disjoint subsets S and T

MAX-CUT

- A cut in an undirected graph is a separation of the vertices V into two disjoint subsets S and T
- A cut edge is an edge that goes between a node in S and a node in T

MAX-CUT

- A cut in an undirected graph is a separation of the vertices V into two disjoint subsets S and T
- A cut edge is an edge that goes between a node in S and a node in T
- An uncut edge is an edge that is not a cut edge

MAX-CUT

- A cut in an undirected graph is a separation of the vertices V into two disjoint subsets S and T
- A cut edge is an edge that goes between a node in S and a node in T
- An uncut edge is an edge that is not a cut edge
- The size of a cut is the number of cut edges

MAX-CUT

- A cut in an undirected graph is a separation of the vertices V into two disjoint subsets S and T
- A cut edge is an edge that goes between a node in S and a node in T
- An uncut edge is an edge that is not a cut edge
- The size of a cut is the number of cut edges
- The MAX-CUT problem asks for a largest cut in a graph G.

Theorem

k-CUT problem is **NP***-complete.*

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]
3. If no such node exists, output the current cut and halt

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]
3. If no such node exists, output the current cut and halt

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]
3. If no such node exists, output the current cut and halt

Theorem

B is a polynomial time, 2-optimal approximation algorithm for MAX-CUT

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]
3. If no such node exists, output the current cut and halt

Theorem

B is a polynomial time, 2-optimal approximation algorithm for MAX-CUT

- At every node v of G, the number of cut edges is at least as large as the number of uncut edges, $|CutEdge(v)| \geq |UNCutEdge(v)|$, otherwise B would have shifted the node v to the other side

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage [$O(|E| \times |V|)$ time]
3. If no such node exists, output the current cut and halt

Theorem

B is a polynomial time, 2-optimal approximation algorithm for MAX-CUT

- At every node v of G, the number of cut edges is at least as large as the number of uncut edges, $|CutEdge(v)| \geq |UNCutEdge(v)|$, otherwise B would have shifted the node v to the other side
- $\sum_{v \in V} |CutEdge(v)| = 2|CutEdge|$, because every cut edge is counted once for each of its two endpoints

MAX-CUT

B On input $\langle G \rangle$, where G is an undirected graph with nodes V

1. Let $S = \emptyset$ and $T = V$:
2. If moving a single node v, either from S to T or from T to S, increases the size of the cut, make that move and repeat this stage $[O(|E| \times |V|)$ time$]$
3. If no such node exists, output the current cut and halt

Theorem

B is a polynomial time, 2-optimal approximation algorithm for MAX-CUT

- At every node v of G, the number of cut edges is at least as large as the number of uncut edges, $|CutEdge(v)| \geq |UNCutEdge(v)|$, otherwise B would have shifted the node v to the other side
- $\sum_{v \in V} |CutEdge(v)| = 2|CutEdge|$, because every cut edge is counted once for each of its two endpoints
- $|CutEdge| = \frac{\sum_{v \in V} |CutEdge(v)|}{2}$
- $\sum_{v \in V} |CutEdge(v)| \geq \sum_{v \in V} |UNCutEdge(v)|$
- $\sum_{v \in V} |CutEdge(v)| + \sum_{v \in V} |UNCutEdge(v)| = 2|E| \Rightarrow$ $\sum_{v \in V} |CutEdge(v)| \geq |E| \Rightarrow |CutEdge| \geq \frac{|E|}{2}$

Outline

Advanced Topics In Complexity Theory

- Approximation Algorithms
- Probabilistic Algorithms
- Alternation
- Interactive Proof Systems
- Parallel Computation
- Cryptography

Probabilistic Algorithm

Nature Perspective

“There are several reasons why probabilistic programming could prove to be revolutionary for machine intelligence and scientific modelling.”[1]

REVIEW

doi:10.1038/nature14541

Probabilistic machine learning and artificial intelligence

Zoubin Ghahramani[1]

[1]Zoubin Ghahramani leads the Cambridge Machine Learning Group, and holds positions at CMU, UCL, and the Alan Turing Institute.

Probabilistic Algorithm

- A probabilistic algorithm is an algorithm designed to use the outcome of a random process, e.g., by "flip a coin"[2]

[2]Probabilistic Programming at RWTH-Aachen University https://moves.rwth-aachen.de/teaching/ws-1819/probabilistic-programming

Probabilistic Algorithm

- A probabilistic algorithm is an algorithm designed to use the outcome of a random process, e.g., by “flip a coin”[2]
- Certain types of problems seem to be more easily solvable by probabilistic algorithms than by deterministic algorithms

[2] Probabilistic Programming at RWTH-Aachen University https://moves.rwth-aachen.de/teaching/ws-1819/probabilistic-programming

Probabilistic Algorithm

- A probabilistic algorithm is an algorithm designed to use the outcome of a random process, e.g., by "flip a coin"[2]
- Certain types of problems seem to be more easily solvable by probabilistic algorithms than by deterministic algorithms
- How can making a decision by flipping a coin ever be better than actually calculating, or even estimating, the best choice in a particular situation?

[2]Probabilistic Programming at RWTH-Aachen University
https://moves.rwth-aachen.de/teaching/ws-1819/probabilistic-programming

Probabilistic Algorithm

- A probabilistic algorithm is an algorithm designed to use the outcome of a random process, e.g., by "flip a coin"[2]
- Certain types of problems seem to be more easily solvable by probabilistic algorithms than by deterministic algorithms
- How can making a decision by flipping a coin ever be better than actually calculating, or even estimating, the best choice in a particular situation?
- Sometimes, calculating the best choice may require excessive time, and estimating it may introduce a bias that invalidates the result

[2] Probabilistic Programming at RWTH-Aachen University
https://moves.rwth-aachen.de/teaching/ws-1819/probabilistic-programming

Sorting by flipping coins

Quicksort:

```
QS(A) =
  if |A| <= 1 { return A; }
  i := ceil(|A|/2);
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return QS(A<) ++ A[i] ++ QS(A>)
```

Worst case complexity:
$O(N^2)$ comparisons

Randomised Quicksort:

```
rQS(A) =
  if |A| <= 1 { return A; }
  i := Unif[1...|A|];
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return rQS(A<) ++ A[i] ++ rQS(A>)
```

Worst case complexity:
$O(N \log N)$ expected comparisons

Monte Carlo: Matrix multiplication

Input: three N^2 square matrices A, B, and C
Output: yes, if $A \cdot B = C$; no, otherwise

- until end 1960s: cubic ($= 3$)
- 1969: 2.808
- 1978: 2.796
- 1979: 2.780
- 1981: 2.522
- 1984: 2.496
- 1989: 2.376
- 2014: 2.373
- 2100: $\cdots$

Monte Carlo: Freivald's matrix multiplication

Input: three $\mathcal{O}(N^2)$ square matrices A, B, and C

Output: yes, if $A \times B = C$; no, otherwise

Deterministic: compute $A \times B$ and compare with C

Complexity: in $\mathcal{O}(N^3)$, best known complexity $\mathcal{O}(N^{2.37})$

Randomised:

1. take a random bit-vector $\vec{x}$ of size N
2. compute $A \times (B\,\vec{x}) - C\,\vec{x}$
3. output yes if this yields the null vector; no otherwise
4. repeat these steps k times

Complexity: in $\mathcal{O}(k{\cdot}N^2)$, with false positive with probability $\leqslant 2^{-k}$

Probabilistic Turing Machine

Definition

A probabilistic Turing machine M is a type of nondeterministic Turing machine in which each nondeterministic step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of M's computation on input w as follows. Define the probability of branch b to be

$$Pr[b] = 2^{-k}$$

where k is the number of coin-flip steps that occur on branch b.

Probabilistic Turing Machine

Definition

A probabilistic Turing machine M is a type of nondeterministic Turing machine in which each nondeterministic step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of M's computation on input w as follows. Define the probability of branch b to be

$$Pr[b] = 2^{-k}$$

where k is the number of coin-flip steps that occur on branch b.

Define the probability that M accepts w to be

$$Pr[M \text{ accepts } w] = \sum_{b \text{ is an accepting branch}} Pr[b]$$

Probabilistic Turing Machine

Definition

A probabilistic Turing machine M is a type of nondeterministic Turing machine in which each nondeterministic step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of M's computation on input w as follows. Define the probability of branch b to be

$$Pr[b] = 2^{-k}$$

where k is the number of coin-flip steps that occur on branch b.

Define the probability that M accepts w to be

$$Pr[M \text{ accepts } w] = \sum_{b \text{ is an accepting branch}} Pr[b]$$

$$Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w]$$

Probabilistic Turing Machine

For TM M of the language A, for every string w, we have

- $w \in A$ if M accepts w
- $w \notin A$ if M rejects w or does not halt

Probabilistic Turing Machine

For TM M of the language A, for every string w, we have

- $w \in A$ if M accepts w
- $w \notin A$ if M rejects w or does not halt

For Probabilistic TM M of a language A and a small probability of error $0 \leq \epsilon < \frac{1}{2}$, for every string w,

- if $w \in A$, then $Pr[M \text{ accepts } w] \geq 1 - \epsilon$ i.e., $Pr[M \text{ rejects } w] \leq \epsilon$
- if $w \notin A$, then $Pr[M \text{ rejects } w] \geq 1 - \epsilon$, i.e., $Pr[M \text{ accepts } w] \leq \epsilon$

Note: $Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w]$

Consider $\epsilon = \frac{1}{4}$, where $Pr[M \text{ accepts } w] = \frac{1}{2}$, then

neither $w \in A$ nor $w \notin A$

Probabilistic Turing Machine

For TM M of the language A, for every string w, we have

- $w \in A$ if M accepts w
- $w \notin A$ if M rejects w or does not halt

For Probabilistic TM M of a language A and a small probability of error $0 \leq \epsilon < \frac{1}{2}$, for every string w,

- if $w \in A$, then $Pr[M \text{ accepts } w] \geq 1 - \epsilon$ i.e., $Pr[M \text{ rejects } w] \leq \epsilon$
- if $w \notin A$, then $Pr[M \text{ rejects } w] \geq 1 - \epsilon$, i.e., $Pr[M \text{ accepts } w] \leq \epsilon$

Note: $Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w]$

Consider $\epsilon = \frac{1}{4}$, where $Pr[M \text{ accepts } w] = \frac{1}{2}$, then

neither $w \in A$ nor $w \notin A$

We may consider error probability bounds that depend on the input length n, e.g., $\epsilon = \frac{1}{2^n}$

Bounded-Error Probabilistic Polynomial-Time (BPP)

Time and space complexity of a probabilistic Turing machine in the same way we do for a nondeterministic Turing machine: by using the worst case computation branch on each input.

Bounded-Error Probabilistic Polynomial-Time (BPP)

Time and space complexity of a probabilistic Turing machine in the same way we do for a nondeterministic Turing machine: by using the worst case computation branch on each input.

Definition

Bounded-Error **P**robabilistic **P**olynomial-Time (**BPP**) is the class of languages that are decided by probabilistic polynomial time Turing machines with an error probability of $\frac{1}{3}$.

Note: $\frac{1}{3}$ can be any bounded error $0 \leq \epsilon < \frac{1}{2}$.

Bounded-Error Probabilistic Polynomial-Time (BPP)

Lemma

Let the bounded error $0 \leq \epsilon < \frac{1}{2}$. Then for any polynomial $p(n)$, a probabilistic polynomial time Turing machine M_1 that operates with error probability ϵ has an equivalent probabilistic polynomial time Turing machine M_2 that operates with an error probability of $\frac{1}{2^{p(n)}}$.

Bounded-Error Probabilistic Polynomial-Time (BPP)

Lemma

Let the bounded error $0 \leq \epsilon < \frac{1}{2}$. Then for any polynomial $p(n)$, a probabilistic polynomial time Turing machine M_1 that operates with error probability ϵ has an equivalent probabilistic polynomial time Turing machine M_2 that operates with an error probability of $\frac{1}{2^{p(n)}}$.

Idea:

1. M_2 simulates M_1 by running it a polynomial number of times and taking the majority vote of the outcomes

Bounded-Error Probabilistic Polynomial-Time (BPP)

Lemma

Let the bounded error $0 \leq \epsilon < \frac{1}{2}$. Then for any polynomial $p(n)$, a probabilistic polynomial time Turing machine M_1 that operates with error probability ϵ has an equivalent probabilistic polynomial time Turing machine M_2 that operates with an error probability of $\frac{1}{2^{p(n)}}$.

Idea:

1. M_2 simulates M_1 by running it a polynomial number of times and taking the majority vote of the outcomes
2. The probability of error decreases exponentially with the number of runs of M_1 made

Proof

M_2 On input x

1. Calculate k (see analysis below)

Proof

M_2 On input x

1. Calculate k (see analysis below)
2. Run $2k$ independent simulations of M_1 on input x

Proof

M_2 On input x

1. Calculate k (see analysis below)
2. Run $2k$ independent simulations of M_1 on input x
3. If most runs of M_1 accept, then accept; otherwise, reject

Proof

- Let S be the sequence of stage 2,

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results,

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x)$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2)$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2)$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1 - \epsilon_x)\epsilon_x \leq (1 - \epsilon)\epsilon$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1 - \epsilon_x)\epsilon_x \leq (1 - \epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$,

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1 - \epsilon_x)\epsilon_x \leq (1 - \epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w = (1 - \epsilon_x)^c (\epsilon_x)^c (\epsilon_x)^{w-c} \leq (1 - \epsilon)^c (\epsilon)^c (\epsilon)^{w-c} = (1 - \epsilon)^c (\epsilon)^w$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1 - \epsilon_x)\epsilon_x \leq (1 - \epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1 - \epsilon_x)^c(\epsilon_x)^w = (1 - \epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1 - \epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1 - \epsilon)^c(\epsilon)^w$
  - $\epsilon < (1 - \epsilon)$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1 - \epsilon_x)\epsilon_x \leq (1 - \epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1 - \epsilon_x)^c (\epsilon_x)^w = (1 - \epsilon_x)^c (\epsilon_x)^c (\epsilon_x)^{w-c} \leq (1 - \epsilon)^c (\epsilon)^c (\epsilon)^{w-c} = (1 - \epsilon)^c (\epsilon)^w$
  - $\epsilon < (1 - \epsilon) \Rightarrow \epsilon^{c-k} \geq (1 - \epsilon)^{c-k}$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \leq (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \geq (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \geq (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \le \epsilon_x \le \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \le (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \le \epsilon_x \le \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \ge (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \ge (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \ge (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \le (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \le w \ge k$ and $c \le k$, $P_s \le (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \le (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \ge (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \ge (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \ge (1-\epsilon)^c\epsilon^w$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \leq (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \geq (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \geq (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \geq (1-\epsilon)^c\epsilon^w \geq P_S$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \leq (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \geq (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \geq (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \geq (1-\epsilon)^c\epsilon^w \geq P_S$
- Let $k \geq \log_{4\epsilon(1-\epsilon)} 2^{-p(n)}$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \leq (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \geq (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \geq (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \geq (1-\epsilon)^c\epsilon^w \geq P_S$
- Let $k \geq \log_{4\epsilon(1-\epsilon)} 2^{-p(n)}$
- $Pr[M_2$ outputs incorrectly on input $x] = \sum_{bad\ S} P_S$
  $\leq 2^{2k}\epsilon^k(1-\epsilon)^k = (4\epsilon(1-\epsilon))^k$

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \le \epsilon_x \le \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \le (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \le \epsilon_x \le \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \ge (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \ge (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \ge (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \le (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \le w \ge k$ and $c \le k$, $P_s \le (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \le (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \ge (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \ge (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \ge (1-\epsilon)^c\epsilon^w \ge P_S$
- Let $k \ge \log_{4\epsilon(1-\epsilon)} 2^{-p(n)}$
- $Pr[M_2$ outputs incorrectly on input $x] = \sum_{bad\ S} P_S$
  $\le 2^{2k}\epsilon^k(1-\epsilon)^k = (4\epsilon(1-\epsilon))^k = (4\epsilon(1-\epsilon))^{-p(n)\log_{4\epsilon(1-\epsilon)} 2} = 2^{-p(n)}$
  Note: there are at most 2^{2k} bad sequences

Proof

- Let S be the sequence of stage 2, S has c correct results and w wrong results, then $c + w = 2k$
- Let ϵ_x be the probability that M_1 is wrong on x, then $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2}$
- The probability that M_2 obtains S: $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w$
- $0 \leq \epsilon_x \leq \epsilon < \frac{1}{2} \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon - \epsilon_x)(\epsilon + \epsilon_x) \Rightarrow (\epsilon - \epsilon_x) \geq (\epsilon^2 - \epsilon_x^2) \Rightarrow (\epsilon - \epsilon^2) \geq (\epsilon_x - \epsilon_x^2) \Rightarrow (1-\epsilon_x)\epsilon_x \leq (1-\epsilon)\epsilon$
- If M_2 outputs incorrectly via S (called bad S), then
  - $c \leq w \geq k$ and $c \leq k$, $P_s \leq (1-\epsilon_x)^c(\epsilon_x)^w = (1-\epsilon_x)^c(\epsilon_x)^c(\epsilon_x)^{w-c} \leq (1-\epsilon)^c(\epsilon)^c(\epsilon)^{w-c} = (1-\epsilon)^c(\epsilon)^w$
  - $\epsilon < (1-\epsilon) \Rightarrow \epsilon^{c-k} \geq (1-\epsilon)^{c-k} \Rightarrow \epsilon^{c-k}(1-\epsilon)^k\epsilon^w \geq (1-\epsilon)^{c-k}(1-\epsilon)^k\epsilon^w \Rightarrow \epsilon^k(1-\epsilon)^k \geq (1-\epsilon)^c\epsilon^w \geq P_S$
- Let $k \geq \log_{4\epsilon(1-\epsilon)} 2^{-p(n)}$
- $Pr[M_2 \text{ outputs incorrectly on input } x] = \sum_{bad\ S} P_S$
  $\leq 2^{2k}\epsilon^k(1-\epsilon)^k = (4\epsilon(1-\epsilon))^k = (4\epsilon(1-\epsilon))^{-p(n)\log_{4\epsilon(1-\epsilon)} 2} = 2^{-p(n)}$
  Note: there are at most 2^{2k} bad sequences
- $Pr[M_1 \text{ accepts } w] \geq 1-\epsilon \iff Pr[M_2 \text{ accepts } w] \geq 1 - 2^{-p(n)}$
- $Pr[M_1 \text{ rejects } w] \geq 1-\epsilon \iff Pr[M_2 \text{ rejects } w] \geq 1 - 2^{-p(n)}$

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself,

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself, otherwise composite.

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself, otherwise composite.

- One way to determine whether a number is prime is to try all possible integers less than that number and see whether any are divisors, also called factors

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself, otherwise composite.

- One way to determine whether a number is prime is to try all possible integers less than that number and see whether any are divisors, also called factors it has exponential time complexity

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself, otherwise composite.

- One way to determine whether a number is prime is to try all possible integers less than that number and see whether any are divisors, also called factors it has exponential time complexity
- AKS primality test: this problem is known in **P**(2004), indeed, $O(n^6)$
  Not feasible when $n = 2048$

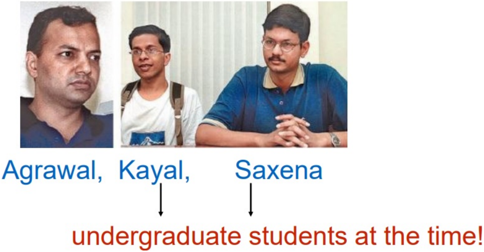
Agrawal, Kayal, Saxena

undergraduate students at the time!

Primality

A prime number is an integer greater than 1 that is not divisible by positive integers other than 1 and itself, otherwise composite.

- One way to determine whether a number is prime is to try all possible integers less than that number and see whether any are divisors, also called factors it has exponential time complexity
- AKS primality test: this problem is known in **P**(2004), indeed, $O(n^6)$ Not feasible when $n = 2048$



We describe a much simpler probabilistic polynomial time algorithm for primality testing, in $O(n^2)$, with tiny error probability

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod{p}$ be the smallest nonnegative y such that $x \equiv_p y$

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod p$ be the smallest nonnegative y such that $x \equiv_p y$
- Every number is equivalent modulo p to some member of the set $\mathbb{Z}_p = \{0, \cdots, p-1\}$

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod{p}$ be the smallest nonnegative y such that $x \equiv_p y$
- Every number is equivalent modulo p to some member of the set $\mathbb{Z}_p = \{0, \cdots, p-1\}$
- Let $\mathbb{Z}_p^+ = \{1, \cdots, p-1\}$, we will denote $p-1$ by -1 under equivalent modulo p

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod{p}$ be the smallest nonnegative y such that $x \equiv_p y$
- Every number is equivalent modulo p to some member of the set $\mathbb{Z}_p = \{0, \cdots, p-1\}$
- Let $\mathbb{Z}_p^+ = \{1, \cdots, p-1\}$, we will denote $p-1$ by -1 under equivalent modulo p

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod p$ be the smallest nonnegative y such that $x \equiv_p y$
- Every number is equivalent modulo p to some member of the set $\mathbb{Z}_p = \{0, \cdots, p-1\}$
- Let $\mathbb{Z}_p^+ = \{1, \cdots, p-1\}$, we will denote $p-1$ by -1 under equivalent modulo p

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- 7 is prime: $2^{7-1} = 64$ and $64 \pmod 7 = 1$, pass the Fermat test

Primality

- x and y are equivalent modulo p, $x \equiv_p y$, if they differ by a multiple of p
- Let $x \pmod{p}$ be the smallest nonnegative y such that $x \equiv_p y$
- Every number is equivalent modulo p to some member of the set $\mathbb{Z}_p = \{0, \cdots, p-1\}$
- Let $\mathbb{Z}_p^+ = \{1, \cdots, p-1\}$, we will denote $p-1$ by -1 under equivalent modulo p

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- 7 is prime: $2^{7-1} = 64$ and $64 \pmod{7} = 1$, pass the Fermat test
- 6 is nonprime: $2^{6-1} = 32$ and $32 \pmod{6} = 2$, fail the Fermat test

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

- For each $i \in \mathbb{Z}_p^+$, since p is prime, $i \times a \pmod{p} \in \mathbb{Z}_p^+$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

- For each $i \in \mathbb{Z}_p^+$, since p is prime, $i \times a \pmod{p} \in \mathbb{Z}_p^+$
- For each $i, j \in \mathbb{Z}_p^+$ such that $i \neq j$, $i \times a \not\equiv_p j \times a$, as $(i - j) \times a \not\equiv_p 0$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

- For each $i \in \mathbb{Z}_p^+$, since p is prime, $i \times a \pmod{p} \in \mathbb{Z}_p^+$
- For each $i, j \in \mathbb{Z}_p^+$ such that $i \neq j$, $i \times a \not\equiv_p j \times a$, as $(i-j) \times a \not\equiv_p 0$
- $\pmod{p} : \{a, 2a, 3a, \cdots, (p-1)a\} \to \mathbb{Z}_p^+$ is a bijective function

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

- For each $i \in \mathbb{Z}_p^+$, since p is prime, $i \times a \pmod{p} \in \mathbb{Z}_p^+$
- For each $i, j \in \mathbb{Z}_p^+$ such that $i \neq j$, $i \times a \not\equiv_p j \times a$, as $(i-j) \times a \not\equiv_p 0$
- $\pmod{p} : \{a, 2a, 3a, \cdots, (p-1)a\} \to \mathbb{Z}_p^+$ is a bijective function
- $a^{p-1} \prod_{i \in \mathbb{Z}_p^+} i = a \times 2a \times 3a \times \cdots \times (p-1)a \equiv_p \prod_{i \in \mathbb{Z}_p^+} i$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

For each $a \in \mathbb{Z}_p^+$, consider $a, 2a, 3a, \cdots, (p-1) \times a$

- For each $i \in \mathbb{Z}_p^+$, since p is prime, $i \times a \pmod{p} \in \mathbb{Z}_p^+$
- For each $i, j \in \mathbb{Z}_p^+$ such that $i \neq j$, $i \times a \not\equiv_p j \times a$, as $(i-j) \times a \not\equiv_p 0$
- $\pmod{p} : \{a, 2a, 3a, \cdots, (p-1)a\} \to \mathbb{Z}_p^+$ is a bijective function
- $a^{p-1} \prod_{i \in \mathbb{Z}_p^+} i = a \times 2a \times 3a \times \cdots \times (p-1)a \equiv_p \prod_{i \in \mathbb{Z}_p^+} i$
- $a^{p-1} \equiv_p 1$, as $\prod_{i \in \mathbb{Z}_p^+} i \not\equiv_p 0$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- A number p is called pseudoprime if it passes Fermat tests for all $a \in \mathbb{Z}_p^+$,

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- A number p is called pseudoprime if it passes Fermat tests for all $a \in \mathbb{Z}_p^+$, $\mathbb{Z}_p =$Galois finite field $\pmod{p}$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- A number p is called pseudoprime if it passes Fermat tests for all $a \in \mathbb{Z}_p^+$, $\mathbb{Z}_p$ =Galois finite field (mod p)
- A number p is prime iff it is pseudoprime but not Carmichael numbers, which are composite yet pass all Fermat tests

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- A number p is called pseudoprime if it passes Fermat tests for all $a \in \mathbb{Z}_p^+$, $\mathbb{Z}_p$ =Galois finite field (mod p)
- A number p is prime iff it is pseudoprime but not Carmichael numbers, which are composite yet pass all Fermat tests

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$,

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

- A number p is called pseudoprime if it passes Fermat tests for all $a \in \mathbb{Z}_p^+$, $\mathbb{Z}_p$ =Galois finite field (mod p)
- A number p is prime iff it is pseudoprime but not Carmichael numbers, which are composite yet pass all Fermat tests

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$
- If $b \in \mathbb{Z}_p^+$ is a nonwitness, i.e., $b^{p-1} \equiv_p 1$,

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$
- If $b \in \mathbb{Z}_p^+$ is a nonwitness, i.e., $b^{p-1} \equiv_p 1$, then $(a \times b \pmod{p})^{p-1} \not\equiv_p 1$

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$
- If $b \in \mathbb{Z}_p^+$ is a nonwitness, i.e., $b^{p-1} \equiv_p 1$, then $(a \times b \pmod{p})^{p-1} \not\equiv_p 1$ which implies that $a \times b \pmod{p}$ is witness

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$
- If $b \in \mathbb{Z}_p^+$ is a nonwitness, i.e., $b^{p-1} \equiv_p 1$, then $(a \times b \pmod{p})^{p-1} \not\equiv_p 1$ which implies that $a \times b \pmod{p}$ is witness
- If $b_1, b_2 \in \mathbb{Z}_p^+$ are nonwitnesses and $b_1 \neq b_2$, then $a \times b_1 \pmod{p}$ and $a \times b_2 \pmod{p}$ are two different witnesses

Primality

Lemma (Fermat test)

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

- Suppose p is not pseudoprime, there is $a \in \mathbb{Z}_p^+$, called witness, such that $a^{p-1} \not\equiv_p 1$
- If $b \in \mathbb{Z}_p^+$ is a nonwitness, i.e., $b^{p-1} \equiv_p 1$, then $(a \times b \pmod{p})^{p-1} \not\equiv_p 1$ which implies that $a \times b \pmod{p}$ is witness
- If $b_1, b_2 \in \mathbb{Z}_p^+$ are nonwitnesses and $b_1 \neq b_2$, then $a \times b_1 \pmod{p}$ and $a \times b_2 \pmod{p}$ are two different witnesses
- Therefore, the number of witnesses must be at least as large as the number of nonwitnesses in $\mathbb{Z}_p^+$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
2. Compute $a_i^{p-1} \pmod{p}$ for all $1 \leq i \leq k$ [Mod exponentiation in **P**]

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
2. Compute $a_i^{p-1} \pmod{p}$ for all $1 \leq i \leq k$ [Mod exponentiation in **P**]
3. If all computed values are 1, accept; otherwise, reject

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
2. Compute $a_i^{p-1} \pmod{p}$ for all $1 \leq i \leq k$ [Mod exponentiation in **P**]
3. If all computed values are 1, accept; otherwise, reject

- If p is pseudoprime, then $Pr[\text{PSEUDOPRIME accepts } p] = 1$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
2. Compute $a_i^{p-1} \pmod{p}$ for all $1 \leq i \leq k$ [Mod exponentiation in **P**]
3. If all computed values are 1, accept; otherwise, reject

- If p is pseudoprime, then $Pr[\text{PSEUDOPRIME accepts } p] = 1$
- If p is not pseudoprime, then $Pr[\text{PSEUDOPRIME accepts } p] \leq \frac{1}{2^k}$

Primality

Theorem (Fermat test)

If p is prime and $a \in \mathbb{Z}_p^+$, then $a^{p-1} \equiv_p 1$.

Lemma

If p is not pseudoprime, it fails the Fermat for at least half of all numbers in $\mathbb{Z}_p^+$, i.e., for each randomly selected $a \in \mathbb{Z}_p^+$, the probability of $a^{p-1} \equiv_p 1$ is at most $\frac{1}{2}$.

PSEUDOPRIME on input p:

1. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
2. Compute $a_i^{p-1} \pmod{p}$ for all $1 \leq i \leq k$ [Mod exponentiation in **P**]
3. If all computed values are 1, accept; otherwise, reject

- If p is pseudoprime, then $Pr[\text{PSEUDOPRIME accepts } p] = 1$
- If p is not pseudoprime, then $Pr[\text{PSEUDOPRIME accepts } p] \leq \frac{1}{2^k}$

How to convert PSEUDOPRIME to PRIME?

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$;

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

- if $\exists b \notin \{1, -1\}$ such that $b^2 \equiv_p 1$,

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

- if $\exists b \notin \{1, -1\}$ such that $b^2 \equiv_p 1$, then $(b' + ip)^2 \equiv_p 1$ for some $b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ and some integer i, i.e., $b = b' + ip$

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

- if $\exists b \notin \{1, -1\}$ such that $b^2 \equiv_p 1$, then $(b' + ip)^2 \equiv_p 1$ for some $b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ and some integer i, i.e., $b = b' + ip$
- then $\exists b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ such that $b'^2 \equiv_p 1$,

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

- if $\exists b \notin \{1, -1\}$ such that $b^2 \equiv_p 1$, then $(b' + ip)^2 \equiv_p 1$ for some $b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ and some integer i, i.e., $b = b' + ip$
- then $\exists b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ such that $b'^2 \equiv_p 1$, i.e., $(b' - 1) \times (b' + 1) = cp$ for some integer c

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, namely $x = 1$ and $x = -1$.

- if $\exists b \notin \{1, -1\}$ such that $b^2 \equiv_p 1$, then $(b' + ip)^2 \equiv_p 1$ for some $b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ and some integer i, i.e., $b = b' + ip$
- then $\exists b' \in \mathbb{Z}_p^+ \setminus \{1, -1\}$ such that $b'^2 \equiv_p 1$, i.e., $(b' - 1) \times (b' + 1) = cp$ for some integer c
- then $b' - 1, b' + 1 \in \mathbb{Z}_p^+$, but cp cannot be expressed by a product of two numbers that are smaller than it is, a contradiction

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, i.e., $x = \pm 1$.

Primality

How to convert PSEUDOPRIME to PRIME?

Observation

- If p is even, then p is prime if $p = 2$; not prime, otherwise
- It remains to consider non-PSEUDOPRIME odd numbers
- Miller-Rabin randomized primality test, an extension of Fermat test, but more effective than Fermat test

Theorem

If p is an odd prime, then $x^2 \equiv_p 1$ has only two solutions, i.e., $x = \pm 1$.

Corollary

If p is an odd number and there exists a nontrivial square root (i.e., not ± 1) of 1, modulo p, then p is composite number.

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject [$x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}$]
10. Accept if all the tests have been passed

- Line 8: Miller-Rabin test,

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject [$x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}$]
10. Accept if all the tests have been passed

- Line 8: Miller-Rabin test, $x_{j-1}^2 = x_j = 1$ and $x_{j-1} \notin \{1, -1\}$, then find a nontrivial square root of 1, modulo p, then p is composite

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject [$x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}$]
10. Accept if all the tests have been passed

- Line 8: Miller-Rabin test, $x_{j-1}^2 = x_j = 1$ and $x_{j-1} \notin \{1, -1\}$, then find a nontrivial square root of 1, modulo p, then p is composite
- Line 9: Fermat test, i.e., $a_i^{p-1} \not\equiv_p 1$, implies that p is not prime

Primality

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

- Line 8: Miller-Rabin test, $x_{j-1}^2 = x_j = 1$ and $x_{j-1} \notin \{1, -1\}$, then find a nontrivial square root of 1, modulo p, then p is composite
- Line 9: Fermat test, i.e., $a_i^{p-1} \not\equiv_p 1$, implies that p is not prime

Lemma

If PRIME rejects p, then p is not prime,

i.e., if p is prime, $Pr[\textit{PRIME accepts } p] = 1$

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

Lemma

If PRIME rejects p, then p is not prime,

i.e., if p is prime, $Pr[PRIME\ accepts\ p] = 1$

Lemma

If p is even, then p is prime iff $Pr[PRIME\ accepts\ p] = 1$,

p is not prime iff $Pr[PRIME\ rejects\ p] = 1$

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

Lemma

If PRIME rejects p, then p is not prime,

i.e., if p is prime, $Pr[\text{PRIME accepts } p] = 1$

Lemma

If p is even, then p is prime iff $Pr[\text{PRIME accepts } p] = 1$,

p is not prime iff $Pr[\text{PRIME rejects } p] = 1$

Lemma

If p is an odd composite number, $Pr[\text{PRIME accepts } p] \leq \frac{1}{2^k}$

Lemma

If p is an odd composite number, $Pr[PRIME \text{ accepts } p] \leq \frac{1}{2^k}$

Lemma

If p is an odd composite number, $Pr[PRIME \text{ accepts } p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p.

Lemma

If p is an odd composite number, $Pr[PRIME \text{ accepts } p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a,

Lemma

If p is an odd composite number, $Pr[PRIME \text{ accepts } p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$,

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1}$

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ *is at most* $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,
  $a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)}$

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,
  $a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)} = 1+pq^{e-1} + \sum_{k=2}^{p} C_p^k q^{k(e-1)} = 1+cp$

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,

$$a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)} = 1+pq^{e-1} + \sum_{k=2}^{p} C_p^k q^{k(e-1)} = 1+cp \equiv_p 1$$

Lemma

If p is an odd composite number, $Pr[PRIME \text{ accepts } p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,
  $a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)} = 1+pq^{e-1} + \sum_{k=2}^{p} C_p^k q^{k(e-1)} = 1+cp \equiv_p 1$
- Since $a^{p-1} \equiv_p 1$, and $a^p = a^{p-1}a \equiv_p a$

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,
  $$a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)} = 1+pq^{e-1} + \sum_{k=2}^{p} C_p^k q^{k(e-1)} = 1+cp \equiv_p 1$$
- Since $a^{p-1} \equiv_p 1$, and $a^p = a^{p-1}a \equiv_p a \not\equiv_p 1$,

Lemma

If p is an odd composite number, $Pr[PRIME\ accepts\ p] \leq \frac{1}{2^k}$

A witness a of p is a number such that PRIME rejects p. If a is not a witness of p, we say a is nonwitness of p, i.e., PRIME accepts p

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- Since p is composite
  - either p is q^e for some $e > 1$ such that $q \in \mathbb{Z}_p^+ \setminus \{1\}$ is a prime
  - p is $q \times r$ such that q and r two odd relative primes
- If $p = q^e$ for some $e > 1$, then let $a = 1 + q^{e-1} \in \mathbb{Z}_p^+ \setminus \{1\}$
- By Binomial Theorem,
  $a^p = (1+q^{e-1})^p = \sum_{k=0}^{p} C_p^k 1^{p-k} q^{k(e-1)} = 1+pq^{e-1} + \sum_{k=2}^{p} C_p^k q^{k(e-1)} = 1+cp \equiv_p 1$
- Since $a^{p-1} \equiv_p 1$, and $a^p = a^{p-1}a \equiv_p a \not\equiv_p 1$, a contradiction

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- Since $x_0 \cdots x_\ell$ are either all 1's or contains -1 at some position (it must exist, as s is odd), e.g., $(p-1)^s \equiv_p -1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- Since $x_0 \cdots x_\ell$ are either all 1's or contains -1 at some position (it must exist, as s is odd), e.g., $(p-1)^s \equiv_p -1$
- Let j be the largest position such that $x_j = -1$ for all possible nonwitnesses $h \in \mathbb{Z}_p^+$, i.e., $h^{s \times 2^j} \equiv_p -1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- Since $x_0 \cdots x_\ell$ are either all 1's or contains -1 at some position (it must exist, as s is odd), e.g., $(p-1)^s \equiv_p -1$
- Let j be the largest position such that $x_j = -1$ for all possible nonwitnesses $h \in \mathbb{Z}_p^+$, i.e., $h^{s \times 2^j} \equiv_p -1$
- Chinese remainder theorem: there is a one to one correspondence between $x \in \mathbb{Z}_{q \times r}$ and $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_r$ such that $x \equiv_q x_1$ and $x \equiv_r x_2$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- Since $x_0 \cdots x_\ell$ are either all 1's or contains -1 at some position (it must exist, as s is odd), e.g., $(p-1)^s \equiv_p -1$
- Let j be the largest position such that $x_j = -1$ for all possible nonwitnesses $h \in \mathbb{Z}_p^+$, i.e., $h^{s\times 2^j} \equiv_p -1$
- Chinese remainder theorem: there is a one to one correspondence between $x \in \mathbb{Z}_{q\times r}$ and $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_r$ such that $x \equiv_q x_1$ and $x \equiv_r x_2$
- By Chinese remainder theorem: $\exists t \in \mathbb{Z}_p^+$ such that $t \equiv_q h$ and $t \equiv_r 1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- Consider any nonwitness a, then, $a^{p-1} \equiv_p 1$, otherwise fails the Fermat test
- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- Since $x_0 \cdots x_\ell$ are either all 1's or contains -1 at some position (it must exist, as s is odd), e.g., $(p-1)^s \equiv_p -1$
- Let j be the largest position such that $x_j = -1$ for all possible nonwitnesses $h \in \mathbb{Z}_p^+$, i.e., $h^{s \times 2^j} \equiv_p -1$
- Chinese remainder theorem: there is a one to one correspondence between $x \in \mathbb{Z}_{q \times r}$ and $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_r$ such that $x \equiv_q x_1$ and $x \equiv_r x_2$
- By Chinese remainder theorem: $\exists t \in \mathbb{Z}_p^+$ such that $t \equiv_q h$ and $t \equiv_r 1 \Rightarrow t^{s \times 2^j} \equiv_q h^{s \times 2^j} \equiv_q -1$ and $t^{s \times 2^j} \equiv_r 1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$
- hence $t^{s\times 2^j} \not\equiv_p \pm 1$,
  - if $t^{s\times 2^{j+1}} \equiv_p 1$, then t is a witness

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$
- hence $t^{s\times 2^j} \not\equiv_p \pm 1$,
    - if $t^{s\times 2^{j+1}} \equiv_p 1$, then t is a witness
    - if $t^{s\times 2^{j+1}} \not\equiv_p 1$, contradicts that j is the largest position having -1

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ *is at most* $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$
- hence $t^{s\times 2^j} \not\equiv_p \pm 1$,
  - if $t^{s\times 2^{j+1}} \equiv_p 1$, then t is a witness
  - if $t^{s\times 2^{j+1}} \not\equiv_p 1$, contradicts that j is the largest position having -1
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$
- hence $t^{s\times 2^j} \not\equiv_p \pm 1$,
  - if $t^{s\times 2^{j+1}} \equiv_p 1$, then t is a witness
  - if $t^{s\times 2^{j+1}} \not\equiv_p 1$, contradicts that j is the largest position having -1
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e.,

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- p can be decomposed into $q \times r$ such that two odd numbers $q > 1$ and $r > 1$ are relative prime
- $\Rightarrow t^{s\times 2^j} \equiv_q h^{s\times 2^j} \equiv_q -1$ and $t^{s\times 2^j} \equiv_r 1$
- $t^{s\times 2^j} \equiv_q -1 \Rightarrow t^{s\times 2^j} \not\equiv_p 1$, since if $t^{s\times 2^j} \equiv_p 1$, then $t^{s\times 2^j} = 1 + cp = 1 + cqr \equiv_q 1$
- $t^{s\times 2^j} \equiv_r 1 \Rightarrow t^{s\times 2^j} \not\equiv_p -1$, since if $t^{s\times 2^j} \equiv_p -1$, then $t^{s\times 2^j} = -1 + cp = -1 + cqr \equiv_r -1$
- hence $t^{s\times 2^j} \not\equiv_p \pm 1$,
  - if $t^{s\times 2^{j+1}} \equiv_p 1$, then t is a witness
  - if $t^{s\times 2^{j+1}} \not\equiv_p 1$, contradicts that j is the largest position having -1
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s \times 2^j} \not\equiv_p \pm 1$, but $t^{s \times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod p$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod p = dt \pmod p$, i.e., $dt \pmod p$ is a unique witness

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$.

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness
  - Consider $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$: if $d't \pmod{p} = dt \pmod{p}$,

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness
  - Consider $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$: if $d't \pmod{p} = dt \pmod{p}$, then $d' = t^{s\times 2^{j+1}} d' \pmod{p}$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness
  - Consider $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$: if $d't \pmod{p} = dt \pmod{p}$, then $d' = t^{s\times 2^{j+1}} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d \pmod{p}$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness
  - Consider $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$: if $d't \pmod{p} = dt \pmod{p}$, then $d' = t^{s\times 2^{j+1}} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d \pmod{p} = t^{s\times 2^{j+1}} d \pmod{p}$

Lemma

If p is an odd composite number, the number of nonwitnesses of p in $\mathbb{Z}_p^+$ is at most $\frac{p-1}{2}$.

- hence $t^{s\times 2^j} \not\equiv_p \pm 1$, but $t^{s\times 2^{j+1}} \equiv_p 1$, i.e., t is a witness
- for each nonwitness $d \in \mathbb{Z}_p^+$, $dt \pmod{p}$ is a witness and does not exist a nonwitness $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$ and $d't \pmod{p} = dt \pmod{p}$, i.e., $dt \pmod{p}$ is a unique witness
  - Since j is the largest position such that $x_j = -1$, then $d^{s\times 2^j} \equiv_p \pm 1$ and $d^{s\times 2^{j+1}} \equiv_p 1$, moreover $t^{s\times 2^{j+1}} \equiv_p 1$. Hence $(dt)^{s\times 2^j} \not\equiv_p \pm 1$ and $dt^{s\times 2^{j+1}} \equiv_p 1$, $dt \pmod{p}$ is a witness
  - Consider $d' \in \mathbb{Z}_p^+$ such that $d \neq d'$: if $d't \pmod{p} = dt \pmod{p}$, then $d' = t^{s\times 2^{j+1}} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d' \pmod{p} = t \times t^{s\times 2^{j+1}-1} d \pmod{p} = t^{s\times 2^{j+1}} d \pmod{p} = d$, a contradiction

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ $[x_j = a_i^{s \times 2^j} \pmod{p}]$
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject $[x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}]$
10. Accept if all the tests have been passed

PRIME on input p:

1. If p is even: accept if $p = 2$, otherwise reject;
2. Select $a_1, \cdots, a_k$ randomly in $\mathbb{Z}_p^+$
3. For each $i = 1$ to k:
4. Let $p - 1 = s \times 2^\ell$ such that s is odd [Note $p - 1$ is even]
5. Let $x_0 = a_i^s \pmod{p}$
6. For each $j = 1$ to ℓ:
7. $x_j = x_{j-1}^2 \pmod{p}$ [$x_j = a_i^{s \times 2^j} \pmod{p}$]
8. If $x_j == 1 \wedge x_{j-1} \notin \{1, -1\}$, then reject
9. If $x_\ell \neq 1$, then reject [$x_\ell = a_i^{s \times 2^\ell} \pmod{p} = a_i^{p-1} \pmod{p}$]
10. Accept if all the tests have been passed

Theorem

If p is prime, $Pr[PRIME\ accepts\ p] = 1$.
If p is not prime, $Pr[PRIME\ accepts\ p] \leq 2^{-k}$,
i.e., $Pr[PRIME\ rejects\ p] \geq 1 - 2^{-k}$.

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

The probabilistic primality algorithm has one-sided error.

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

The probabilistic primality algorithm has one-sided error.

- When the algorithm outputs reject, we know that the input must be composite

PRIMES

Let
$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

The probabilistic primality algorithm has one-sided error.

- When the algorithm outputs reject, we know that the input must be composite
- When the output is accept, we know only that the input could be prime or composite

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

The probabilistic primality algorithm has one-sided error.

- When the algorithm outputs reject, we know that the input must be composite
- When the output is accept, we know only that the input could be prime or composite

Definition

RP is the class of languages that are decided by probabilistic polynomial time TM where inputs in the language are accepted with a probability of at least $\frac{1}{2}$, and inputs not in the language are rejected with a probability of 1.

PRIMES

Let

$$PRIMES = \{n \mid n \text{ is a prime number in binary}\}$$

Theorem

$PRIMES \in \mathbf{BPP}$

The probabilistic primality algorithm has one-sided error.

- When the algorithm outputs reject, we know that the input must be composite
- When the output is accept, we know only that the input could be prime or composite

Definition

RP is the class of languages that are decided by probabilistic polynomial time TM where inputs in the language are accepted with a probability of at least $\frac{1}{2}$, and inputs not in the language are rejected with a probability of 1.

Theorem

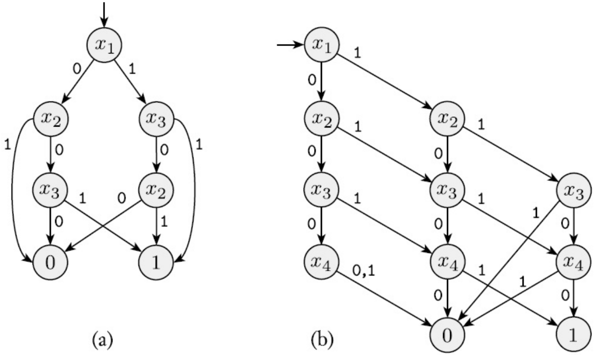
COMPOSITES $\in$ RP *and PRIME* $\in$ *co*RP

Branching Program

Definition

A branching program is a directed acyclic graph where all nodes are labeled by variables, except for two output nodes labeled 0 or 1.
The nodes that are labeled by variables are called query nodes. Every query node has two outgoing edges: one labeled 0 and the other labeled 1. Both output nodes have no outgoing edges. One of the nodes in a branching program is designated the start node.

$$f : \{0,1\}^V \to \{0,1\}$$



Branching Program

Two branching programs are equivalent if they determine the same Boolean function

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2\rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Theorem

*EQ_{BP} is co**NP**-complete.*

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Theorem

*EQ_{BP} is co**NP**-complete.*

- EQ_{BP} is in *co***NP**.

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Theorem

*EQ_{BP} is co**NP**-complete.*

- EQ_{BP} is in *co***NP**. $\overline{EQ}_{BP}$ is in **NP**. Nondeterminately select an assignment; accept if two programs evaluate to two different values on the assignment

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2\rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Theorem

*EQ_{BP} is co**NP**-complete.*

- EQ_{BP} is in *co***NP**. $\overline{EQ}_{BP}$ is in **NP**. Nondeterminately select an assignment; accept if two programs evaluate to two different values on the assignment
- *co***NP**-hardness.

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

Theorem

*EQ_{BP} is co**NP**-complete.*

- EQ_{BP} is in *co***NP**. $\overline{EQ}_{BP}$ is in **NP**. Nondeterminately select an assignment; accept if two programs evaluate to two different values on the assignment
- *co***NP**-hardness. $\overline{SAT} \leq_{\mathrm{P}} EQ_{BP}$. ϕ to P_1 and P_2 denotes the Boolean function that is always 0

Branching Program

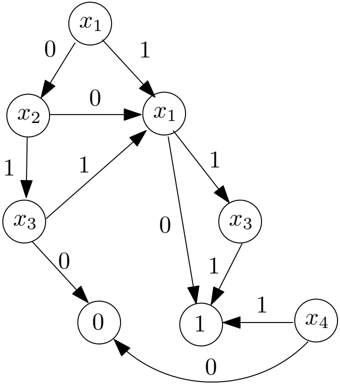
Two branching programs are equivalent if they determine the same Boolean function

Branching Program

Two branching programs are equivalent if they determine the same Boolean function

$$EQ_{BP} = \{\langle P_1, P_2\rangle \mid P_1, P_2 \text{ are equivalent branching programs}\}$$

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4)$$



Read-once Branching Program

A read-once branching program is one that can query each variable at most one time on every directed path from the start node to an output node.

Read-once Branching Program

A read-once branching program is one that can query each variable at most one time on every directed path from the start node to an output node.

$EQ_{ROBP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent read-once branching programs}\}$

Read-once Branching Program

A read-once branching program is one that can query each variable at most one time on every directed path from the start node to an output node.

$$EQ_{ROBP} = \{\langle P_1, P_2\rangle \mid P_1, P_2 \text{ are equivalent read-once branching programs}\}$$

Theorem

EQ_{BP} is in **BPP**.

Read-once Branching Program

A read-once branching program is one that can query each variable at most one time on every directed path from the start node to an output node.

$$EQ_{ROBP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent read-once branching programs}\}$$

Theorem

EQ_{BP} is in **BPP**.

1. No polynomial time algorithm is known for this problem, so it provides an example of probabilism apparently expanding the class of languages whereby membership can be tested efficiently.

Read-once Branching Program

A read-once branching program is one that can query each variable at most one time on every directed path from the start node to an output node.

$EQ_{ROBP} = \{\langle P_1, P_2 \rangle \mid P_1, P_2 \text{ are equivalent read-once branching programs}\}$

Theorem

EQ_{BP} is in **BPP**.

1. No polynomial time algorithm is known for this problem, so it provides an example of probabilism apparently expanding the class of languages whereby membership can be tested efficiently.
2. This algorithm introduces the technique of assigning non-Boolean values to normally Boolean variables in order to analyze the behavior of some Boolean function of those variables

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.
2. Accept if P_1 and P_2 agree on the assignment and reject otherwise.
3. However, there are 2^m number of assignments (m denotes the number of variables),

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.
2. Accept if P_1 and P_2 agree on the assignment and reject otherwise.
3. However, there are 2^m number of assignments (m denotes the number of variables), the probability that we would select that assignment is exponentially small.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.
2. Accept if P_1 and P_2 agree on the assignment and reject otherwise.
3. However, there are 2^m number of assignments (m denotes the number of variables), the probability that we would select that assignment is exponentially small.
4. Hence high probability even when P_1 and P_2 are not equivalent, and that is unsatisfactory.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.
2. Accept if P_1 and P_2 agree on the assignment and reject otherwise.
3. However, there are 2^m number of assignments (m denotes the number of variables), the probability that we would select that assignment is exponentially small.
4. Hence high probability even when P_1 and P_2 are not equivalent, and that is unsatisfactory.

Randomly selecting a non-Boolean assignment to the variables, and evaluate P_1 and P_2 in a suitably defined manner.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP***.*

Naive trial:

1. Randomly selects an assignment of variables and evaluate these branching programs on the assignment.
2. Accept if P_1 and P_2 agree on the assignment and reject otherwise.
3. However, there are 2^m number of assignments (m denotes the number of variables), the probability that we would select that assignment is exponentially small.
4. Hence high probability even when P_1 and P_2 are not equivalent, and that is unsatisfactory.

Randomly selecting a non-Boolean assignment to the variables, and evaluate P_1 and P_2 in a suitably defined manner. If P_1 and P_2 are not equivalent, the random evaluations will likely be unequal.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

D on input $\langle P_1, P_2 \rangle$, two read-once BP over variables $x_1, \cdots, x_m$:

1. Select elements a_1 through a_m at random from a finite field $\mathcal{F}$ with at least $3m$ elements.
2. Assign 1 to the start nodes.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

D on input $\langle P_1, P_2 \rangle$, two read-once BP over variables $x_1, \cdots, x_m$:

1. Select elements a_1 through a_m at random from a finite field $\mathcal{F}$ with at least $3m$ elements.
2. Assign 1 to the start nodes.
3. For BP P_i:

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

D on input $\langle P_1, P_2 \rangle$, two read-once BP over variables $x_1, \cdots, x_m$:

1. Select elements a_1 through a_m at random from a finite field $\mathcal{F}$ with at least $3m$ elements.
2. Assign 1 to the start nodes.
3. For BP P_i:
4. For each node v of P_i in topological order:
5. Assign $\sum_{(v',v)\in E_0}(1-a_j)p(v') + \sum_{(v',v)\in E_1} a_j p(v')$ to v, where a_j denotes the variable of v'

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

D on input $\langle P_1, P_2 \rangle$, two read-once BP over variables $x_1, \cdots, x_m$:

1. Select elements a_1 through a_m at random from a finite field $\mathcal{F}$ with at least $3m$ elements.
2. Assign 1 to the start nodes.
3. For BP P_i:
4. For each node v of P_i in topological order:
5. Assign $\sum_{(v',v) \in E_0} (1 - a_j) p(v') + \sum_{(v',v) \in E_1} a_j p(v')$ to v, where a_j denotes the variable of v'
6. Accept if the assigned values of 1-labeled output nodes in P_1 and P_2 are identical; otherwise reject.

Read-once Branching Program

Theorem

EQ_{ROBP} is in **BPP**.

D on input $\langle P_1, P_2 \rangle$, two read-once BP over variables $x_1, \cdots, x_m$:

1. Select elements a_1 through a_m at random from a finite field $\mathcal{F}$ with at least $3m$ elements.
2. Assign 1 to the start nodes.
3. For BP P_i:
4. For each node v of P_i in topological order:
5. Assign $\sum_{(v',v)\in E_0}(1-a_j)p(v') + \sum_{(v',v)\in E_1} a_j p(v')$ to v, where a_j denotes the variable of v'
6. Accept if the assigned values of 1-labeled output nodes in P_1 and P_2 are identical; otherwise reject.

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

where y_j^i is either x_j, $(1 - x_j)$ or 1

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

  where y_j^i is either x_j, $(1 - x_j)$ or 1

- For each y_j^i that is 1, it can be replaced by $x_j + (1 - x_j)$, then the product term $y_1^i y_2^i \cdots y_m^i$ is split into two product terms: $(y_1^i y_2^i \cdots y_m^i)[y_j^i \mapsto x_j]$ and $(y_1^i y_2^i \cdots y_m^i)[y_j^i \mapsto (1 - x_j)]$
- For each i, the product term $y_1^i y_2^i \cdots y_m^i$ corresponds to a path in P from the start node to the 1-labeled output node

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

  where y_j^i is either x_j, $(1 - x_j)$ or 1
- For each y_j^i that is 1, it can be replaced by $x_j + (1 - x_j)$, then the product term $y_1^i y_2^i \cdots y_m^i$ is split into two product terms: $(y_1^i y_2^i \cdots y_m^i)[y_j^i \mapsto x_j]$ and $(y_1^i y_2^i \cdots y_m^i)[y_j^i \mapsto (1 - x_j)]$
- For each i, the product term $y_1^i y_2^i \cdots y_m^i$ corresponds to a path in P from the start node to the 1-labeled output node
- We assume that the polynomials can transformed into this form, i.e., no y_j^i is 1.

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

- If P_1 and P_2 are equivalent,

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

- If P_1 and P_2 are equivalent, then the polynomials of P_1 and P_2 contains identical product terms,

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

- If P_1 and P_2 are equivalent, then the polynomials of P_1 and P_2 contains identical product terms, hence D always accepts

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

- If P_1 and P_2 are equivalent, then the polynomials of P_1 and P_2 contains identical product terms, hence D always accepts
- If P_1 and P_2 are not equivalent,

Read-once Branching Program

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

- The assigned value of 1-labeled output node can be seen as a polynomial:

$$\sum_{i=1}^{n} y_1^i y_2^i \cdots y_m^i$$

- If P_1 and P_2 are equivalent, then the polynomials of P_1 and P_2 contains identical product terms, hence D always accepts
- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.
- Induction step $d \geq 1$.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.
- Induction step $d \geq 1$. If p is non-zero polynomial with a root at a,

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.
- Induction step $d \geq 1$. If p is non-zero polynomial with a root at a, then $\frac{p}{(x-a)}$ is a non-zero degree-$(d-1)$ polynomial.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.
- Induction step $d \geq 1$. If p is non-zero polynomial with a root at a, then $\frac{p}{(x-a)}$ is a non-zero degree-$(d-1)$ polynomial.By the induction hypothesis, $\frac{p}{(x-a)}$ has at most $(d-1)$ roots.

Read-once Branching Program

- If P_1 and P_2 are not equivalent, then hence D rejects with a probability of at least $\frac{2}{3}$.

Lemma

For every $d \geq 0$, a degree-d polynomial p on a single variable x either has at most d roots, or is every where equal to 0

By induction on d:

- Base step $d = 0$. Then p is a constant. If that constant is non-zero, the polynomial clearly has no roots.
- Induction step $d \geq 1$. If p is non-zero polynomial with a root at a, then $\frac{p}{(x-a)}$ is a non-zero degree-$(d-1)$ polynomial.By the induction hypothesis, $\frac{p}{(x-a)}$ has at most $(d-1)$ roots. Therefore, p has at most d roots.

Read-once Branching Program

Lemma

Let $\mathcal{F}$ be a finite field with $f > 0$ elements, and p be a non-zero polynomial on variables $x_1, \cdots, x_m$, where each variable has degree at most d. If $a_1, \cdots, a_m$ are selected randomly from $\mathcal{F}$, then

$$Pr[p(a_1, \cdots, a_m) = 0] \leq \frac{md}{f}$$

By induction on m:

Read-once Branching Program

Lemma

Let $\mathcal{F}$ be a finite field with $f > 0$ elements, and p be a non-zero polynomial on variables $x_1, \cdots, x_m$, where each variable has degree at most d. If $a_1, \cdots, a_m$ are selected randomly from $\mathcal{F}$, then

$$Pr[p(a_1, \cdots, a_m) = 0] \leq \frac{md}{f}$$

By induction on m:

- Base step $m = 1$.

Read-once Branching Program

Lemma

Let $\mathcal{F}$ be a finite field with $f > 0$ elements, and p be a non-zero polynomial on variables $x_1, \cdots, x_m$, where each variable has degree at most d. If $a_1, \cdots, a_m$ are selected randomly from $\mathcal{F}$, then

$$Pr[p(a_1, \cdots, a_m) = 0] \leq \frac{md}{f}$$

By induction on m:

- Base step $m = 1$. By previously lemma, p has at most d roots.

Read-once Branching Program

- Induction step $m > 1$.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$
- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.By the induction hypothesis,

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then

$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.By the induction hypothesis, the probability that p_j evaluates to 0 is at most $\frac{(m-1)d}{f}$ (p_j has at most $(m-1)$ variables)

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then

$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.By the induction hypothesis, the probability that p_j evaluates to 0 is at most $\frac{(m-1)d}{f}$ (p_j has at most $(m-1)$ variables)
  - Or some p_i does not evaluate to 0, and a_1 is a root of the single variable polynomial p_x obtained by evaluating $p_0, \cdots, p_d$ on $a_2, \cdots, a_m$.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$

- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.By the induction hypothesis, the probability that p_j evaluates to 0 is at most $\frac{(m-1)d}{f}$ (p_j has at most $(m-1)$ variables)
  - Or some p_i does not evaluate to 0, and a_1 is a root of the single variable polynomial p_x obtained by evaluating $p_0, \cdots, p_d$ on $a_2, \cdots, a_m$. The probability that p_x evaluates to 0 is at most $\frac{d}{f}$.

Read-once Branching Program

- Induction step $m > 1$. For each $i \leq d$, let p_i be the polynomial comprising the terms of p containing x_1^i, but x_1^i has been factored out. Then
$$p = p_0 + x_1 p_1 + x_1^2 p_2 + \cdots + x_1^d p_d$$
- If $p(a_1, \cdots, a_m) = 0$, then
  - Either all p_i evaluate to 0. At least one p_j is nonzero, as $p \neq 0$. The probability that all p_i evaluate to 0 is at most the probability that p_j evaluates to 0.By the induction hypothesis, the probability that p_j evaluates to 0 is at most $\frac{(m-1)d}{f}$ (p_j has at most $(m-1)$ variables)
  - Or some p_i does not evaluate to 0, and a_1 is a root of the single variable polynomial p_x obtained by evaluating $p_0, \cdots, p_d$ on $a_2, \cdots, a_m$. The probability that p_x evaluates to 0 is at most $\frac{d}{f}$.
  - Then, the probability that $a_1, \cdots, a_m$ is a root of p is at most $\frac{(m-1)d}{f} + \frac{d}{f} = \frac{md}{f}$

Read-once Branching Program

Lemma

Let $\mathcal{F}$ be a finite field with $f > 0$ elements, and p be a non-zero polynomial on variables $x_1, \cdots, x_m$, where each variable has degree at most d. If $a_1, \cdots, a_m$ are selected randomly from $\mathcal{F}$, then

$$Pr[p(a_1, \cdots, a_m) = 0] \leq \frac{md}{f}$$

Read-once Branching Program

Lemma

Let $\mathcal{F}$ be a finite field with $f > 0$ elements, and p be a non-zero polynomial on variables $x_1, \cdots, x_m$, where each variable has degree at most d. If $a_1, \cdots, a_m$ are selected randomly from $\mathcal{F}$, then

$$Pr[p(a_1, \cdots, a_m) = 0] \leq \frac{md}{f}$$

Since $f = 3m$ and $d = 1$, we have:

Theorem

This algorithm runs in polynomial time and decides EQ_{ROBP} with an error probability of at most $\frac{1}{3}$.

Outline

Alternation

- Alternation is a generalization of nondeterminism

Alternation

- Alternation is a generalization of nondeterminism
- An alternating algorithm may contain instructions to branch a process into multiple child processes, just as in a nondeterministic algorithm

Alternation

- Alternation is a generalization of nondeterminism
- An alternating algorithm may contain instructions to branch a process into multiple child processes, just as in a nondeterministic algorithm
  - A nondeterministic computation accepts if any one of the branching accepts, seen as OR-tree

Alternation

- Alternation is a generalization of nondeterminism
- An alternating algorithm may contain instructions to branch a process into multiple child processes, just as in a nondeterministic algorithm
  - A nondeterministic computation accepts if any one of the branching accepts, seen as OR-tree
  - An alternating computation has two designated cases:

Alternation

- Alternation is a generalization of nondeterminism
- An alternating algorithm may contain instructions to branch a process into multiple child processes, just as in a nondeterministic algorithm
  - A nondeterministic computation accepts if any one of the branching accepts, seen as OR-tree
  - An alternating computation has two designated cases: accepts if any of the children branching accepts, or accepts if all of the children branchings, seen as AND/OR-tree accept.

Alternation

- Alternation is a generalization of nondeterminism
- An alternating algorithm may contain instructions to branch a process into multiple child processes, just as in a nondeterministic algorithm
  - A nondeterministic computation accepts if any one of the branching accepts, seen as OR-tree
  - An alternating computation has two designated cases: accepts if any of the children branching accepts, or accepts if all of the children branchings, seen as AND/OR-tree accept.
- Using alternation, we may simplify various proofs in time/space complexity theory and exhibit a surprising connection between the time and space complexity measures

Alternating TM

Definition

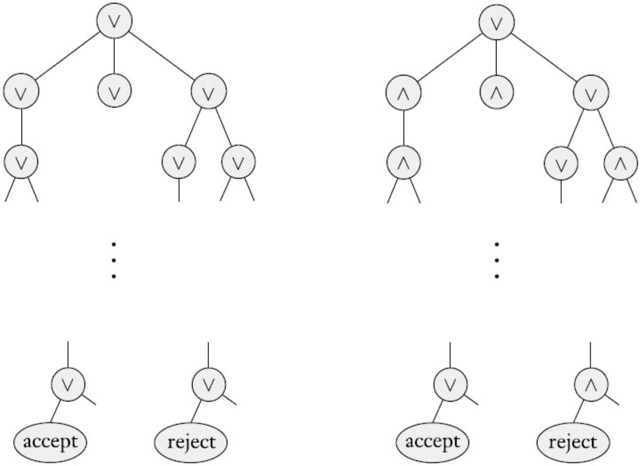
A alternating TM is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{rejcet}})$, where

1. Q is finite set of states such that $Q \setminus \{q_{\text{accept}}, q_{\text{rejcet}}\} = Q_\exists \uplus Q_\forall$
2. Σ is a finite nonempty input alphabet not containing the blank symbol $\sqcup$,
3. Γ is finite nonempty tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. δ is a transition function

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \{L, R\}).$$

5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Alternating TM



nondeterministic alternating

Alternating Time and Space

Time and space complexity of these machines in the same way that we did for nondeterministic Turing machines: by taking the maximum time or space used by any computation branch.

Alternating Time and Space

Time and space complexity of these machines in the same way that we did for nondeterministic Turing machines: by taking the maximum time or space used by any computation branch.

$$\textbf{ATIME}(t(n)) = \{L \mid L \text{ is decided by an } O(t(n)) \text{ time alternating TM}\}$$

$$\textbf{ASPACE}(t(n)) = \{L \mid L \text{ is decided by an } O(t(n)) \text{ space alternating TM}\}$$

Alternating Time and Space

Time and space complexity of these machines in the same way that we did for nondeterministic Turing machines: by taking the maximum time or space used by any computation branch.

$$\mathbf{ATIME}(t(n)) = \{L \mid L \text{ is decided by an } O(t(n)) \text{ time alternating TM}\}$$

$$\mathbf{ASPACE}(t(n)) = \{L \mid L \text{ is decided by an } O(t(n)) \text{ space alternating TM}\}$$

$$\mathbf{AP} = \bigcup_k \mathbf{ATIME}(n^k)$$

$$\mathbf{APSPACE} = \bigcup_k \mathbf{ASPACE}(n^k)$$

$$\mathbf{AL} = \mathbf{ASPACE}(\log n)\}$$

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables,

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables, i.e., $\neg\phi$ is unsatisfiable

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables, i.e., $\neg\phi$ is unsatisfiable

$$TAUT = \{\langle\phi\rangle \mid \phi \text{ is a tautology}\}$$

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables, i.e., $\neg\phi$ is unsatisfiable

$$TAUT = \{\langle\phi\rangle \mid \phi \text{ is a tautology}\}$$

Theorem

TAUT is in **AP**.

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables, i.e., $\neg\phi$ is unsatisfiable

$$TAUT = \{\langle\phi\rangle \mid \phi \text{ is a tautology}\}$$

Theorem

TAUT is in **AP**.

M on input $\langle\phi\rangle$:

1. Universally select all assignments to the variables of ϕ
2. For each selected assignment, evaluate ϕ
3. If ϕ evaluates to 1, accept; otherwise reject

Tautology

A Boolean formula ϕ is a tautology if it evaluates to 1 on every assignment to its variables, i.e., $\neg\phi$ is unsatisfiable

$$TAUT = \{\langle\phi\rangle \mid \phi \text{ is a tautology}\}$$

Theorem
TAUT is in **AP**.

M on input $\langle\phi\rangle$:

1. Universally select all assignments to the variables of ϕ
2. For each selected assignment, evaluate ϕ
3. If ϕ evaluates to 1, accept; otherwise reject

Theorem
*co***NP** $\subseteq$ **AP**.

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function).

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle\phi\rangle \mid \phi \text{ is a minimal formula}\}$$

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle \phi \rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle\phi\rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

M on input $\langle\phi\rangle$:

1. Universally select all formulas ψ that are shorter than of ϕ

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle \phi \rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

M on input $\langle \phi \rangle$:

1. Universally select all formulas ψ that are shorter than of ϕ
2. Existentially select an assignment to the variables of ϕ

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle \phi \rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

M on input $\langle \phi \rangle$:

1. Universally select all formulas ψ that are shorter than of ϕ
2. Existentially select an assignment to the variables of ϕ
3. Evaluate ϕ and ψ on this assignment

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle \phi \rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

M on input $\langle \phi \rangle$:

1. Universally select all formulas ψ that are shorter than of ϕ
2. Existentially select an assignment to the variables of ϕ
3. Evaluate ϕ and ψ on this assignment
4. If ϕ and ψ evaluate to the same value, reject; otherwise accept

MIN-Formula

Two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it.

$$\text{MIN-FORMULA} = \{\langle \phi \rangle \mid \phi \text{ is a minimal formula}\}$$

Theorem

MIN-FORMULA is in **AP**.

M on input $\langle \phi \rangle$:

1. Universally select all formulas ψ that are shorter than of ϕ
2. Existentially select an assignment to the variables of ϕ
3. Evaluate ϕ and ψ on this assignment
4. If ϕ and ψ evaluate to the same value, reject; otherwise accept

It is not known whether MIN-FORMULA is in **NP** or in co**NP**

Connection between the time and space complexity

Theorem

$$\forall f(n) \geq n, \textbf{ATIME}(f(n)) \subseteq \text{SPACE}(f(n)) \subseteq \textbf{ATIME}(f^2(n))$$

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) = \text{TIME}(2^{O(f(n))})$$

Connection between the time and space complexity

Theorem

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) = \mathrm{TIME}(2^{O(f(n))})$$

Corollary

AL=**P**, **AP**=PSPACE *and* **APSPACE**=EXPTIME

Connection between the time and space complexity

Theorem

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$
$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) = \mathrm{TIME}(2^{O(f(n))})$$

Corollary

AL=**P**, **AP**=PSPACE *and* **APSPACE**=EXPTIME

Open problems: $\mathbf{NP} = \mathbf{AP}$? and $\mathbf{P} = \mathbf{AP}$?

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM,

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. M_d performs a depth-first search of M_a's computation tree to determine which nodes in the tree are accepting

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. M_d performs a depth-first search of M_a's computation tree to determine which nodes in the tree are accepting
2. M_d accepts if it determines that the root of the tree, corresponding to M_a's starting configuration, is accepting

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. M_d performs a depth-first search of M_a's computation tree to determine which nodes in the tree are accepting
2. M_d accepts if it determines that the root of the tree, corresponding to M_a's starting configuration, is accepting
3. The depth of M_a's computation tree is at most $O(f(n))$,

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. M_d performs a depth-first search of M_a's computation tree to determine which nodes in the tree are accepting
2. M_d accepts if it determines that the root of the tree, corresponding to M_a's starting configuration, is accepting
3. The depth of M_a's computation tree is at most $O(f(n))$, and during each recursion M_d only need to store the nondeterministic choice instead of the configuration

Connection between the time and space complexity

$$\forall f(n) \geq n, \textbf{ATIME}(f(n)) \subseteq \text{SPACE}(f(n))$$

Let M_a be a $f(n)$ time alternating TM, we construct a $f(n)$ space deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. M_d performs a depth-first search of M_a's computation tree to determine which nodes in the tree are accepting
2. M_d accepts if it determines that the root of the tree, corresponding to M_a's starting configuration, is accepting
3. The depth of M_a's computation tree is at most $O(f(n))$, and during each recursion M_d only need to store the nondeterministic choice instead of the configuration
4. The total space is $O(f(n))$, but the total time is $O(2^{f(n)})$

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathsf{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM,

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

Connection between the time and space complexity

$$\forall f(n) \geq n, \text{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.

Connection between the time and space complexity

$$\forall f(n) \geq n, \text{SPACE}(f(n)) \subseteq \textbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.
2. If $t > 1$,
3. Existentially to guess a configuration c_m of M_d [$O(f(n))$ time]

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.
2. If $t > 1$,
3. Existentially to guess a configuration c_m of M_d [$O(f(n))$ time]
4. Universally CANYIELD$(c_1, c_m, t/2)$ and CANYIELD$(c_m, c_2, t/2)$

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathsf{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.
2. If $t > 1$,
3. Existentially to guess a configuration c_m of M_d [$O(f(n))$ time]
4. Universally CANYIELD$(c_1, c_m, t/2)$ and CANYIELD$(c_m, c_2, t/2)$
5. If both are accepted, then accept; otherwise reject.

Connection between the time and space complexity

$$\forall f(n) \geq n, \text{SPACE}(f(n)) \subseteq \textbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.
2. If $t > 1$,
3. Existentially to guess a configuration c_m of M_d [$O(f(n))$ time]
4. Universally CANYIELD$(c_1, c_m, t/2)$ and CANYIELD$(c_m, c_2, t/2)$
5. If both are accepted, then accept; otherwise reject.

M_a on input w:

1. CANYIELD$(c_{start}, c_{accept}, 2^{df(n)})$, where d is selected such that M has no more than $2^{df(n)}$ configurations within its space bound

Connection between the time and space complexity

$$\forall f(n) \geq n, \mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$$

Let M_d be a $f(n)$ space deterministic TM, we construct a $f^2(n)$ time alternating TM M_a that simulates M_d as follows:

CANYIELD on input c_1, c_2, and t:

1. If $t = 1$, then test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of M_d. Accept if either test succeeds; reject if both fail.
2. If $t > 1$,
3. Existentially to guess a configuration c_m of M_d [$O(f(n))$ time]
4. Universally CANYIELD$(c_1, c_m, t/2)$ and CANYIELD$(c_m, c_2, t/2)$
5. If both are accepted, then accept; otherwise reject.

M_a on input w:

1. CANYIELD$(c_{start}, c_{accept}, 2^{df(n)})$, where d is selected such that M has no more than $2^{df(n)}$ configurations within its space bound

M_a uses at most $O(f(n)) \times \log 2^{df(n)} = O(f^2(n))$ time, space is $O(f(n))$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM,

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
    - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
    - (c, c') is an edge iff M_a can move from c to c' in one step

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
    - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
    - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
    - Mask a node if it is universal configuration and all its successors are masked

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

- The number of nodes is at most $2^{O(f(n))}$,

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

- The number of nodes is at most $2^{O(f(n))}$, hence the size of the graph

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

- The number of nodes is at most $2^{O(f(n))}$, hence the size of the graph
- Thus, the graph can be constructed in $2^{O(f(n))}$ time

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

- The number of nodes is at most $2^{O(f(n))}$, hence the size of the graph
- Thus, the graph can be constructed in $2^{O(f(n))}$ time
- It scans at most $2^{O(f(n))}$ times and each scan is in $2^{O(f(n))}$ time

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))})$$

Let M_a be a $O(f(n))$ space alternating TM, we construct a $2^{O(f(n))}$ time deterministic TM M_d that simulates M_a as follows:

M_d on input w:

1. Construct a directed graph:
   - Nodes are configurations of M_a on w that use at most $d \times f(n)$ space
   - (c, c') is an edge iff M_a can move from c to c' in one step
2. Mark all the accepting configurations in the graph
3. Repeatedly scans all the nodes in the graph until no update:
   - Mask a node if it is universal configuration and all its successors are masked
   - Mask a node if it is existential configuration and one of its successors is masked

- The number of nodes is at most $2^{O(f(n))}$, hence the size of the graph
- Thus, the graph can be constructed in $2^{O(f(n))}$ time
- It scans at most $2^{O(f(n))}$ times and each scan is in $2^{O(f(n))}$ time
- Total: $2^{O(f(n))}$ time, and also space

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \supseteq \text{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{O(f(n))}$ time deterministic TM,

Connection between the time and space complexity

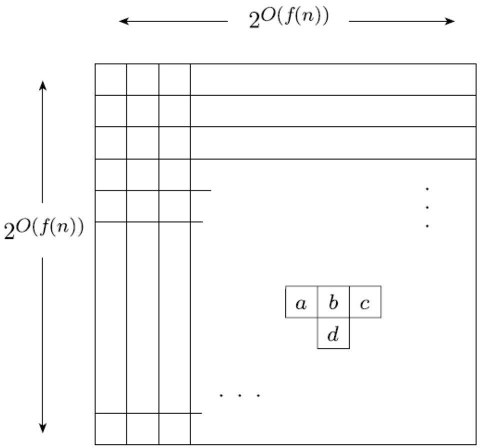
$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{O(f(n))}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \supseteq \text{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{O(f(n))}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:



Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \supseteq \text{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write $\sqcup$ on the tape on acceptance

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write ␣ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write ␣ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.
2. Otherwise, existentially guesses three symbols s_1, s_2, s_2

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write ␣ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.
2. Otherwise, existentially guesses three symbols s_1, s_2, s_2
3. Verify whether $cell[i-1, j-1, s_1], cell[i-1, j, s_2], cell[i-1, j+1, s_3]$ can yield $cell[i, j, s]$ in M_d's transition function

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \mathrm{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write ␣ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.
2. Otherwise, existentially guesses three symbols s_1, s_2, s_2
3. Verify whether $cell[i-1, j-1, s_1], cell[i-1, j, s_2], cell[i-1, j+1, s_3]$ can yield $cell[i, j, s]$ in M_d's transition function
4. If yes, then universally run $CellCheck(i-1, j-1, s_1)$, $CellCheck(i-1, j, s_2)$ and $CellCheck(i-1, j+1, s_3)$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \textbf{ASPACE}(f(n)) \supseteq \text{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write $\sqcup$ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.
2. Otherwise, existentially guesses three symbols s_1, s_2, s_2
3. Verify whether $cell[i-1, j-1, s_1], cell[i-1, j, s_2], cell[i-1, j+1, s_3]$ can yield $cell[i, j, s]$ in M_d's transition function
4. If yes, then universally run $CellCheck(i-1, j-1, s_1)$, $CellCheck(i-1, j, s_2)$ and $CellCheck(i-1, j+1, s_3)$

M_d on input w:

1. Universally $CellCheck(2^{k \cdot f(n)}, 0, q_{accept})$, $CellCheck(2^{k \cdot f(n)}, 1, \sqcup)$, $\cdots$ $CellCheck(2^{k \cdot f(n)}, 2^{k \cdot f(n)}, \sqcup)$

Connection between the time and space complexity

$$\forall f(n) \geq \log n, \mathbf{ASPACE}(f(n)) \supseteq \text{TIME}(2^{O(f(n))})$$

Let M_d be a $2^{k \cdot f(n)}$ time deterministic TM, we construct a $O(f(n))$ space alternating TM M_a that simulates M_d as follows:
Assume that M_d moves its head to the left-hand end of the tape and write ␣ on the tape on acceptance

CellCheck on i, j, s:

1. If $i = 0$, then accept if $s = w_j$, otherwise reject.
2. Otherwise, existentially guesses three symbols s_1, s_2, s_2
3. Verify whether $cell[i-1, j-1, s_1], cell[i-1, j, s_2], cell[i-1, j+1, s_3]$ can yield $cell[i, j, s]$ in M_d's transition function
4. If yes, then universally run $CellCheck(i-1, j-1, s_1)$, $CellCheck(i-1, j, s_2)$ and $CellCheck(i-1, j+1, s_3)$

M_d on input w:

1. Universally $CellCheck(2^{k \cdot f(n)}, 0, q_{accept})$, $CellCheck(2^{k \cdot f(n)}, 1, ␣)$, $\cdots$ $CellCheck(2^{k \cdot f(n)}, 2^{k \cdot f(n)}, ␣)$

We only need to store i, j, s which only need space $O(f(n))$ using binary representation, time is still $O(2^{f(n)})$

Understanding Proofs

Target	Proof	Other side
$\mathbf{ATIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$	post-order tree travseral tree depth$=O(f(n))$	$\mathrm{TIME}(2^{f(n)})$
$\mathrm{SPACE}(f(n)) \subseteq \mathbf{ATIME}(f^2(n))$	CANYIELD	$\mathbf{ASPACE}(f(n))$
$\mathbf{ASPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$	graph travseral graph size$=O(2^{f(n)})$	$\mathrm{SPACE}(2^{f(n)})$
$\mathrm{TIME}(2^{O(f(n))}) \subseteq \mathbf{ASPACE}(f(n))$	*CellCheck*	$\mathbf{ATIME}(2^{f(n)})$

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Σ_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \cdots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

where $Q_i = \forall$ if i is even, otherwise $Q_i = \exists$.

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Σ_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$

where $Q_i = \forall$ if i is even, otherwise $Q_i = \exists$.

Definition

For every $i \geq 1$, a Σ_i-alternating TM is an alternating TM such that the initial state is in $Q_\exists$ and for every input and on every computation branching starting from the starting configuration, M can alternate at most $i - 1$ times, i.e., in total at most i universal and existential steps.

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Σ_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

where $Q_i = \forall$ if i is even, otherwise $Q_i = \exists$.

Definition

For every $i \geq 1$, a Σ_i-alternating TM is an alternating TM such that the initial state is in $Q_\exists$ and for every input and on every computation branching starting from the starting configuration, M can alternate at most $i-1$ times, i.e., in total at most i universal and existential steps.

Theorem

$L \in \Sigma_i^p$ iff there is a Σ_i-alternating TM M such that M can decide L in polynomial time

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Π_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

where $Q_i = \exists$ if i is even, otherwise $Q_i = \forall$.

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Π_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

where $Q_i = \exists$ if i is even, otherwise $Q_i = \forall$.

Definition

For every $i \geq 1$, a Π_i-alternating TM is an alternating TM such that the initial state is in $Q_\forall$ and for every input and on every computation branching starting from the starting configuration, M can alternate at most $i - 1$ times, i.e., in total at most i universal and existential steps.

Polynomial Hierarchy

Definition

For every $i \geq 1$ a language L is Π_i^p if there is a polynomial time TM M and a polynomial time computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

where $Q_i = \exists$ if i is even, otherwise $Q_i = \forall$.

Definition

For every $i \geq 1$, a Π_i-alternating TM is an alternating TM such that the initial state is in $Q_\forall$ and for every input and on every computation branching starting from the starting configuration, M can alternate at most $i-1$ times, i.e., in total at most i universal and existential steps.

Theorem

$L \in \Pi_i^p$ iff there is a Π_i-alternating TM M such that M can decide L in polynomial time

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT
- $\Pi_1^p = co\mathbf{NP}$, e.g., Tautology

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT
- $\Pi_1^p = co\mathbf{NP}$, e.g., Tautology
- For every $i \geq 1$, $\Pi_i^p = co\Sigma_i^p$ and $co\Pi_i^p = \Sigma_i^p$

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT
- $\Pi_1^p = co\mathbf{NP}$, e.g., Tautology
- For every $i \geq 1$, $\Pi_i^p = co\Sigma_i^p$ and $co\Pi_i^p = \Sigma_i^p$
- MIN-FORMULA$\in \Pi_2^p$

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT
- $\Pi_1^p = co\mathbf{NP}$, e.g., Tautology
- For every $i \geq 1$, $\Pi_i^p = co\Sigma_i^p$ and $co\Pi_i^p = \Sigma_i^p$
- MIN-FORMULA$\in \Pi_2^p$
- $\mathbf{PH} = \bigcup_{i \geq 1} \Pi_i^p \subseteq \text{PSPACE}$

Some Relations Involving PH

- $\Sigma_1^p = \mathbf{NP}$, e.g., SAT
- $\Pi_1^p = co\mathbf{NP}$, e.g., Tautology
- For every $i \geq 1$, $\Pi_i^p = co\Sigma_i^p$ and $co\Pi_i^p = \Sigma_i^p$
- MIN-FORMULA$\in \Pi_2^p$
- $\mathbf{PH} = \bigcup_{i \geq 1} \Pi_i^p \subseteq$ PSPACE
- Open problem $\mathbf{PH} =$ PSPACE

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:
  - $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:
  - $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
  - $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$

Some Relations Involving Oracle TMs

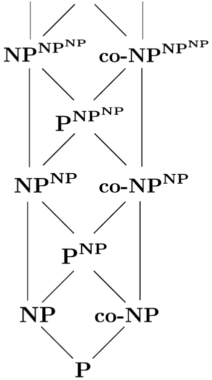
- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:
  - $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
  - $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
  - $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:
  - $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
  - $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
  - $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $\Sigma_i^p \subseteq \Delta_{i+1}^p \subseteq \Sigma_{i+1}^p$
- $\Pi_i^p \subseteq \Delta_{i+1}^p \subseteq \Pi_{i+1}^p$

Some Relations Involving Oracle TMs

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}$
- For every $i \geq 0$:
  - $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
  - $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
  - $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $\Sigma_i^p \subseteq \Delta_{i+1}^p \subseteq \Sigma_{i+1}^p$
- $\Pi_i^p \subseteq \Delta_{i+1}^p \subseteq \Pi_{i+1}^p$



Properties of the Polynomial Hierarchy

- If $\mathbf{P} \neq \mathbf{NP}$ and $\mathbf{NP} \neq co\mathbf{NP}$, then $\Sigma_i^p \subsetneq \Sigma_{i+1}^p$

Properties of the Polynomial Hierarchy

- If $\mathbf{P} \neq \mathbf{NP}$ and $\mathbf{NP} \neq co\mathbf{NP}$, then $\Sigma_i^p \subsetneq \Sigma_{i+1}^p$
- This conjecture is stated as the polynomial hierarchy does not collapse

Properties of the Polynomial Hierarchy

- If $\mathbf{P} \neq \mathbf{NP}$ and $\mathbf{NP} \neq co\mathbf{NP}$, then $\Sigma_i^p \subsetneq \Sigma_{i+1}^p$
- This conjecture is stated as the polynomial hierarchy does not collapse
- The polynomial hierarchy is said to collapse if there is some $i \geq 1$, $\Sigma_i^p = \Sigma_{i+1}^p$

Properties of the Polynomial Hierarchy

- If $\mathbf{P} \neq \mathbf{NP}$ and $\mathbf{NP} \neq co\mathbf{NP}$, then $\Sigma_i^p \subsetneq \Sigma_{i+1}^p$
- This conjecture is stated as the polynomial hierarchy does not collapse
- The polynomial hierarchy is said to collapse if there is some $i \geq 1$, $\Sigma_i^p = \Sigma_{i+1}^p$

Theorem

If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{PH} = \mathbf{P}$, i.e., the polynomial hierarchy collapses to $\mathbf{P}$.

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{PH} = \mathbf{P}$, *i.e., the polynomial hierarchy collapses to* $\mathbf{P}$.

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$.

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{PH} = \mathbf{P}$, *i.e., the polynomial hierarchy collapses to* $\mathbf{P}$.

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$.

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{PH} = \mathbf{P}$, *i.e., the polynomial hierarchy collapses to* $\mathbf{P}$.

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$
- Define a new language L' as
$$\langle x, u_1 \rangle \in L' \iff \forall u_2 \in \{0,1\}^{q(|x|)} \dots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

- Define a new language L' as

$$\begin{aligned} \langle x, u_1 \rangle \in L' \quad \iff \quad & \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$

- Then $L' \in \Pi_{i-1}^p$.

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$
- Define a new language L' as
$$\langle x, u_1 \rangle \in L' \iff \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$
- Then $L' \in \Pi_{i-1}^p$. By applying the induction hypothesis: $\Pi_{i-1}^p \subseteq \mathbf{P}$, hence $L' \in \mathbf{P}$, i.e., exists a PTIME TM M' deciding L'

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$*, then* $\mathbf{PH} = \mathbf{P}$*, i.e., the polynomial hierarchy collapses to* $\mathbf{P}$*.*

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that
$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$
- Define a new language L' as
$$\begin{aligned} \langle x, u_1 \rangle \in L' \quad \Longleftrightarrow \quad & \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$
- Then $L' \in \Pi_{i-1}^p$. By applying the induction hypothesis: $\Pi_{i-1}^p \subseteq \mathbf{P}$, hence $L' \in \mathbf{P}$, i.e., exists a PTIME TM M' deciding L'
- Therefore, $x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} M'(x, u_1) = accept$

Properties of the Polynomial Hierarchy

Theorem

If **P** = **NP**, *then* **PH** = **P**, *i.e., the polynomial hierarchy collapses to* **P**.

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$
- Define a new language L' as
$$\langle x, u_1 \rangle \in L' \iff \forall u_2 \in \{0,1\}^{q(|x|)} \dots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept$$
- Then $L' \in \Pi_{i-1}^p$. By applying the induction hypothesis: $\Pi_{i-1}^p \subseteq \mathbf{P}$, hence $L' \in \mathbf{P}$, i.e., exists a PTIME TM M' deciding L'
- Therefore, $x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} M'(x, u_1) = accept$
- So, $L \in \Sigma_1^p = \mathbf{NP} = \mathbf{P}$

Properties of the Polynomial Hierarchy

Theorem

If $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{PH} = \mathbf{P}$, *i.e., the polynomial hierarchy collapses to* $\mathbf{P}$.

We show that $\Sigma_i^p, \Pi_i^p \subseteq \mathbf{P}$ by induction on i:

- The base step $i = 1$. $\Pi_i^p = co\mathbf{NP} = co\mathbf{P} = \mathbf{P}$ and $\Sigma_i^p = \mathbf{NP} = \mathbf{P}$
- The induction step $i > 1$. Let $L \in \Sigma_i^p$. There is a PTIME TM M and a PTIME computable function q such that
$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$
- Define a new language L' as
$$\begin{aligned} \langle x, u_1 \rangle \in L' \quad \iff \quad & \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_i) = accept \end{aligned}$$
- Then $L' \in \Pi_{i-1}^p$. By applying the induction hypothesis: $\Pi_{i-1}^p \subseteq \mathbf{P}$, hence $L' \in \mathbf{P}$, i.e., exists a PTIME TM M' deciding L'
- Therefore, $x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} M'(x, u_1) = accept$
- So, $L \in \Sigma_1^p = \mathbf{NP} = \mathbf{P}$
- $\Pi_i^p = co\Sigma_i^P = co\mathbf{P} = \mathbf{P}$

Properties of the Polynomial Hierarchy

Theorem

If $\Pi_i^p = \Sigma_i^p$ for some $i \geq 1$, then $\mathbf{PH} = \Sigma_i^p$, i.e., the polynomial hierarchy collapses to the i-th level.

Properties of the Polynomial Hierarchy

Theorem

If $\Pi_i^p = \Sigma_i^p$ for some $i \geq 1$, then $\mathbf{PH} = \Sigma_i^p$, i.e., the polynomial hierarchy collapses to the i-th level.

Similar to the case $i = 1$, do it by yourself.

Completeness of the Polynomial Hierarchy

Definition

For every $i \geq 1$, a language L is Σ_i^p-complete if

- $L \in \Sigma_i^p$ and
- for every $L' \in \Sigma_i^p$, $L' \leq_{\mathrm{P}} L$

Completeness of the Polynomial Hierarchy

Definition

For every $i \geq 1$, a language L is Σ_i^p-complete if

- $L \in \Sigma_i^p$ and
- for every $L' \in \Sigma_i^p$, $L' \leq_{\mathrm{P}} L$

Example

- SAT is **NP**-complete, therefore is Σ_1^p-complete

Completeness of the Polynomial Hierarchy

Definition

For every $i \geq 1$, a language L is Σ_i^p-complete if

- $L \in \Sigma_i^p$ and
- for every $L' \in \Sigma_i^p$, $L' \leq_{\mathrm{P}} L$

Example

- SAT is **NP**-complete, therefore is Σ_1^p-complete
- QBF is in the form of

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \cdots, x_n)$$

where $Q_i \in \{\forall, \exists\}$ and ϕ is an unquantified boolean formula

Completeness of the Polynomial Hierarchy

Definition

For every $i \geq 1$, a language L is Σ_i^p-complete if

- $L \in \Sigma_i^p$ and
- for every $L' \in \Sigma_i^p$, $L' \leq_{\mathrm{P}} L$

Example

- SAT is **NP**-complete, therefore is Σ_1^p-complete
- QBF is in the form of

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \cdots, x_n)$$

where $Q_i \in \{\forall, \exists\}$ and ϕ is an unquantified boolean formula

$$TQBF = \left\{ \langle \varphi \rangle \;\middle|\; \varphi \text{ is a true fully quantifier Boolean formula} \right\}.$$

Theorem

TQBF is PSPACE-*complete.*

Examples continued

- Define

$$\Sigma_i\mathrm{SAT} = \{\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i) = \mathit{true}\}$$

where $Q_i = \forall$ if i is even, otherwise $\exists$, each x_i is a vector of Boolean variables

Examples continued

- Define

$$\Sigma_i\text{SAT} = \{\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i) = \mathit{true}\}$$

where $Q_i = \forall$ if i is even, otherwise $\exists$, each x_i is a vector of Boolean variables

Examples continued

- Define

$$\Sigma_i\mathrm{SAT} = \{\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i) = \mathit{true}\}$$

  where $Q_i = \forall$ if i is even, otherwise $\exists$, each x_i is a vector of Boolean variables

Theorem

$\Sigma_i\mathrm{SAT}$ *is* Σ_i^p*-complete.*

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_iSAT is Σ_i^p-complete.

- Σ_iSAT is in Σ_i^p. Consider $\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i)$.

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_iSAT *is* Σ_i^p*-complete.*

- Σ_iSAT is in Σ_i^p. Consider $\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i)$. There is a polynomial time TM M and a PTIME computable function q such that

$$\begin{aligned} \phi \in \Sigma_i\text{SAT} \quad \Longleftrightarrow \quad & \exists x_1' \in \{0,1\}^{q(|\phi|)} \forall x_2' \in \{0,1\}^{q(|\phi|)} \dots \\ & Q_i X_i' \in \{0,1\}^{q(|\phi|)} M(\phi(x_1', x_2', \cdots, x_i')) \end{aligned}$$

where $q(|\phi|) = \max_{i=1}^{i} |x_i|$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_iSAT *is* Σ_i^p*-complete.*

- Σ_iSAT is in Σ_i^p. Consider $\exists x_1 \forall x_2 \cdots Q_i x_i \phi(x_1, x_2, \cdots, x_i)$. There is a polynomial time TM M and a PTIME computable function q such that

$$\begin{aligned}\phi \in \Sigma_i\text{SAT} \quad \iff \quad & \exists x_1' \in \{0,1\}^{q(|\phi|)} \forall x_2' \in \{0,1\}^{q(|\phi|)} \ldots \\ & Q_i X_i' \in \{0,1\}^{q(|\phi|)} M(\phi(x_1', x_2', \cdots, x_i'))\end{aligned}$$

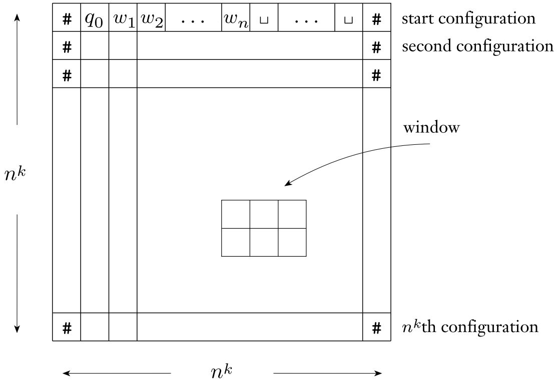
  where $q(|\phi|) = \max_{i=1}^{i} |x_i|$

- Σ_iSAT is Σ_i^p-hard.

Recall: SAT is **NP**-hard

Let N be an NTM that decides a language A in time n^k for some $k \in \mathbb{N}$.

A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of the branch of the computation of N on input w.



$$\varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}}.$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard.

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard. Suppose k is odd. Let $L \in \Sigma_k^p$.

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

$\Sigma_k\mathrm{SAT}$ *is* Σ_k^p*-complete.*

- $\Sigma_k\mathrm{SAT}$ is Σ_k^p-hard. Suppose k is odd. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$\begin{aligned} x \in L \quad \iff \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ & \exists u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept \end{aligned}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

$\Sigma_k\mathrm{SAT}$ *is* Σ_k^p*-complete.*

- $\Sigma_k\mathrm{SAT}$ is Σ_k^p-hard. Suppose k is odd. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \exists u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept$$

Let M' be a nondeterministic TM such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \forall u_{k-1} \in \{0,1\}^{q(|x|)} M'(x, u_1, \cdots, u_{k-1}) = accept$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard. Suppose k is odd. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that
$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \exists u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = \mathit{accept} \end{aligned}$$

Let M' be a nondeterministic TM such that

$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \forall u_{k-1} \in \{0,1\}^{q(|x|)} M'(x, u_1, \cdots, u_{k-1}) = \mathit{accept} \end{aligned}$$

$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \forall u_{k-1} \in \{0,1\}^{q(|x|)} \\ & \exists \overrightarrow{x_{i,j,s}} (\varphi_{\mathrm{cell}} \wedge \varphi_{\mathrm{start}} \wedge \varphi_{\mathrm{move}} \wedge \varphi_{\mathrm{accept}}) = \mathit{true} \end{aligned}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard.

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

$\Sigma_k\mathsf{SAT}$ *is* Σ_k^p*-complete.*

- $\Sigma_k\mathsf{SAT}$ is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$.

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = \mathit{accept}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept \end{aligned}$$

$$\begin{aligned} x \in \overline{L} \quad \Longleftrightarrow \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \exists u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = reject \end{aligned}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT *is* Σ_k^p*-complete.*

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept$$

$$x \in \overline{L} \iff \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \exists u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = reject$$

$$x \in \overline{L} \iff \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \exists u_k \in \{0,1\}^{q(|x|)} \overline{M}(x, u_1, \cdots, u_k) = accept$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT is Σ_k^p-complete.

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$
\begin{aligned}
x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\
& \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept
\end{aligned}
$$

$$
\begin{aligned}
x \in \overline{L} \quad \Longleftrightarrow \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \dots \\
& \exists u_k \in \{0,1\}^{q(|x|)} \overline{M}(x, u_1, \cdots, u_k) = accept
\end{aligned}
$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT is Σ_k^p-complete.

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$x \in L \iff \begin{array}{l} \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \\ \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept \end{array}$$

$$x \in \overline{L} \iff \begin{array}{l} \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \dots \\ \exists u_k \in \{0,1\}^{q(|x|)} \overline{M}(x, u_1, \cdots, u_k) = accept \end{array}$$

$$x \in \overline{L} \iff \begin{array}{l} \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \dots \\ \forall u_{k-1} \in \{0,1\}^{q(|x|)} \\ \exists \overrightarrow{x_{i,j,s}} (\varphi_{\mathrm{cell}} \wedge \varphi_{\mathrm{start}} \wedge \varphi_{\mathrm{move}} \wedge \varphi_{\mathrm{accept}}) = true \end{array}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT is Σ_k^p-complete.

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$\begin{aligned} x \in L \quad \Longleftrightarrow \quad & \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept \end{aligned}$$

$$\begin{aligned} x \in \overline{L} \quad \Longleftrightarrow \quad & \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots \\ & \forall u_{k-1} \in \{0,1\}^{q(|x|)} \\ & \exists \overrightarrow{x_{i,j,s}} (\varphi_{\mathrm{cell}} \wedge \varphi_{\mathrm{start}} \wedge \varphi_{\mathrm{move}} \wedge \varphi_{\mathrm{accept}}) = true \end{aligned}$$

Examples continued

Theorem (Meyer and Stockmeyer, 1972)

Σ_kSAT is Σ_k^p-complete.

- Σ_kSAT is Σ_k^p-hard. Suppose k is even. Let $L \in \Sigma_k^p$. There is a PTIME TM M and a PTIME computable function q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \forall u_k \in \{0,1\}^{q(|x|)} M(x, u_1, \cdots, u_k) = accept$$

$$x \in \overline{L} \iff \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \dots \forall u_{k-1} \in \{0,1\}^{q(|x|)} \exists \overrightarrow{x_{i,j,s}} (\varphi_{\mathrm{cell}} \wedge \varphi_{\mathrm{start}} \wedge \varphi_{\mathrm{move}} \wedge \varphi_{\mathrm{accept}}) = true$$

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \exists u_{k-1} \in \{0,1\}^{q(|x|)} \forall \overrightarrow{x_{i,j,s}} (\neg\varphi_{\mathrm{cell}} \vee \neg\varphi_{\mathrm{start}} \vee \neg\varphi_{\mathrm{move}} \vee \neg\varphi_{\mathrm{accept}}) = true$$

Outline

Interactive Proof Systems

- Probabilistic polynomial time algorithms provide a probabilistic analog to **P**.

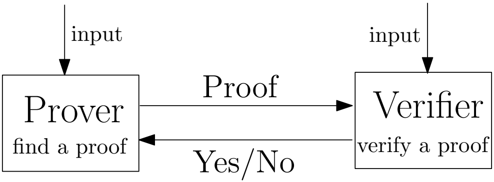
Interactive Proof Systems

- Probabilistic polynomial time algorithms provide a probabilistic analog to **P**.
- Interactive proof systems provide a way to define a probabilistic analog of the class **NP**.

Interactive Proof Systems

- Probabilistic polynomial time algorithms provide a probabilistic analog to **P**.
- Interactive proof systems provide a way to define a probabilistic analog of the class **NP**.

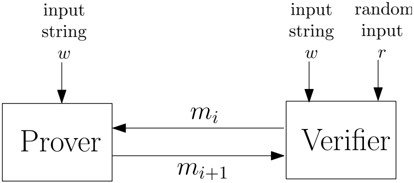
SAT problem



Interactive Proof Systems

- Verifier is a function $V : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{accept, reject\}$
- Prover is a function $P : \Sigma^* \times \Sigma^* \to \Sigma^*$

Interactive Proof Systems



$\forall i \geq 1,\ i$ is odd

$P(w, m_1 \sharp \cdots \sharp m_i) = m_{i+1}$

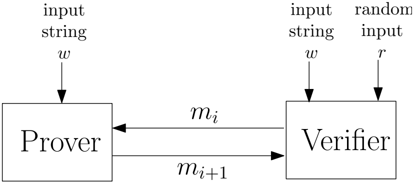
$\forall i \geq 0,\ i$ is even

$V(w, r, m_1 \sharp \cdots \sharp m_i) = m_{i+1}$

Interactive Proof Systems

- Verifier is a function $V : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{accept, reject\}$
- Prover is a function $P : \Sigma^* \times \Sigma^* \to \Sigma^*$

Interactive Proof Systems



$\forall i \geq 1,\ i$ is odd $\quad P(w, m_1 \sharp \cdots \sharp m_i) = m_{i+1}$

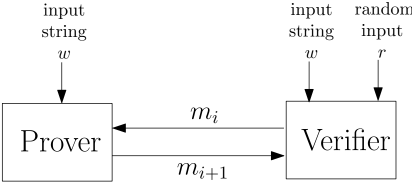
$\forall i \geq 0,\ i$ is even $\quad V(w, r, m_1 \sharp \cdots \sharp m_i) = m_{i+1}$

$(V \leftrightarrow P)(w, r) =$ accepts for given input string w and random input r, if V outputs the accept message, i.e., $m_i =$accept for some i.

Interactive Proof Systems

- Verifier is a function $V : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{accept, reject\}$
- Prover is a function $P : \Sigma^* \times \Sigma^* \to \Sigma^*$

Interactive Proof Systems



$\forall i \geq 1,\ i$ is odd $\qquad \forall i \geq 0,\ i$ is even

$P(w, m_1\sharp \cdots \sharp m_i) = m_{i+1} \qquad V(w, r, m_1\sharp \cdots \sharp m_i) = m_{i+1}$

$(V \leftrightarrow P)(w, r) =$ accepts for given input string w and random input r, if V outputs the accept message, i.e., $m_i =$accept for some i.

$$\Pr[V \leftrightarrow P \text{ accepts } w] = \Pr[(V \leftrightarrow P)(w, r) = \text{accepts}]$$

Interactive Polynomial Time

Definition

A language A is in interactive polynomial Time (IP) if some polynomial time computable function V exists such that for some (arbitrary) function P and for every (arbitrary) function $\widetilde{P}$ and for every string w with length n

- $w \in A$ implies that $\Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$
- $w \notin A$ implies that $\Pr[V \leftrightarrow \widetilde{P} \text{ accepts } w] \leq \frac{1}{3}$

where the lengths of the Verifier's random input, each of the messages exchanged between the Verifier and the Prover are $p(n)$ and the total number of messages exchanged is at most $p(n)$ for some polynomial p.

Interactive Polynomial Time

Definition

A language A is in interactive polynomial Time (IP) if some polynomial time computable function V exists such that for some (arbitrary) function P and for every (arbitrary) function $\widetilde{P}$ and for every string w with length n

- $w \in A$ implies that $\Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$
- $w \notin A$ implies that $\Pr[V \leftrightarrow \widetilde{P} \text{ accepts } w] \leq \frac{1}{3}$

where the lengths of the Verifier's random input, each of the messages exchanged between the Verifier and the Prover are $p(n)$ and the total number of messages exchanged is at most $p(n)$ for some polynomial p.

Theorem

NP $\subseteq$ IP *and* **BPP** $\subseteq$ IP.

Graph isomorphism

Definition

Two graphs are isomorphic if they are same up-to node renaming

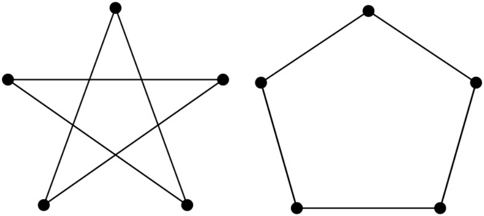
$$\mathrm{ISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$$

Graph isomorphism

Definition

Two graphs are isomorphic if they are same up-to node renaming

$$\text{ISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$$

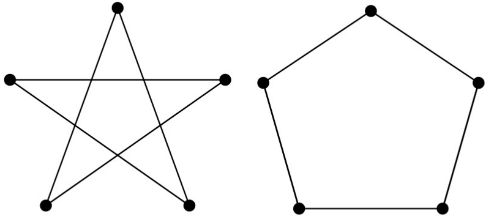


Graph isomorphism

Definition

Two graphs are isomorphic if they are same up-to node renaming

$$\text{ISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$$



Theorem

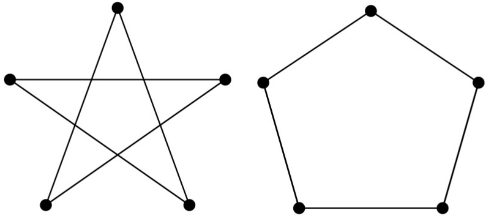
$ISO \in \mathbf{NP}$.

Graph isomorphism

Definition

Two graphs are isomorphic if they are same up-to node renaming

$$\text{ISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$$



Theorem

ISO $\in$ **NP**.

Open problem: ISO is **NP**-complete or ISO $\in$ **P**?

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \text{IP}$

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \text{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \mathrm{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \text{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \mathrm{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \mathrm{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

- If $\langle G_1, G_2\rangle \in$ NONISO,

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \mathrm{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

- If $\langle G_1, G_2 \rangle \in \text{NONISO}$, then there is a Prover that can identify whether H came from G_1 or G_2

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \text{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

- If $\langle G_1, G_2\rangle \in \text{NONISO}$, then there is a Prover that can identify whether H came from G_1 or G_2
- If $\langle G_1, G_2\rangle \in \text{ISO}$,

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \text{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

- If $\langle G_1, G_2\rangle \in \text{NONISO}$, then there is a Prover that can identify whether H came from G_1 or G_2
- If $\langle G_1, G_2\rangle \in \text{ISO}$, then no Prover even with unlimited computational power can identify whether H came from G_1 or G_2, it has 50-50 chance

Graph non-isomorphism

$$\text{NONISO} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are not isomorphic graphs}\}$$

Theorem

$NONISO \in co\mathbf{NP} \cap \mathrm{IP}$

1. Verifier randomly selects either G_1 or G_2 and then randomly reorders its nodes to obtain a graph H
2. The Verifier sends H to the Prover
3. The Prover must respond by declaring whether G_1 or G_2 was the source of H
4. If Prover's response is correct, then accept; otherwise reject

- If $\langle G_1, G_2 \rangle \in \text{NONISO}$, then there is a Prover that can identify whether H came from G_1 or G_2
- If $\langle G_1, G_2 \rangle \in \text{ISO}$, then no Prover even with unlimited computational power can identify whether H came from G_1 or G_2, it has 50-50 chance
- The Verifier can repeat the above protocol in order to get the desired error probability

Theorem
IP $=$ PSPACE.

Theorem

IP $=$ PSPACE.

- For any language in PSPACE, a Prover can convince a probabilistic polynomial time Verifier about the membership of a string in the language, even though a conventional proof of membership might be exponentially long
- Proof: read the textbook.

Outline

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously
- Parallel computers may solve certain problems much faster than sequential computers, which can only do a single operation at a time

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously
- Parallel computers may solve certain problems much faster than sequential computers, which can only do a single operation at a time
- Introduce the theory of parallel computation

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously
- Parallel computers may solve certain problems much faster than sequential computers, which can only do a single operation at a time
- Introduce the theory of parallel computation
  - describe one model of a parallel computer

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously
- Parallel computers may solve certain problems much faster than sequential computers, which can only do a single operation at a time
- Introduce the theory of parallel computation
  - describe one model of a parallel computer
  - give examples of certain problems that lend themselves well to parallelization

Parallel Computer

- A parallel computer is one that can perform multiple operations simultaneously
- Parallel computers may solve certain problems much faster than sequential computers, which can only do a single operation at a time
- Introduce the theory of parallel computation
  - describe one model of a parallel computer
  - give examples of certain problems that lend themselves well to parallelization
  - explore the possibility that parallelism may not be suitable for certain other problem

Boolean Circuits

Definition

A Boolean circuit is a collection of gates and inputs connected by wires. Cycles aren't permitted. Gates take three forms: AND gates, OR gates, and NOT gates

- The size of a circuit C is the number of gates that it contains
- The size complexity of a circuit family $(C_0, C_1, C_2, \cdots)$ is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the size of C_n
- The depth of a circuit is the length (number of wires) of the longest path from an input variable to the output gate. Depth minimal circuits and circuit families, and the depth complexity of circuit families are similar

Definition

The circuit complexity of a language is the size complexity of a minimal circuit family for that language.
The circuit depth complexity of a language is defined similarly, using depth instead of size.

Uniform Boolean Circuits

Definition

A family of circuits $(C_0, C_1, C_2, \dots)$ is uniform if some log space transducer T outputs $\langle C_n \rangle$ when T's input is 1^n.

Uniform Boolean Circuits

Definition

A family of circuits $(C_0, C_1, C_2, \ldots)$ is uniform if some log space transducer T outputs $\langle C_n \rangle$ when T's input is 1^n.

- A uniform family of circuits is a model of a parallel computer,

Uniform Boolean Circuits

Definition

A family of circuits $(C_0, C_1, C_2, \ldots)$ is uniform if some log space transducer T outputs $\langle C_n \rangle$ when T's input is 1^n.

- A uniform family of circuits is a model of a parallel computer, where each gate to be an individual processor

Uniform Boolean Circuits

Definition

A family of circuits $(C_0, C_1, C_2, \ldots)$ is uniform if some log space transducer T outputs $\langle C_n \rangle$ when T's input is 1^n.

- A uniform family of circuits is a model of a parallel computer, where each gate to be an individual processor
- A language has simultaneous size-depth circuit complexity at most $(f(n), g(n))$ if a uniform circuit family exists for that language with size complexity $f(n)$ and depth complexity $g(n)$

Example

The m-input parity function $\texttt{parity}_m : \{0,1\}^m \to \{0,1\}$ outputs 1 if an odd number of 1's appear in the input variables.

Example

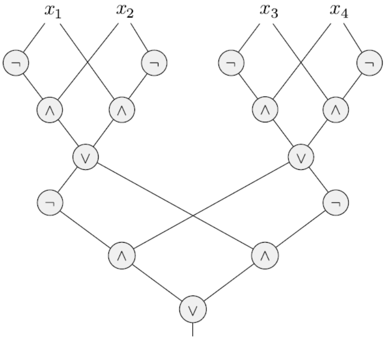
The m-input parity function $\texttt{parity}_m : \{0,1\}^m \to \{0,1\}$ outputs 1 if an odd number of 1's appear in the input variables.

$$\texttt{parity}_m(x_1, \cdots, x_m) = \bigoplus_{i=1}^{m} x_i \text{ and } x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y)$$

Example

The m-input parity function $\texttt{parity}_m : \{0,1\}^m \to \{0,1\}$ outputs 1 if an odd number of 1's appear in the input variables.

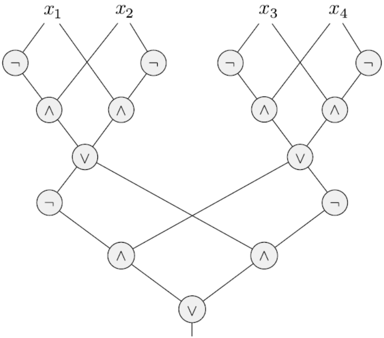
$$\texttt{parity}_m(x_1, \cdots, x_m) = \bigoplus_{i=1}^{m} x_i \text{ and } x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y)$$



Example

The m-input parity function $\texttt{parity}_m : \{0,1\}^m \to \{0,1\}$ outputs 1 if an odd number of 1's appear in the input variables.

$$\texttt{parity}_m(x_1, \cdots, x_m) = \bigoplus_{i=1}^{m} x_i \text{ and } x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y)$$



The simultaneous size-depth circuit complexity: $(O(n), O(\log n))$, as each $\oplus$ costs 5 gates

Boolean Matrix Multiplication

- The input of Boolean matrix multiplication has $2m^2 = n$ variables representing two $m \times m$ matrices $A = \{a_{ik}\}_{1 \leq i,k \leq m}$ and $B = \{b_{ik}\}_{1 \leq i,k \leq m}$

Boolean Matrix Multiplication

- The input of Boolean matrix multiplication has $2m^2 = n$ variables representing two $m \times m$ matrices $A = \{a_{ik}\}_{1 \leq i,k \leq m}$ and $B = \{b_{ik}\}_{1 \leq i,k \leq m}$
- The output is a m^2 values representing a $m \times m$ Boolean matrix $C = \{c_{ik}\}_{1 \leq i,k \leq m}$, where

$$c_{ik} = \bigvee_{1 \leq j \leq m} (a_{ij} \wedge b_{jk})$$

Boolean Matrix Multiplication

- The input of Boolean matrix multiplication has $2m^2 = n$ variables representing two $m \times m$ matrices $A = \{a_{ik}\}_{1 \le i,k \le m}$ and $B = \{b_{ik}\}_{1 \le i,k \le m}$
- The output is a m^2 values representing a $m \times m$ Boolean matrix $C = \{c_{ik}\}_{1 \le i,k \le m}$, where

$$c_{ik} = \bigvee_{1 \le j \le m} (a_{ij} \wedge b_{jk})$$

- For c_{ik}, we use gate g_{ijk} to compute $(a_{ij} \wedge b_{jk})$ for each j
- For each i, k, we use a binary tree of $\vee$ gates to compute $\bigvee_{1 \le j \le m} g_{ijk}$

Boolean Matrix Multiplication

- The input of Boolean matrix multiplication has $2m^2 = n$ variables representing two $m \times m$ matrices $A = \{a_{ik}\}_{1 \le i,k \le m}$ and $B = \{b_{ik}\}_{1 \le i,k \le m}$
- The output is a m^2 values representing a $m \times m$ Boolean matrix $C = \{c_{ik}\}_{1 \le i,k \le m}$, where

$$c_{ik} = \bigvee_{1 \le j \le m} (a_{ij} \wedge b_{jk})$$

- For c_{ik}, we use gate g_{ijk} to compute $(a_{ij} \wedge b_{jk})$ for each j
- For each i, k, we use a binary tree of $\vee$ gates to compute $\bigvee_{1 \le j \le m} g_{ijk}$

The simultaneous size-depth circuit complexity:
$(O(m^3), O(\log m)) = (O(n^{1.5}), O(\log n))$

The Class NC

Definition

For $i \geq 1$, let NC^i be the class of languages that can be decided by a uniform family of circuits with polynomial size and $O(\log^i n)$ depth.

The Class NC

Definition

For $i \geq 1$, let NC^i be the class of languages that can be decided by a uniform family of circuits with polynomial size and $O(\log^i n)$ depth. Let

$$NC = \bigcup_{i \geq 1} \mathsf{NC}^i.$$

The Class NC

Definition

For $i \geq 1$, let NC^i be the class of languages that can be decided by a uniform family of circuits with polynomial size and $O(\log^i n)$ depth. Let

$$NC = \bigcup_{i \geq 1} \mathsf{NC}^i.$$

Functions that are computed by such circuit families are called NC^i computable or NC computable.

The Class NC

Definition

For $i \geq 1$, let NC^i be the class of languages that can be decided by a uniform family of circuits with polynomial size and $O(\log^i n)$ depth. Let

$$NC = \bigcup_{i \geq 1} \mathrm{NC}^i.$$

Functions that are computed by such circuit families are called NC^i computable or NC computable.

The NC^i computable or NC computable problems may be considered to be highly parallelizable with a moderate number of processors

Boolean Matrix Multiplication$\in \mathrm{NC}^1$

Connection Between TM Space and Circuit Depth

Theorem

$\mathsf{NC}^1 \subseteq \mathbf{L}$, *i.e., problems that are solvable in logarithmic depth are also solvable in logarithmic space.*

Let A be a language in NC^1.

Connection Between TM Space and Circuit Depth

Theorem

$\mathsf{NC}^1 \subseteq \mathbf{L}$, *i.e., problems that are solvable in logarithmic depth are also solvable in logarithmic space.*

Let A be a language in NC^1.

M on input w with length n:

Connection Between TM Space and Circuit Depth

Theorem

$\mathsf{NC}^1 \subseteq \mathbf{L}$, *i.e., problems that are solvable in logarithmic depth are also solvable in logarithmic space.*

Let A be a language in NC^1.

M on input w with length n:

1. Construct C_n as the log space transducer T for the uniform circuit family of A

Connection Between TM Space and Circuit Depth

Theorem

$NC^1 \subseteq \mathbf{L}$, *i.e., problems that are solvable in logarithmic depth are also solvable in logarithmic space.*

Let A be a language in NC^1.

M on input w with length n:

1. Construct C_n as the log space transducer T for the uniform circuit family of A
2. Evaluate C_n using a depth-first search from the output gate

Connection Between TM Space and Circuit Depth

Theorem

$\mathsf{NC}^1 \subseteq \mathbf{L}$, *i.e., problems that are solvable in logarithmic depth are also solvable in logarithmic space.*

Let A be a language in NC^1.

M on input w with length n:

1. Construct C_n as the log space transducer T for the uniform circuit family of A
2. Evaluate C_n using a depth-first search from the output gate

The depth of C_n is $O(\log n)$, therefore recursion depth is $O(\log n)$, hence space of M

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq \mathrm{NC}^2$, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq \mathrm{NC}^2$, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq \mathrm{NC}^2$, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.
- If the reading head in c_1 is i and c_2 is the immediate successor of c_1 in M after reading 1 (resp. 0),

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.
- If the reading head in c_1 is i and c_2 is the immediate successor of c_1 in M after reading 1 (resp. 0), then (c_1, c_2) is an directed edge in G_n with label w_i if w_i (resp. $\overline{w_i}$).

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.
- If the reading head in c_1 is i and c_2 is the immediate successor of c_1 in M after reading 1 (resp. 0), then (c_1, c_2) is an directed edge in G_n with label w_i if w_i (resp. $\overline{w_i}$).
- A log space transducer is capable of constructing G_n by setting the edges according to $|w|$

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq \mathrm{NC}^2$, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.
- If the reading head in c_1 is i and c_2 is the immediate successor of c_1 in M after reading 1 (resp. 0), then (c_1, c_2) is an directed edge in G_n with label w_i if w_i (resp. $\overline{w_i}$).
- A log space transducer is capable of constructing G_n by setting the edges according to $|w|$ and therefore C_n on 1^n input.

Connection Between TM Space and Circuit Depth

Theorem

NL $\subseteq$ NC^2, *i.e., problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.*

Let A be a language over $\Sigma = \{0, 1\}$ and decided by a **NL** TM M.

- We construct a uniform circuit family $(C_0, C_1, \cdots)$ for A.
- To construct C_n, we construct a graph G_n that is similar to the computation graph for M on an input w of length n. The graph G_n has $O(n^2)$ nodes.
- The inputs to the circuit are variables $w_1, \cdots, w_n$.
- If the reading head in c_1 is i and c_2 is the immediate successor of c_1 in M after reading 1 (resp. 0), then (c_1, c_2) is an directed edge in G_n with label w_i if w_i (resp. $\overline{w_i}$).
- A log space transducer is capable of constructing G_n by setting the edges according to $|w|$ and therefore C_n on 1^n input.
- C_n has polynomial size and $O(\log^2 n)$ depth, see Theorem 8.25 (*PATH* is **NL**-complete) and Theorem 9.30 (TIME$(t(n)) \Rightarrow$ circuit complexity $O(t^2(n))$).

Connection Between TM Space and Circuit Depth

$$\mathsf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathsf{NC}^2$$

Connection Between TM Space and Circuit Depth

$$\mathsf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathsf{NC}^2$$

Theorem

$\mathsf{NC} \subseteq \mathbf{P}$.

Connection Between TM Space and Circuit Depth

$$\mathrm{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathrm{NC}^2$$

Theorem

$\mathrm{NC} \subseteq \mathbf{P}$.

A polynomial time algorithm can run the log space transducer to generate circuit C_n and simulate it on an input of length n

P-completeness

Definition

A language B is **P**-complete if

- $B \in \mathbf{P}$

P-completeness

Definition

A language B is **P**-complete if

- $B \in \mathbf{P}$ and
- every $A \in \mathbf{P}$ is log space reducible to B.

P-completeness

Definition

A language B is **P**-complete if

- $B \in \mathbf{P}$ and
- every $A \in \mathbf{P}$ is log space reducible to B.

Theorem

If $A \leq_{\mathrm{L}} B$ *and* $B \in \mathrm{NC}$, *then* $A \in \mathrm{NC}$.

P-completeness

Definition

A language B is **P**-complete if

- $B \in \mathbf{P}$ and
- every $A \in \mathbf{P}$ is log space reducible to B.

Theorem

If $A \leq_{\mathrm{L}} B$ and $B \in \mathrm{NC}$, then $A \in \mathrm{NC}$.

Because of $\mathbf{NL} \subseteq \mathrm{NC}^2$, NC circuit families can compute log space reductions.

P-completeness

Definition

A language B is **P**-complete if

- $B \in \mathbf{P}$ and
- every $A \in \mathbf{P}$ is log space reducible to B.

Theorem

If $A \leq_{\mathrm{L}} B$ and $B \in \mathrm{NC}$, then $A \in \mathrm{NC}$.

Because of $\mathbf{NL} \subseteq \mathrm{NC}^2$, NC circuit families can compute log space reductions.

$$\text{CIRCUIT-VALUE} = \{\langle C, x\rangle \mid C \text{ is a Boolean circuit and } C(x) = 1\}$$

Theorem

CIRCUIT-VALUE is **P***-complete.*

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

- CIRCUIT-VALUE is in **P**: depth-first search of the circuit

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

- CIRCUIT-VALUE is in **P**: depth-first search of the circuit
- CIRCUIT-VALUE is **P**-hard: any language $A \in \mathbf{P}$ to CIRCUIT-VALUE, similar to Theorem 9.30 (CIRCUIT-SAT is NP-complete)

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

- CIRCUIT-VALUE is in **P**: depth-first search of the circuit
- CIRCUIT-VALUE is **P**-hard: any language $A \in \mathbf{P}$ to CIRCUIT-VALUE, similar to Theorem 9.30 (CIRCUIT-SAT is NP-complete)
  - On input w, the reduction produces a circuit that simulates the polynomial time Turing machine for A.

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

- CIRCUIT-VALUE is in **P**: depth-first search of the circuit
- CIRCUIT-VALUE is **P**-hard: any language $A \in \mathbf{P}$ to CIRCUIT-VALUE, similar to Theorem 9.30 (CIRCUIT-SAT is NP-complete)
  - On input w, the reduction produces a circuit that simulates the polynomial time Turing machine for A.
  - The input to the circuit is w itself.

P-completeness

Theorem

CIRCUIT-VALUE is **P***-complete.*

- CIRCUIT-VALUE is in **P**: depth-first search of the circuit
- CIRCUIT-VALUE is **P**-hard: any language $A \in \mathbf{P}$ to CIRCUIT-VALUE, similar to Theorem 9.30 (CIRCUIT-SAT is NP-complete)
  - On input w, the reduction produces a circuit that simulates the polynomial time Turing machine for A.
  - The input to the circuit is w itself.
  - The reduction can be carried out in log space because the circuit it produces has a simple and repetitive structure.

Outline

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.
  - Evidence of a code's quality was obtained by hiring experts who tried to break it.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.
  - Evidence of a code's quality was obtained by hiring experts who tried to break it.
  - Complexity theory provides another way to gain evidence for a code's security.

Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.
  - Evidence of a code's quality was obtained by hiring experts who tried to break it.
  - Complexity theory provides another way to gain evidence for a code's security.
  - One of the advantages of using complexity theory as a foundation for cryptography is that it helps to clarify the assumptions being made when we argue about security.

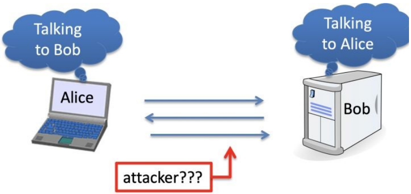
Cryptography

- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.
  - Evidence of a code's quality was obtained by hiring experts who tried to break it.
  - Complexity theory provides another way to gain evidence for a code's security.
  - One of the advantages of using complexity theory as a foundation for cryptography is that it helps to clarify the assumptions being made when we argue about security.
  - NP-completeness concerns worst-case complexity,

Cryptography

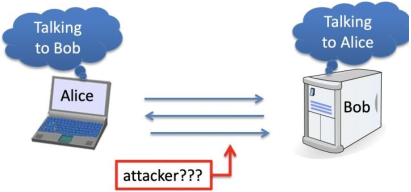
- A key that is too short may be discovered through a brute-force search of the entire space of possible keys.
- The only way to get perfect cryptographic security is with keys that are as long as the combined length of all messages sent, called one-time pad.
- One-time pads are too cumbersome to be considered practical.
- we are unable to prove mathematically that codes are unbreakable.
  - Evidence of a code's quality was obtained by hiring experts who tried to break it.
  - Complexity theory provides another way to gain evidence for a code's security.
  - One of the advantages of using complexity theory as a foundation for cryptography is that it helps to clarify the assumptions being made when we argue about security.
  - NP-completeness concerns worst-case complexity, but, we need to measure average-case complexity rather than worst-case complexity, e.g., integer factorization

The One Time Pad & Perfect Security



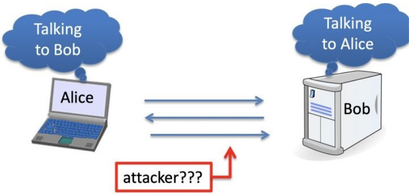
- Three spaces (K, M, C) and two algorithms (E, D), $E : K \times M \to C$ is enc. alg., $D : K \times C \to M$ is dec. alg.
  $\forall m \in M, \forall k \in K : \; D(k, E(k, m)) = m$

The One Time Pad & Perfect Security



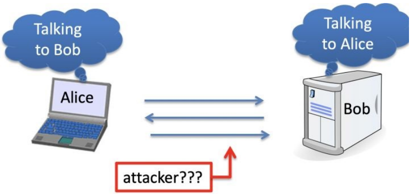
- Three spaces (K, M, C) and two algorithms (E, D), $E : K \times M \to C$ is enc. alg., $D : K \times C \to M$ is dec. alg.
  $\forall m \in M, \forall k \in K : \; D(k, E(k, m)) = m$
- The One Time Pad: $K = M = C = \{0, 1\}^n$, $E(k, m) = k \oplus m$, $D(k, c) = k \oplus c$

The One Time Pad & Perfect Security



- Three spaces (K, M, C) and two algorithms (E, D), $E : K \times M \to C$ is enc. alg., $D : K \times C \to M$ is dec. alg.
  $\forall m \in M, \forall k \in K : \; D(k, E(k, m)) = m$
- The One Time Pad: $K = M = C = \{0, 1\}^n$, $E(k, m) = k \oplus m$, $D(k, c) = k \oplus c = k \oplus E(k, m) = k \oplus k \oplus m = m$

The One Time Pad & Perfect Security

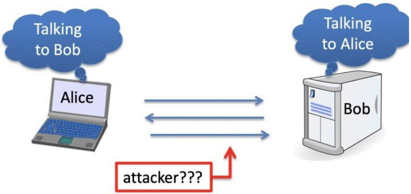


- Three spaces (K, M, C) and two algorithms (E, D), $E : K \times M \to C$ is enc. alg., $D : K \times C \to M$ is dec. alg.
  $\forall m \in M, \forall k \in K : \; D(k, E(k, m)) = m$
- The One Time Pad: $K = M = C = \{0, 1\}^n$, $E(k, m) = k \oplus m$,
  $D(k, c) = k \oplus c = k \oplus E(k, m) = k \oplus k \oplus m = m$
- Perfect Security (Claude Elwood Shannon): Three spaces (K, M, C) and two algorithms (E, D) is perfect security iff

$$\forall c \in C, \; \forall m_1, m_2 \in M : \; Pr[E(k, m_0) = c] = Pr[E(k, m_1) = c]$$

- The adversary is not able to distinguish the plaintext of c, as, the distribution of plaintext such that $E(k, m) = c$ is uniform

The One Time Pad & Perfect Security



- Three spaces (K, M, C) and two algorithms (E, D), $E : K \times M \to C$ is enc. alg., $D : K \times C \to M$ is dec. alg.
  $\forall m \in M, \forall k \in K : \ D(k, E(k, m)) = m$
- The One Time Pad: $K = M = C = \{0, 1\}^n$, $E(k, m) = k \oplus m$, $D(k, c) = k \oplus c = k \oplus E(k, m) = k \oplus k \oplus m = m$
- Perfect Security (Claude Elwood Shannon): Three spaces (K, M, C) and two algorithms (E, D) is perfect security iff

$$\forall c \in C, \ \forall m_1, m_2 \in M : \ Pr[E(k, m_0) = c] = Pr[E(k, m_1) = c]$$

- The adversary is not able to distinguish the plaintext of c, as, the distribution of plaintext such that $E(k, m) = c$ is uniform
- 2nd place Bell Labs Prize, InstaHide: Instance-hiding Schemes for Private Distributed Learning (Yangsibo Huang, Zhao Song, Kai Li, Sanjeev Arora, ICML 2020)

Cryptography

- Private-key cryptosystem: the same key is used for both encryption and decryption, e.g., AES

Cryptography

- Private-key cryptosystem: the same key is used for both encryption and decryption, e.g., AES
- Public-key cryptosystem: the decryption key is different from, and not easily computed from, the encryption key, e.g., RSA

Cryptography

- Private-key cryptosystem: the same key is used for both encryption and decryption, e.g., AES
- Public-key cryptosystem: the decryption key is different from, and not easily computed from, the encryption key, e.g., RSA
- Certain public-key cryptosystems can also be used for digital signatures, i.e., an individual applies his secret decryption algorithm to a message before sending it, anyone can check that it actually came from him by applying the public encryption algorithm.

Cryptography

- Private-key cryptosystem: the same key is used for both encryption and decryption, e.g., AES
- Public-key cryptosystem: the decryption key is different from, and not easily computed from, the encryption key, e.g., RSA
- Certain public-key cryptosystems can also be used for digital signatures, i.e., an individual applies his secret decryption algorithm to a message before sending it, anyone can check that it actually came from him by applying the public encryption algorithm.
- One-way functions: allow us to construct secure private-key cryptosystems

Cryptography

- Private-key cryptosystem: the same key is used for both encryption and decryption, e.g., AES
- Public-key cryptosystem: the decryption key is different from, and not easily computed from, the encryption key, e.g., RSA
- Certain public-key cryptosystems can also be used for digital signatures, i.e., an individual applies his secret decryption algorithm to a message before sending it, anyone can check that it actually came from him by applying the public encryption algorithm.
- One-way functions: allow us to construct secure private-key cryptosystems
- Trapdoor functions: allow us to construct public-key cryptosystems

Preliminaries

- A function $f : \Sigma^* \to \Sigma^*$ is length-preserving if the lengths of w and $f(w)$ are equal for every $w \in \Sigma^*$.

Preliminaries

- A function $f : \Sigma^* \to \Sigma^*$ is length-preserving if the lengths of w and $f(w)$ are equal for every $w \in \Sigma^*$.
- A length-preserving function $f : \Sigma^* \to \Sigma^*$ is a permutation if it never maps two strings to the same place, i.e., $f(x) \neq f(y)$ for all $x \neq y$.

Preliminaries

- A function $f : \Sigma^* \to \Sigma^*$ is length-preserving if the lengths of w and $f(w)$ are equal for every $w \in \Sigma^*$.
- A length-preserving function $f : \Sigma^* \to \Sigma^*$ is a permutation if it never maps two strings to the same place, i.e., $f(x) \neq f(y)$ for all $x \neq y$.
- A probabilistic TM M computes a probabilistic function $M : \Sigma^* \to \Sigma^*$,

$$Pr[M(w) = x] = Pr[M \text{ accepts } w \text{ and outputs } x]$$

where w is the input, x is the output, i.e., the context on the tape when M halts.

Preliminaries

- A function $f : \Sigma^* \to \Sigma^*$ is length-preserving if the lengths of w and $f(w)$ are equal for every $w \in \Sigma^*$.
- A length-preserving function $f : \Sigma^* \to \Sigma^*$ is a permutation if it never maps two strings to the same place, i.e., $f(x) \neq f(y)$ for all $x \neq y$.
- A probabilistic TM M computes a probabilistic function $M : \Sigma^* \to \Sigma^*$,

$$Pr[M(w) = x] = Pr[M \text{ accepts } w \text{ and outputs } x]$$

  where w is the input, x is the output, i.e., the context on the tape when M halts.

Note that M may sometimes fail to accept on input w:

$$\sum_{x \in \Sigma^*} Pr[M(w) = x] \leq 1$$

One-way permutation

Definition

A one-way permutation is a permutation f with the following two properties:

- It is computable in polynomial time.

One-way permutation

Definition

A one-way permutation is a permutation f with the following two properties:

- It is computable in polynomial time.
- For every probabilistic polynomial time TM M, every k, and sufficiently large n, if we pick a random w of length n and run M on input $f(w)$, $Pr_{M,w}[M(f(w)) = w] \leq n^{-k}$, where $Pr_{M,w}$ means that the probability is taken over the random choices made by M and the random selection of w.

One-way permutation

Definition

A one-way permutation is a permutation f with the following two properties:

- It is computable in polynomial time.
- For every probabilistic polynomial time TM M, every k, and sufficiently large n, if we pick a random w of length n and run M on input $f(w)$, $Pr_{M,w}[M(f(w)) = w] \leq n^{-k}$, where $Pr_{M,w}$ means that the probability is taken over the random choices made by M and the random selection of w.

For one-way permutations, any probabilistic polynomial time algorithm has only a small probability of inverting f; that is, it is unlikely to compute w from $f(w)$

One-way function

Definition

A one-way function is a length-preserving function f with the following two properties:

- It is computable in polynomial time.

One-way function

Definition

A one-way function is a length-preserving function f with the following two properties:

- It is computable in polynomial time.
- For every probabilistic polynomial time TM M, every k, and sufficiently large n, if we pick a random w of length n and run M on input $f(w)$,

$$Pr_{M,w}[M(f(w)) = y, \text{ where } f(y) = f(w)] \leq n^{-k}$$

One-way function

Definition

A one-way function is a length-preserving function f with the following two properties:

- It is computable in polynomial time.
- For every probabilistic polynomial time TM M, every k, and sufficiently large n, if we pick a random w of length n and run M on input $f(w)$,

$$Pr_{M,w}[M(f(w)) = y, \text{ where } f(y) = f(w)] \leq n^{-k}$$

For one-way functions, any probabilistic polynomial time algorithm is unlikely to be able to find any y that maps to $f(w)$.

One-way function: example

The multiplication function mult is a candidate for a one-way function.

Definition

- Let $\Sigma = \{0, 1\}$, for any $w \in \Sigma^*$, let $\mathrm{mult}(w)$ be the string representing the product of the first and second halves of w, i.e.,

$$\mathrm{mult}(w) = u \cdot v$$

  where $w = uv$ such that $|u| = |v|$ if $|w|$ is even, and $|u| = |v| + 1$ if $|w|$ odd.

We pad $\mathrm{mult}(w)$ with leading 0s so that it has the same length as w.

One-way function: example

The multiplication function mult is a candidate for a one-way function.

Definition

- Let $\Sigma = \{0, 1\}$, for any $w \in \Sigma^*$, let $\text{mult}(w)$ be the string representing the product of the first and second halves of w, i.e.,

$$\text{mult}(w) = u \cdot v$$

  where $w = uv$ such that $|u| = |v|$ if $|w|$ is even, and $|u| = |v| + 1$ if $|w|$ odd.

We pad $\text{mult}(w)$ with leading 0s so that it has the same length as w.

Despite a great deal of research into the integer factorization problem, no probabilistic polynomial time algorithm is known that can invert mult , even on a polynomial fraction of inputs

One-way function: application

One simple application of a one-way function is a provably secure password system.

One-way function: application

One simple application of a one-way function is a provably secure password system.

- A user must enter a password to gain access to some resource.

One-way function: application

One simple application of a one-way function is a provably secure password system.

- A user must enter a password to gain access to some resource.
- The system keeps a database of users' passwords in an encrypted form computed via some one-way function.

One-way function: application

One simple application of a one-way function is a provably secure password system.

- A user must enter a password to gain access to some resource.
- The system keeps a database of users' passwords in an encrypted form computed via some one-way function.
- When a user enters a password, the system checks it for validity by encrypting it to determine whether it matches the version stored in the database.

One-way function: application

One simple application of a one-way function is a provably secure password system.

- A user must enter a password to gain access to some resource.
- The system keeps a database of users' passwords in an encrypted form computed via some one-way function.
- When a user enters a password, the system checks it for validity by encrypting it to determine whether it matches the version stored in the database.
- An encryption scheme that is difficult to invert is desirable because it makes the unencrypted password difficult to obtain from the encrypted form.

One-way function: application

One simple application of a one-way function is a provably secure password system.

- A user must enter a password to gain access to some resource.
- The system keeps a database of users' passwords in an encrypted form computed via some one-way function.
- When a user enters a password, the system checks it for validity by encrypting it to determine whether it matches the version stored in the database.
- An encryption scheme that is difficult to invert is desirable because it makes the unencrypted password difficult to obtain from the encrypted form.

We don't know whether the existence of a one-way function alone is enough to allow the construction of a public-key cryptosystem

Trapdoor function

- A family of functions $\{f_i\}_{i \in \Sigma^*}$ can be represented by the single function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for every $i \in \Sigma^*$ and $w \in \Sigma^*$: $f(i, w) = f_i(w)$

Trapdoor function

- A family of functions $\{f_i\}_{i \in \Sigma^*}$ can be represented by the single function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for every $i \in \Sigma^*$ and $w \in \Sigma^*$: $f(i, w) = f_i(w)$
- f is length-preserving if for each $i \in \Sigma^*$, the functions f_i is length preserving.

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$
- For every w of length n, and every output $\langle i, t \rangle$ of G that occurs with nonzero probability for some input to G,
  $h(t, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)$

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$
- For every w of length n, and every output $\langle i, t \rangle$ of G that occurs with nonzero probability for some input to G,
  $h(t, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)$
- G generates an index i while simultaneously generating a value t that allows f_i to be inverted quickly.

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$
- For every w of length n, and every output $\langle i, t \rangle$ of G that occurs with nonzero probability for some input to G,
  $h(t, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)$

- G generates an index i while simultaneously generating a value t that allows f_i to be inverted quickly.
- Condition 2 says that f_i is hard to invert in the absence of t.

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$
- For every w of length n, and every output $\langle i, t \rangle$ of G that occurs with nonzero probability for some input to G,
  $h(t, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)$

- G generates an index i while simultaneously generating a value t that allows f_i to be inverted quickly.
- Condition 2 says that f_i is hard to invert in the absence of t.
- Condition 3 says that f_i is easy to invert when t is known.

Trapdoor function

Definition

A trapdoor function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a length-preserving indexing function that has an auxiliary probabilistic polynomial time TM G and an auxiliary function $h : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:

- Functions f and h are computable in polynomial time
- For every probabilistic polynomial time TM E, every k, and sufficiently large n, if we pick a random output $\langle i, t \rangle$ of G on 1^n and a random $w \in \Sigma^n$, then
  $Pr_{E,w}[E(i, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)] \leq n^{-k}$
- For every w of length n, and every output $\langle i, t \rangle$ of G that occurs with nonzero probability for some input to G,
  $h(t, f_i(w)) = y, \text{ where } f_i(y) = f_i(w)$

- G generates an index i while simultaneously generating a value t that allows f_i to be inverted quickly.
- Condition 2 says that f_i is hard to invert in the absence of t.
- Condition 3 says that f_i is easy to invert when t is known.
- Function h is the inverting function.

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.
2. Compute $N = pq$ and the value $\phi(N) = (p-1)(q-1)$.

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.
2. Compute $N = pq$ and the value $\phi(N) = (p-1)(q-1)$.
3. Select randomly a number e between 1 and $\phi(N)$ that is relatively prime to $\phi(N)$.

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.
2. Compute $N = pq$ and the value $\phi(N) = (p-1)(q-1)$.
3. Select randomly a number e between 1 and $\phi(N)$ that is relatively prime to $\phi(N)$.
4. Compute $d = e^{-1} \text{mod } \phi(N)$, i.e., $de \equiv_{\phi(N)} 1$

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.
2. Compute $N = pq$ and the value $\phi(N) = (p-1)(q-1)$.
3. Select randomly a number e between 1 and $\phi(N)$ that is relatively prime to $\phi(N)$.
4. Compute $d = e^{-1} \text{mod } \phi(N)$, i.e., $de \equiv_{\phi(N)} 1$
5. Output $((N, e), d)$, where is (N, e) the public key and d the is private key

Trapdoor function: example

The trapdoor function that underlies the well-known RSA cryptosystem.

G on input 1^n: generator machine G

1. Select randomly two prime numbers p, q of size n.
2. Compute $N = pq$ and the value $\phi(N) = (p-1)(q-1)$.
3. Select randomly a number e between 1 and $\phi(N)$ that is relatively prime to $\phi(N)$.
4. Compute $d = e^{-1} \text{mod } \phi(N)$, i.e., $de \equiv_{\phi(N)} 1$
5. Output $((N, e), d)$, where is (N, e) the public key and d the is private key

- The trapdoor function f: $f_{N,e}(w) = w^e \pmod{N}$
- The inverting function h: $h(d, x) = x^d \pmod{N}$

$$
\begin{aligned}
h(d, f_{N,e}(w)) &= (w^e \pmod{N})^d \pmod{N} \\
&= w^{de} \pmod{N} = w
\end{aligned}
$$