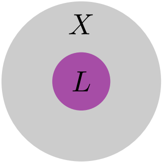
Design and Analysis of Algorithm
Basics of Complexity Theory

Outline

Decision Problem

Decision Problem: recognition of a set of strings $L \subseteq X$

- X: a set of strings
- x: a string in X (each string corresponds to an instance)
- L: language (a subset of X satisfying some property)

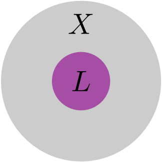


Task: Decide membership — if $x \in L$

Decision Problem

Decision Problem: recognition of a set of strings $L \subseteq X$

- X: a set of strings
- x: a string in X (each string corresponds to an instance)
- L: language (a subset of X satisfying some property)



Task: Decide membership — if $x \in L$

Example

- $X = \mathbb{N}$
- L are Primes $= \{2, 3, 5, 7, 11, 13, \dots\}$
- decide if x is a prime.

Motivation for Complexity Theory

We always want to know if a given problem can be *efficiently* solved by an algorithm.

Motivation for Complexity Theory

We always want to know if a given problem can be *efficiently* solved by an algorithm.

1. Precisely model algorithms
   - What is computation?
   - What is computable?

2. Precisely define what does it means for efficient.

Outline

Turing Machine

1936, London Mathematical Society: On computable numbers, with an application to the Entscheidungs Problem.

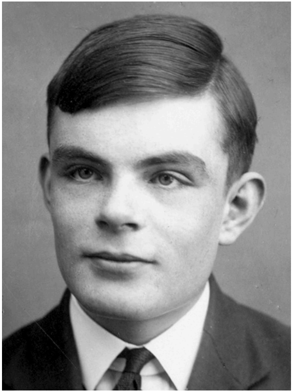
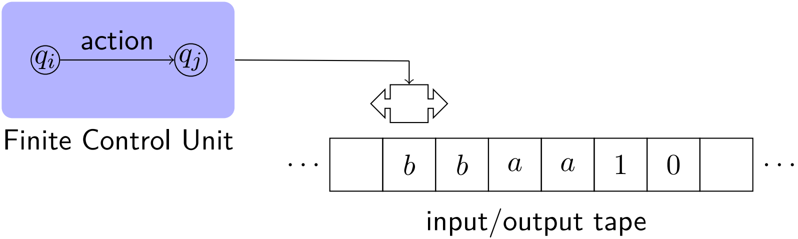


Figure: Alan Turing

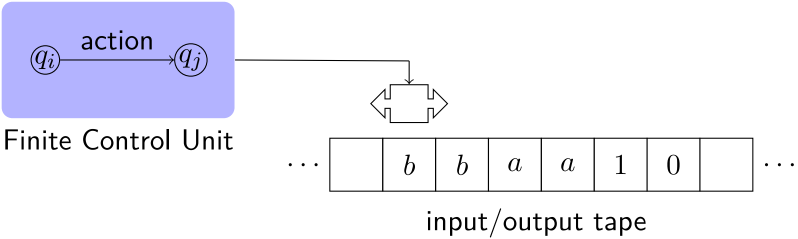
Turing Machine

Turing machine: automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.



Turing Machine

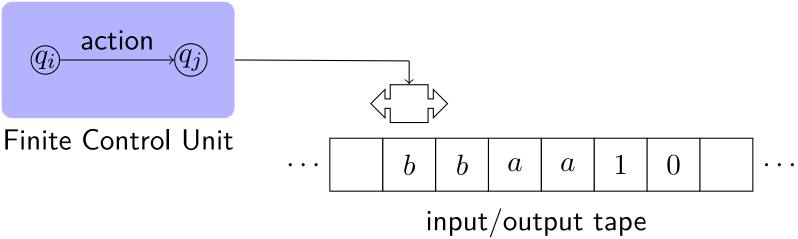
Turing machine: automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.



- At the beginning, the tape contains the input in several cells. Other places are empty.

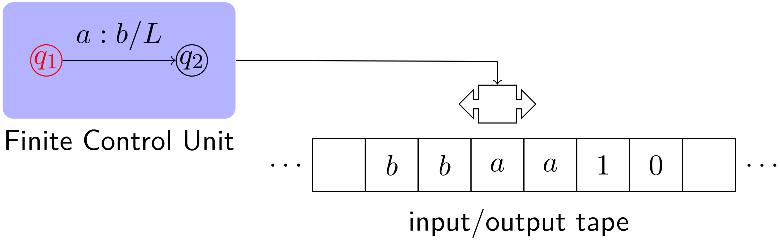
Turing Machine

Turing machine: automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.



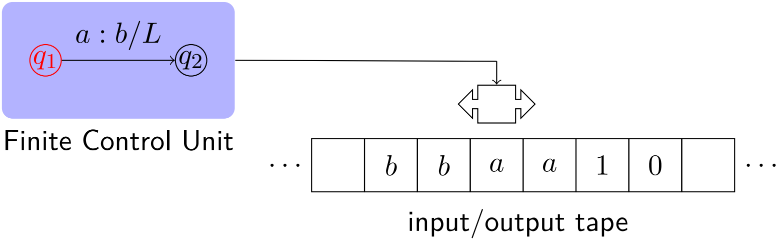
- At the beginning, the tape contains the input in several cells. Other places are empty.
- During computation, the control unit monitor current state and the head value, can do the following operations:
  1. wipe off old value and write new values
  2. change the current state
  3. move head left or right

An Example



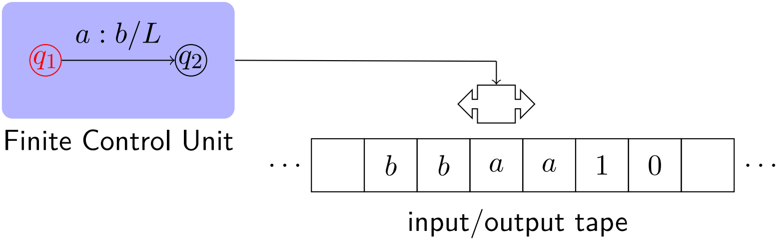
$a : b/L$
q_1
q_2
Finite Control Unit
$\cdots$
b
b
a
a
1
0
$\cdots$
input/output tape

An Example

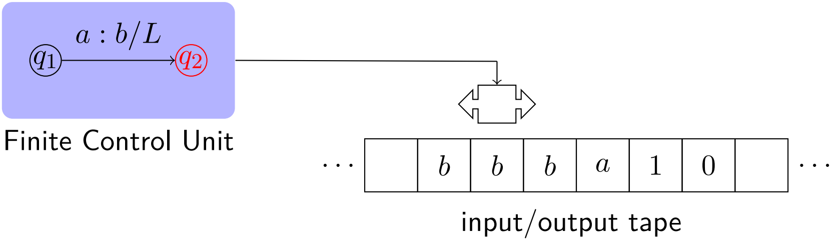


- $a \to b$; move left; current state $q_1 \to q_2$

An Example



- $a \to b$; move left; current state $q_1 \to q_2$



Intuition of Turing Machine

Mimic how human being solve a problem

Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)

Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)
- TM is provided a tape, which contains infinite cells (paper)

Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)
- TM is provided a tape, which contains infinite cells (paper)
- a symbol can be scanned from a cell or printed to a cell (reading and writing)

Formal Definition

Definition 1 (Turing Machine)

TM consists $(Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$

- Q: a finite set of states
- Σ: input alphabets
- Γ: working alphabets (including $\perp$, $\Sigma \subseteq \Gamma$)
- q_0: the initial state of Q;
- $q_{\mathsf{acc}}, q_{\mathsf{rej}}$: accept and reject state of Q
- δ: transition function

$$\delta : (Q \backslash \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Running Time of TM

Definition 2

We denote the running time of TM by $t_M(n)$, which is the maximum steps that TM runs on all inputs of length n

Polynomial Time

$$\bigcup_{k\in\mathbb{N}} \mathsf{TIME}(n^k)$$

The Extended Church-Turing Thesis

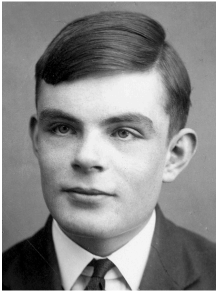
 

Figure: Alonzo Church & Alan Turing

Everyone's intuition of **Efficient** Algorithms = **Polynomial-Time** deterministic TMs

Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

1. NDTM may proceed according to several possible transitions
   - We can assume that every configuration leads to two possible configurations.

Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

1. NDTM may proceed according to several possible transitions
   - We can assume that every configuration leads to two possible configurations.
2. NDTM *accepts* iff there *exists* at least one of its branches accepts.

Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

1. NDTM may proceed according to several possible transitions
   - We can assume that every configuration leads to two possible configurations.
2. NDTM *accepts* iff there *exists* at least one of its branches accepts.

NDTM doesn't really correspond to any real-world physical model, it's just a theoretical construction.

Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

1. NDTM may proceed according to several possible transitions
   - We can assume that every configuration leads to two possible configurations.
2. NDTM *accepts* iff there *exists* at least one of its branches accepts.

NDTM doesn't really correspond to any real-world physical model, it's just a theoretical construction.

Non-determinism doesn't give TM any power to recognize more languages.

- Any NDTM can be simulated by a TM (with potentially exponential time overhead) by trying all branches of the NDTM machine "in parallel" by using BFS.

Notes

Some important facts about TM

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0,1\}^*$.

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0, 1\}^*$.
2. Any $\{0, 1\}^*$ represents some TM.

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0,1\}^*$.
2. Any $\{0,1\}^*$ represents some TM.
3. Any TM has many representations in the form of $\{0,1\}^*$.

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0,1\}^*$.
2. Any $\{0,1\}^*$ represents some TM.
3. Any TM has many representations in the form of $\{0,1\}^*$.

Why TMs are so powerful?

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0,1\}^*$.
2. Any $\{0,1\}^*$ represents some TM.
3. Any TM has many representations in the form of $\{0,1\}^*$.

Why TMs are so powerful?

- TM has a working tape (好记性不如烂笔头)

Notes

Some important facts about TM

1. Any TM can be encoded as $\{0,1\}^*$.
2. Any $\{0,1\}^*$ represents some TM.
3. Any TM has many representations in the form of $\{0,1\}^*$.

Why TMs are so powerful?

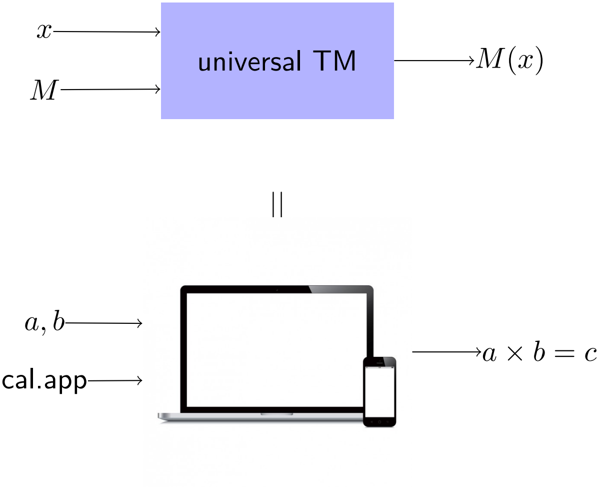
- TM has a working tape (好记性不如烂笔头)
- TM itself can be treated as data! TM can take another TM as its input.

Universal TM

Universal TM

$$||$$

Universal TM



Outline

Outline

1. Decision Problem
2. Deterministic Computation
3. Several Important Complexity Classes
   - $\mathcal{P}$ vs. $\mathcal{NP}$
   - $\mathcal{NP}$-complete
4. Randomized Computation
   - $\mathcal{BPP}$

We have introduced

- the notion of decision problems $L \subseteq X$
- the concept of (non-deterministic) TM

Time Complexity Hierarchy: $\mathcal{P}$ and $\mathcal{NP}$

We have introduced

- the notion of decision problems $L \subseteq X$
- the concept of (non-deterministic) TM

We say a TM M accepts ("1" signify acceptance) a language L if:

$$x \in L \iff M(x) = 1$$

- L is decidable by M (M solves L)

We have introduced

- the notion of decision problems $L \subseteq X$
- the concept of (non-deterministic) TM

We say a TM M accepts ("1" signify acceptance) a language L if:

$$x \in L \iff M(x) = 1$$

- L is decidable by M (M solves L)

Next, we introduce two important sets of problems, characterized by time complexity by DTM and NDTM:

$\mathcal{P}$ and $\mathcal{NP}$

$\mathcal{P}$ Complexity

Definition 3 ($\mathcal{P}$ Language)

$L \in \mathcal{P}$ if there exists a deterministic poly-time TM M such that

$$M(x) = 1 \iff x \in L$$

$\mathcal{P}$ Complexity

Definition 3 ($\mathcal{P}$ Language)

$L \in \mathcal{P}$ if there exists a deterministic poly-time TM M such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most $p(n)$ steps, where $p(\cdot)$ is some polynomial and n is the length of input

$\mathcal{P}$ Complexity

Definition 3 ($\mathcal{P}$ Language)

$L \in \mathcal{P}$ if there exists a deterministic poly-time TM M such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most $p(n)$ steps, where $p(\cdot)$ is some polynomial and n is the length of input

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$$

$\mathcal{P}$ Complexity

Definition 3 ($\mathcal{P}$ Language)

$L \in \mathcal{P}$ if there exists a deterministic poly-time TM M such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most $p(n)$ steps, where $p(\cdot)$ is some polynomial and n is the length of input

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$$

Example of $\mathcal{P}$ Languages

- $L = \{\text{even integers}\}$, M just need to check if the last bit is 0.
- $L = \mathsf{PRIME}$, M is the AKS primality test algorithm.

$\mathcal{NP}$ Complexity

Definition 4 ($\mathcal{NP}$ Languages - Conventional)

$L \in \mathcal{NP}$ if there exists a non-deterministic poly-time TM M:

$$x \in L \iff M(x) = 1$$

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k)$$

Definition 4 ($\mathcal{NP}$ Languages - Conventional)

$L \in \mathcal{NP}$ if there exists a non-deterministic poly-time TM M:

$$x \in L \iff M(x) = 1$$

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k)$$

Alert

$\mathcal{NP}$ means non-deterministic poly-time, not non-poly-time!

Modern Definition

Definition 5 ($\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$ if there exists a deterministic poly-time TM M:

$$x \in L \iff \exists w \in \{0,1\}^{\mathsf{poly}(n)} \text{ s.t. } M(x,w) = 1$$

Modern Definition

Definition 5 ($\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$ if there exists a deterministic poly-time TM M:

$$x \in L \iff \exists w \in \{0,1\}^{\mathsf{poly}(n)} \text{ s.t. } M(x,w) = 1$$

- If $M(x,w) = 1$, w is called as a "witness" for $x \in L$. One can think of w as an efficiently-verifiable "certificate" or "proof" for $x \in L$.

Modern Definition

Definition 5 ($\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$ if there exists a deterministic poly-time TM M:

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(n)} \text{ s.t. } M(x,w) = 1$$

- If $M(x,w) = 1$, w is called as a "witness" for $x \in L$. One can think of w as an efficiently-verifiable "certificate" or "proof" for $x \in L$.
- We require w is *short*, i.e, $|w| = \text{poly}(n)$. This is natural since M runs in poly-time in n.

Modern Definition

Definition 5 ($\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$ if there exists a deterministic poly-time TM M:

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(n)} \text{ s.t. } M(x,w) = 1$$

- If $M(x,w) = 1$, w is called as a "witness" for $x \in L$. One can think of w as an efficiently-verifiable "certificate" or "proof" for $x \in L$.
- We require w is *short*, i.e, $|w| = \text{poly}(n)$. This is natural since M runs in poly-time in n.

$L \in \mathcal{NP}$ iff there are *short proofs* for membership in L.

Modern Definition

Definition 5 ($\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$ if there exists a deterministic poly-time TM M:

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(n)} \text{ s.t. } M(x,w) = 1$$

- If $M(x,w) = 1$, w is called as a "witness" for $x \in L$. One can think of w as an efficiently-verifiable "certificate" or "proof" for $x \in L$.
- We require w is *short*, i.e, $|w| = \text{poly}(n)$. This is natural since M runs in poly-time in n.

$L \in \mathcal{NP}$ iff there are *short proofs* for membership in L.

Equivalence between traditional and modern definitions

- Even though M is a deterministic machine, its second argument w captures the nondeterminism in the definition.

Examples of $\mathcal{NP}$ Language - Composites

$L =$ COMPOSITE

- instance x is an integer
- a witness w for $x \in L$ is a non-trivial factor of x
- M just need to check if w divides x, which could be done in polynomial time.

Examples of $\mathcal{NP}$ Language - Composites

$L =$ COMPOSITE

- instance x is an integer
- a witness w for $x \in L$ is a non-trivial factor of x
- M just need to check if w divides x, which could be done in polynomial time.

Example of COMPOSITE

- instance: 15
- witness: 3, 5

Examples of $\mathcal{NP}$ Language - Composites

$L =$ COMPOSITE

- instance x is an integer
- a witness w for $x \in L$ is a non-trivial factor of x
- M just need to check if w divides x, which could be done in polynomial time.

Example of COMPOSITE

- instance: 15
- witness: 3, 5

In fact, COMPOSITE also belong to $\mathcal{P}$ (think why?)

Examples of $\mathcal{NP}$ Language - SAT and 3-SAT

SAT: Given a CNF formula Φ, check if it has a satisfying truth assignment.

3-SAT: SAT where each clause contains exactly 3 literals

witness: an assignment of truth values to the Boolean variables

Examples of $\mathcal{NP}$ Language - SAT and 3-SAT

SAT: Given a CNF formula Φ, check if it has a satisfying truth assignment.

3-SAT: SAT where each clause contains exactly 3 literals

witness: an assignment of truth values to the Boolean variables

Example of 3-SAT

- instance $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
- witness: $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$

Examples of $\mathcal{NP}$ Language - Hamilton Path

Hamilton Graph: Given an undirected graph $G = (V, E)$, does there exists a simple path that visits every node?

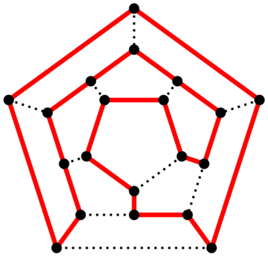


Figure: Hamiltonian Graph (a path traverses through each verticals exactly once)

witness: a path

M check if the path contains each node in V exactly once

As per definition, $\mathcal{P} \subseteq \mathcal{NP}$. Because $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$:

- $M'(x, w)$ can always sets $w = \perp$ and decide whether $x \in L$ using M.
- Alternatively, "short" M can be viewed as a witness for $x \in L$. Think about why the description of M is short?

$\mathcal{P}$ vs. $\mathcal{NP}$

As per definition, $\mathcal{P} \subseteq \mathcal{NP}$. Because $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$:

- $M'(x, w)$ can always sets $w = \perp$ and decide whether $x \in L$ using M.
- Alternatively, "short" M can be viewed as a witness for $x \in L$. Think about why the description of M is short?

$\mathcal{P}$ = the set of decision problems whose all instances can be efficiently decided
$\mathcal{NP}$ = the set of decision problems whose yes instances can be efficiently decided with short proofs

$\mathcal{P}$ vs. $\mathcal{NP}$

As per definition, $\mathcal{P} \subseteq \mathcal{NP}$. Because $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$:

- $M'(x, w)$ can always sets $w = \perp$ and decide whether $x \in L$ using M.
- Alternatively, "short" M can be viewed as a witness for $x \in L$. Think about why the description of M is short?

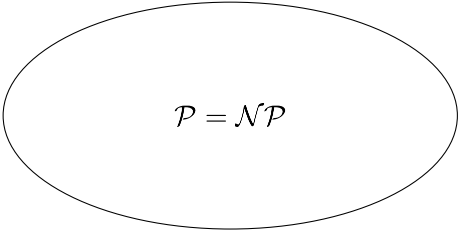
$\mathcal{P}$ = the set of decision problems whose all instances can be efficiently decided
$\mathcal{NP}$ = the set of decision problems whose yes instances can be efficiently decided with short proofs

1971: Cook, Edmonds, Levin, Yablonski, Gödel

Perhaps the most prominent question in TCS:

$$\mathcal{P} =? \mathcal{NP}$$

$\mathcal{P} = \mathcal{NP}$



If $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!

If $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!

Cryptography as we know it may be impossible. Cryptographic researchers are out of job.

If $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!

Cryptography as we know it may be impossible. Cryptographic researchers are out of job.

In principle, every aspect of life could be efficiently and globally optimized $\cdots$

$\cdots$ life as we know it would be different!

The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

Let $f : \{0,1\}^n \rightarrow \{0,1\}^m$. To efficiently find a pre-image x of y, the idea is to determine x bit-by-bit. $f(x_1 || \cdots || x_n) = y$.

The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

Let $f : \{0,1\}^n \to \{0,1\}^m$. To efficiently find a pre-image x of y, the idea is to determine x bit-by-bit. $f(x_1||\cdots||x_n) = y$.

Define a collection of languages $L_i = \{(y,z)|\exists w \text{ s.t. } y = f(z||w)\}$, where $z \in \{0,1\}^i$, $w \in \{0,1\}^{n-i}$

- clearly $L_i \in \mathcal{NP}$ and thus also belong to $\mathcal{P}$ by assumption, we define algorithm Invert as:

The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

Let $f : \{0,1\}^n \to \{0,1\}^m$. To efficiently find a pre-image x of y, the idea is to determine x bit-by-bit. $f(x_1||\cdots||x_n) = y$.

Define a collection of languages $L_i = \{(y, z) | \exists w \text{ s.t. } y = f(z||w)\}$, where $z \in \{0,1\}^i$, $w \in \{0,1\}^{n-i}$

- clearly $L_i \in \mathcal{NP}$ and thus also belong to $\mathcal{P}$ by assumption, we define algorithm Invert as:

Algorithm 4: Invert(y)

1: $z = \epsilon$;
2: **for** $i \leftarrow 1$ **to** n **do**
3:     **if** $(y, z||0) \in L_i$ **then** $z = z||0$;
4:     **else** $z = z||1$;
5: **end**
6: **return** z

The Reverse Direction

OWF exists $\Rightarrow \mathcal{P} \neq \mathcal{NP}$

- We have many candidates of OWFs, but they require assumptions.

The Reverse Direction

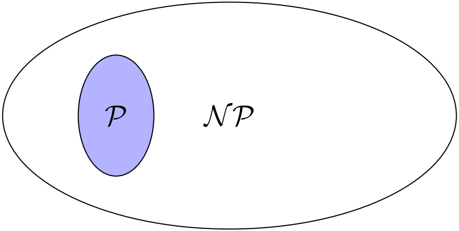
OWF exists $\Rightarrow \mathcal{P} \neq \mathcal{NP}$

- We have many candidates of OWFs, but they require assumptions.

Warning

OWFs do not exist *does not imply* $\mathcal{P} = \mathcal{NP}$

Consensus



Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

- Because $\mathcal{NP}$ contains many languages that are not believed to be in $\mathcal{P}$, such as: 3-SAT and Hamiltonian graph.

Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

- Because $\mathcal{NP}$ contains many languages that are not believed to be in $\mathcal{P}$, such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with n variables?

Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

- Because $\mathcal{NP}$ contains many languages that are not believed to be in $\mathcal{P}$, such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with n variables?

Exhaustive search: try all 2^n truth assignments.

Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

- Because $\mathcal{NP}$ contains many languages that are not believed to be in $\mathcal{P}$, such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with n variables?

Exhaustive search: try all 2^n truth assignments.

Q: Can we do anything substantially more clever?

Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion: $\mathcal{P} \neq \mathcal{NP}$

- Because $\mathcal{NP}$ contains many languages that are not believed to be in $\mathcal{P}$, such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with n variables?

Exhaustive search: try all 2^n truth assignments.

Q: Can we do anything substantially more clever?

Conjecture: $\underbrace{\text{No poly-time algorithm}}_{\text{intractable}}$ for 3-SAT

Outline

1. Decision Problem
2. Deterministic Computation
3. Several Important Complexity Classes
   - $\mathcal{P}$ vs. $\mathcal{NP}$
   - $\mathcal{NP}$-complete
4. Randomized Computation
   - $\mathcal{BPP}$

Motivation of Reduction

$\mathcal{NP}$ is the set of many problems.

How to figure out the relations among them?

A central approach is finding reductions

Polynomial Time Reducibility

Language L' is *poly-time reducible* or *reduces* to language L, written as $L' \leq_p L$, if there is a determinstic poly-time function $\mathcal{R} : L' \to L$ so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

Polynomial Time Reducibility

Language L' is *poly-time reducible* or *reduces* to language L, written as $L' \leq_p L$, if there is a determinstic poly-time function $\mathcal{R} : L' \to L$ so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$ is called a poly-time reduction from L to L'.

Polynomial Time Reducibility

Language L' is *poly-time reducible* or *reduces* to language L, written as $L' \leq_p L$, if there is a determinstic poly-time function $\mathcal{R} : L' \rightarrow L$ so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$ is called a poly-time reduction from L to L'.

$L' \leq_p L$ implies L' is not harder than L

Polynomial Time Reducibility

Language L' is *poly-time reducible* or *reduces* to language L, written as $L' \leq_p L$, if there is a determinstic poly-time function $\mathcal{R} : L' \to L$ so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$ is called a poly-time reduction from L to L'.

$L' \leq_p L$ implies L' is not harder than L

We should pay attention to:

- the direction of $\mathcal{R}$
- the time complexity of $\mathcal{R}$

Definition 6 ($\mathcal{NP}$-Hard)

L is said to be $\mathcal{NP}$-hard if for every $\mathcal{NP}$-language L', there is a deterministic poly-time algorithm (a reduction) $\mathcal{R}$:

$$x \in L' \iff \mathcal{R}(x) \in L.$$

$\mathcal{NP}$-Hard

Definition 6 ($\mathcal{NP}$-Hard)

L is said to be $\mathcal{NP}$-hard if for every $\mathcal{NP}$-language L', there is a deterministic poly-time algorithm (a reduction) $\mathcal{R}$:

$$x \in L' \iff \mathcal{R}(x) \in L.$$

- We can interpret that the languages in $\mathcal{NP}$ is not harder than that in $\mathcal{NP}$-hard.

Definition 6 ($\mathcal{NP}$-Hard)

L is said to be $\mathcal{NP}$-hard if for every $\mathcal{NP}$-language L', there is a deterministic poly-time algorithm (a reduction) $\mathcal{R}$:

$$x \in L' \iff \mathcal{R}(x) \in L.$$

- We can interpret that the languages in $\mathcal{NP}$ is not harder than that in $\mathcal{NP}$-hard.

Fact: languages in $\mathcal{NP}$-hard may not fall in $\mathcal{NP}$.

$\mathcal{NP}$-Complete

Definition 7 ($\mathcal{NP}$-Complete)

L is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard, and is itself in $\mathcal{NP}$.

$\mathcal{NP}$-Complete

Definition 7 ($\mathcal{NP}$-Complete)

L is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard, and is itself in $\mathcal{NP}$.

Definition Intuition: $\mathcal{NP}$-complete represents the set of hardest problems in $\mathcal{NP}$.

$\mathcal{NP}$-Complete

Definition 7 ($\mathcal{NP}$-Complete)

L is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard, and is itself in $\mathcal{NP}$.

Definition Intuition: $\mathcal{NP}$-complete represents the set of hardest problems in $\mathcal{NP}$.

- We can solve all problems in $\mathcal{NP}$ if we find an efficient algorithm for any problems in $\mathcal{NP}$-complete.

Theorem 8

Suppose $Y \in \mathcal{NP}$*-complete, then* $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$.

Theorem 8

Suppose $Y \in \mathcal{NP}$-complete, then $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$.

$\Leftarrow$: $Y \in \mathcal{NP}$-complete and thus of course $Y \in \mathcal{NP}$. Now suppose $\mathcal{P} = \mathcal{NP}$, we have $Y \in \mathcal{P}$.

Theorem 8

Suppose $Y \in \mathcal{NP}$*-complete, then* $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$*.*

$\Leftarrow$: $Y \in \mathcal{NP}$-complete and thus of course $Y \in \mathcal{NP}$. Now suppose $\mathcal{P} = \mathcal{NP}$, we have $Y \in \mathcal{P}$.

$\Rightarrow$: $\forall X \in \mathcal{NP}$, $X \leq_p Y$ becuase $Y \in \mathcal{NP}$-complete. Now suppose $Y \in \mathcal{P}$, we further have $\mathcal{NP} \subseteq \mathcal{P}$. We already know $\mathcal{P} \subseteq \mathcal{NP}$, thus $\mathcal{P} = \mathcal{NP}$.

Theorem 8

Suppose $Y \in \mathcal{NP}$-complete, then $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$.

$\Leftarrow$: $Y \in \mathcal{NP}$-complete and thus of course $Y \in \mathcal{NP}$. Now suppose $\mathcal{P} = \mathcal{NP}$, we have $Y \in \mathcal{P}$.

$\Rightarrow$: $\forall X \in \mathcal{NP}$, $X \leq_p Y$ becuase $Y \in \mathcal{NP}$-complete. Now suppose $Y \in \mathcal{P}$, we further have $\mathcal{NP} \subseteq \mathcal{P}$. We already know $\mathcal{P} \subseteq \mathcal{NP}$, thus $\mathcal{P} = \mathcal{NP}$.

- This theorem essentially states that if $\mathcal{P} \cap \mathcal{NPC}$ is non-empty iff $\mathcal{P} = \mathcal{NP}$.

$\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still): $\mathcal{P} \neq \mathcal{NP}$.

$\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still): $\mathcal{P} \neq \mathcal{NP}$.

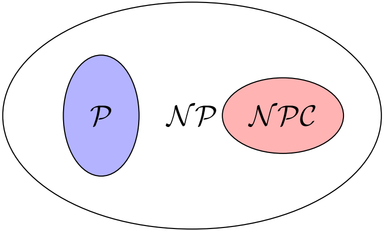


Figure: $\mathcal{P} \neq \mathcal{NP}$

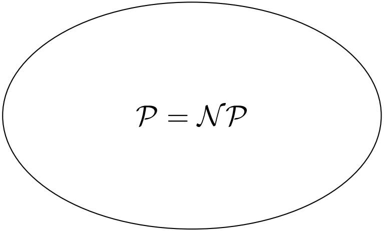


Figure: $\mathcal{P} = \mathcal{NP}$

$\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still): $\mathcal{P} \neq \mathcal{NP}$.

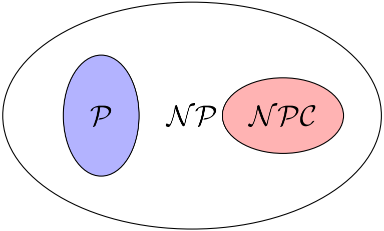


Figure: $\mathcal{P} \neq \mathcal{NP}$

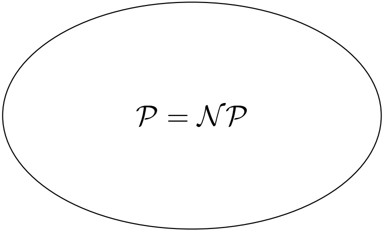
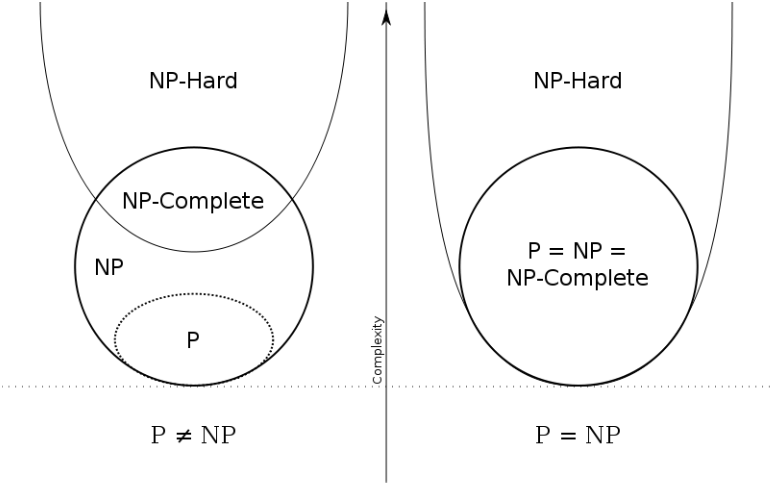


Figure: $\mathcal{P} = \mathcal{NP}$

Why we believe $\mathcal{P} \neq \mathcal{NP}$? Because some problems appear significantly harder.


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Outline

Motivation of Randomized Algorithm

TM models deterministic algorithms.

TM does not seem to capture one aspect of reality — the ability to make random choices during computation

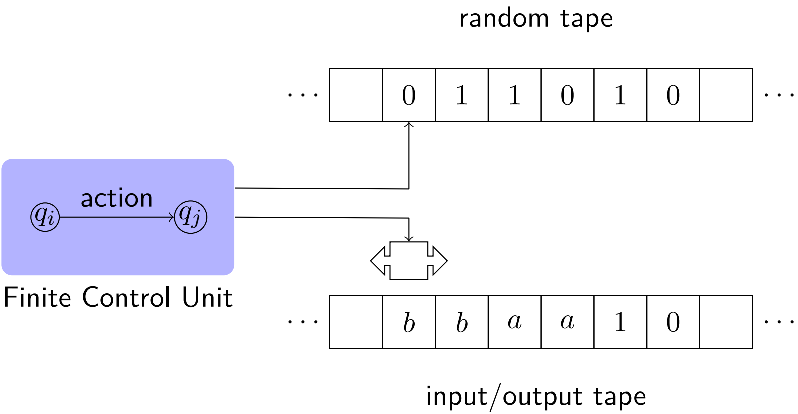
- Most programming languages provide a built-in RNG.

It makes sense to consider algorithms that can toss a coin, a.k.a. use a source of random bits. Such algorithms have been implicitly studied for a long time.

- estimate facts about a large sample by taking a small sample
- simulate real-world systems that are themselves probabilistic, such as nuclear fission and the stock market
- differential equations

Probabilistic Turing Machine

Probabilistic Polynomial-time TM models probabilistic algorithm.



PTM vs. NDTM

NDTM is a TM with two transition functions. PTM is syntactically similar.

The difference is in how we interpret the working of TM.

- In a PTM, each transition is taken with probability $1/2$, a computation that runs for time t gives rise 2^t branches in the graph of all computations, each of which is taken with probability $1/2^t$. $\Pr[M(x) = 1]$ is simply the *fraction* of branches that end with M outputting a 1.
- In a NDTM, $M(x) = 1$ iff there exists a branch that outputs 1

On a conceptual level, PTM and NDTM are very different

- PTM like TM and unlike NDTM, is intended to model realistic computation devices.

Outline

Bounded-Error Probabilistic Polynomial Time

Definition 9 ($\mathcal{BPP}$ Complexity)

$L \in \mathcal{BPP}$ iff there exists a probabilistic polynomial time TM M such that:

$$\forall x \in L : \quad \Pr[M(x) = 1] \geq \alpha$$
$$\forall x \notin L : \quad \Pr[M(x) = 1] \leq \beta$$

Bounded-Error Probabilistic Polynomial Time

Definition 9 ($\mathcal{BPP}$ Complexity)

$L \in \mathcal{BPP}$ iff there exists a probabilistic polynomial time TM M such that:

$$\forall x \in L: \quad \Pr[M(x) = 1] \geq \alpha$$
$$\forall x \notin L: \quad \Pr[M(x) = 1] \leq \beta$$

Bounded-error Probabilistic Polynomial Time (weak version)

- A typical choices is $\alpha = 2/3$, $\beta = 1/3$. In this case, the class of decision problems solvable by a probabilistic TM in polynomial time with an error probability e bounded away from $1/3$ for all instances

Reduce the Error (1/2)

In practice, an error probability of $1/3$ might not be acceptable.

Reduce the Error (1/2)

In practice, an error probability of $1/3$ might not be acceptable.

However, the choice of $1/3$ could be arbitrary and the set $\mathcal{BPP}$ will be unchanged.

Reduce the Error (1/2)

In practice, an error probability of $1/3$ might not be acceptable.

However, the choice of $1/3$ could be arbitrary and the set $\mathcal{BPP}$ will be unchanged.

- It can be any constant between $(0, 1/2)$.

Reduce the Error (1/2)

In practice, an error probability of $1/3$ might not be acceptable.

However, the choice of $1/3$ could be arbitrary and the set $\mathcal{BPP}$ will be unchanged.

- It can be any constant between $(0, 1/2)$.
- It does not even have to be constant: e could be as high as $1/2 - n^{-c}$ on one hand, or as small as 2^{-n^c} on the other hand, where c is any positive constant, and n is the length of input.

Reduce the Error (1/2)

In practice, an error probability of $1/3$ might not be acceptable.

However, the choice of $1/3$ could be arbitrary and the set $\mathcal{BPP}$ will be unchanged.

- It can be any constant between $(0, 1/2)$.
- It does not even have to be constant: e could be as high as $1/2 - n^{-c}$ on one hand, or as small as 2^{-n^c} on the other hand, where c is any positive constant, and n is the length of input.
- The idea is if the algorithm is run many times, the chance that the majority of the runs are wrong drops off exponentially as a consequence of the Chernoff bound.

Reduce the Error (2/2)

This makes it possible to create a highly accurate algorithm by merely running the algorithm several times and taking a “majority vote” of the answers.

Reduce the Error (2/2)

This makes it possible to create a highly accurate algorithm by merely running the algorithm several times and taking a "majority vote" of the answers.

Chernoff Bounds (Lower Tail): Let $X = \sum_{i=1}^{n} X_i$, $\Pr[X_i] = p$, $\mu = \mathbb{E}(X) = np$.

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2} \text{ for all } 0 \leq \delta < 1$$

Reduce the Error (2/2)

This makes it possible to create a highly accurate algorithm by merely running the algorithm several times and taking a "majority vote" of the answers.

Chernoff Bounds (Lower Tail): Let $X = \sum_{i=1}^{n} X_i$, $\Pr[X_i] = p$, $\mu = \mathbb{E}(X) = np$.

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2} \text{ for all } 0 \leq \delta < 1$$

Do the Majority Vote, i.e., set $(1-\delta)\mu = n/2$ and thus $\delta = 1 - 1/2p$, we obtain:

$$\Pr[X \leq n/2] \leq e^{-n\frac{(1-2p)^2}{8p}}$$

About $\mathcal{BPP}$

$\mathcal{BPP}$ is one of the largest practical class of problems, since problems in $\mathcal{BPP}$ have efficient probabilistic algorithms that can be run quickly on real modern machines.

About $\mathcal{BPP}$

$\mathcal{BPP}$ is one of the largest practical class of problems, since problems in $\mathcal{BPP}$ have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly, $\mathcal{BPP} \supseteq \mathcal{P}$, since a deterministic machine is a special case of a probabilistic machine.

About $\mathcal{BPP}$

$\mathcal{BPP}$ is one of the largest practical class of problems, since problems in $\mathcal{BPP}$ have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly, $\mathcal{BPP} \supseteq \mathcal{P}$, since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in $\mathcal{BPP}$ but not known to be in $\mathcal{P}$. The number of such problems is decreasing, and it is conjectured that $\mathcal{P} = \mathcal{BPP}$.

About $\mathcal{BPP}$

$\mathcal{BPP}$ is one of the largest practical class of problems, since problems in $\mathcal{BPP}$ have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly, $\mathcal{BPP} \supseteq \mathcal{P}$, since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in $\mathcal{BPP}$ but not known to be in $\mathcal{P}$. The number of such problems is decreasing, and it is conjectured that $\mathcal{P} = \mathcal{BPP}$.

For a long time, one of the most famous problems that was known to be in $\mathcal{BPP}$ but not known to be in $\mathcal{P}$ was the PRIME.

About $\mathcal{BPP}$

$\mathcal{BPP}$ is one of the largest practical class of problems, since problems in $\mathcal{BPP}$ have efficient probabilistic algorithms that can be run quickly on real modern machines.

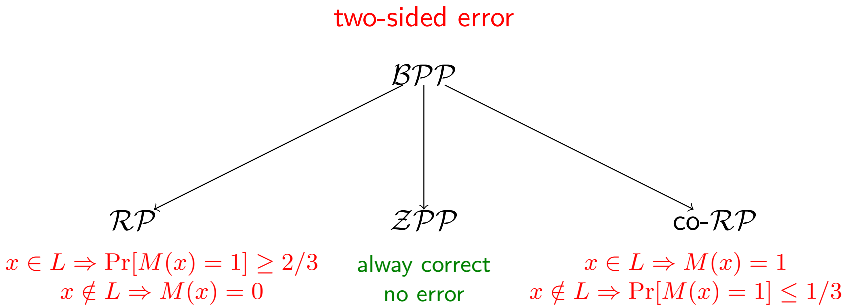
- Clearly, $\mathcal{BPP} \supseteq \mathcal{P}$, since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in $\mathcal{BPP}$ but not known to be in $\mathcal{P}$. The number of such problems is decreasing, and it is conjectured that $\mathcal{P} = \mathcal{BPP}$.

For a long time, one of the most famous problems that was known to be in $\mathcal{BPP}$ but not known to be in $\mathcal{P}$ was the PRIME.

[Agrawal, Kayal, Saxena 2002]: gave a deterministic polynomial-time algorithm for PRIME, thus showing that it is in $\mathcal{P}$.

One-sided and Zero-sided Error

$\mathcal{ZPP}$: probabilistic polynomial-time TM always returns correct YES or NO answer, or halts with low probability, a.k.a. running time is polynomial in expectation for every input

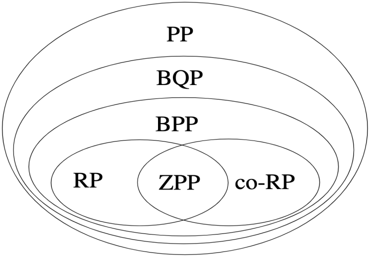


- $\mathcal{BPP}$: Monte Carlo algorithms (probabilistic) likely to be correct in strict polynomial running time
- $\mathcal{ZPP}$: Las Vegas algorithms (probabilistic) are always correct in expected polynomial running time

$\mathcal{BPP}$ in Relation to Other Probabilistic Complexity Classes

$\mathcal{BQP}$ (bounded-error quantum polynomial time): the class of decision problems solvable by a quantum TM in polynomial time with bounded error

- It is the quantum analogue of $\mathcal{BPP}$



Limits of $\mathcal{BPP}$

Consensus: $\mathcal{P} \subseteq \underline{\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}} \subseteq \mathcal{BPP} \subseteq \mathcal{NP}$

$\mathcal{P} \subseteq \mathcal{BPP}$

- An important example of a problem in $\mathcal{BPP}$ still not known to be in $\mathcal{P}$ is polynomial identity testing — determining whether a polynomial is identically equal to the zero polynomial, when you have access to the value of the polynomial for any given input, but not to the coefficients.

$\mathcal{BPP} \subseteq \mathcal{NP}$

- Adleman's theorem: $\mathcal{BPP} \subseteq P/\text{poly}$ (polynomial-size Boolean circuits)
- Karp-Levin theorem: $\mathcal{NP} \subseteq P/\text{poly} \Rightarrow \text{PH} = \sum_2^P$

Thus, $\mathcal{NP} \subseteq \mathcal{BPP}$ will imply collapse of PH, which is unlikely to be true. In other words, $\nexists$ bounded-error probabilistic algorithms for $\mathcal{NPC}$ problems.