Introduction to the Theory of Computation

Bowen Deng
Department of Mathematics
University of Science and Technology of China
dengbowen@mail.ustc.edu.cn

Instructor: Yuan Li (Neo Li)
yuan_li@fudan.edu.cn

Textbook: Introduction to the Theory of Computation
(Michael Sipser, 3rd edition)

Advanced Reading: Computational Complexity
(Arora & Barak)

Fudan University • 2025 FISS

Contents

0 Overview **2**

1 Big-O notation **3**

2 Alphabet and Language **7**

3 Finite Automaton **8**

4 Turing Machine **15**

5 Universal TM **20**

6 Reduction **24**

0 Overview

In ToC, we study

1. **Introduction:** Big-O notation, alphabets and languages.
2. **Models of computation:** Finite automaton, Turing machine.
3. **Computability:** What is computable?
4. **Complexity Theory:** What problems are efficiently solvable?

In complexity theory, we study how much resources (time space) are required to solve a problem.

`chicken and rabbit` / `in the same cage` / `10 animals, 32 legs` $\xrightarrow{\textbf{reduction}}$ `system of linear equations`

$$\begin{cases} x + y = 10 \\ 2x + 4y = 32 \end{cases}$$

A problem is considered efficiently solvable iff there exists a **polynomial-time** algorithm.
(e.g. $O(n), O(n^2), O(n^{100})$)

Central problem: P vs NP

$$\begin{cases} \textbf{P:} \text{ All problems that are solvable in polynomial time} \\ \textbf{NP:} \text{ All problems that can be verified in polynomial time} \end{cases}$$

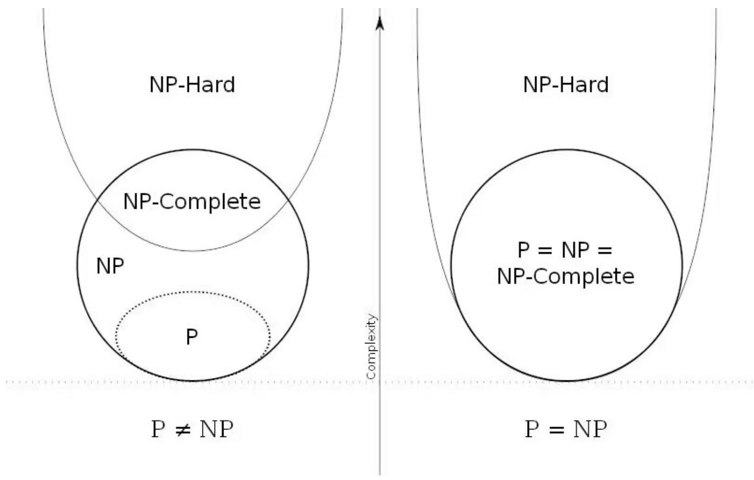

Figure 1: P $\neq$ NP or P = NP

Solving any **NP-complete** problem will solve all problems in NP!

1 Big-O notation

Why do we need Big-O notation?

1. **Analyze algorithm** (e.g. $10n^2 + 5n = O(n^2)$)
2. (Math) **Ignore low-order terms**

Def 1.1 Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = O_{x\to\infty}(g(x))$ iff

$$(\exists\, C > 0)(\exists\, N)(\forall\, x \geq N)(|f(x)| \leq C|g(x)|)$$

When there is no ambiguity, we write $f(x) = O(g(x))$ or $f = O(g)$.

Ex 1 $10n^2 + 5n = O(n^2)$

Proof $g(n) = n^2$, let $C = 15$ and $N = 1$, we have

$$|10n^2 + 5n| = 10n^2 + 5n \leq 10n^2 + 5n^2 = 15n^2 = 15g(n) \qquad \square$$

Ex 2 $T(n) = 6n^4 - 2n^3 + 5$, prove $T(n) = O(n^4)$.

Proof

$$\begin{aligned} |T(n)| &= |6n^4 - 2n^3 + 5| \\ &\leq 6n^4 + 2n^3 + 5 \\ &\leq 6n^4 + 2n^4 + 5n^4 \quad (n \geq 1) \\ &= 13n^4 \quad (C = 13) \end{aligned}$$

$$(\exists\, c = 13)(\exists\, N = 1)(\forall\, n \geq 1)(|T(n)| \leq 13n^4) \qquad \square$$

Ex 3 $1 + 2 + \cdots + n = O(n^2)$

Proof $|1 + 2 + \cdots + n| \leq n + n + \cdots + n$ (n terms) $= n^2$ $\square$

Remark Actually $1 + 2 + \cdots + n = \frac{1}{2}n(n+1) = \frac{1}{2}n^2 + \frac{1}{2}n = \frac{1}{2}n^2 + O(n)$.

That's how Big-O notation ignores the low-order term.

$O(1)$ is a **constant**. Let $g(x) = 1$ the condition is

$$(\exists\, C > 0)(\exists\, N)(\forall\, n \geq N)(|f(x)| \leq C)$$

For example, $100 = O(1)$ and also $100^{100^{100}} = O(1)$.

Ex 4 $1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n + O(1)$

Proof The result is equivalent to

$$1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n = O(1)$$

So we only need to prove

$$0 < 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n \leq 1$$

As is known $\int \frac{1}{x}dx = \ln x + C$, so $\int_1^n \frac{1}{x}dx = \ln n$.

$$\begin{aligned} 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n &= 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \int_1^n \frac{1}{x}dx \\ &= 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \int_1^2 \frac{1}{x}dx - \int_1^2 \frac{1}{x}dx - \cdots - -\int_1^2 \frac{1}{x}dx \\ &= 1 + (\frac{1}{2} - \int_1^2 \frac{1}{x}dx)(\leq 0) + (\frac{1}{3} - \int_2^3 \frac{1}{x}dx)(\leq 0) + \cdots + (\frac{1}{n} - \int_{n-1}^n \frac{1}{x}dx)(\leq 0) \\ &\leq 1 \end{aligned}$$

$$
\begin{aligned}
1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n &= 1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^n \frac{1}{x}dx \\
&= 1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^2 \frac{1}{x}dx-\int_1^2 \frac{1}{x}dx-\cdots-\int_1^2 \frac{1}{x}dx \\
&= (1-\int_1^2 \frac{1}{x}dx)(>0)+(\frac{1}{2}-\int_2^3 \frac{1}{x}dx)(>0)+\cdots+(\frac{1}{n-1}-\int_{n-1}^n \frac{1}{x}dx)(>0)+\frac{1}{n} \\
&\geq \frac{1}{n}>0
\end{aligned}
$$

So we prove $0<1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n\leq 1$ and prove the result. $\square$

Remark `"Think geometrically, Prove algebraically."` (John Tate)

<u>**Def 1.2**</u> Let $f,g:\mathbf{R}\to\mathbf{R}$. Write $f(x)=O_{x\to a}(g(x))$ iff

$$(\exists\, C>0)(\exists\, \delta>0)(\forall\, x\in[a-\delta,a+\delta])(|f(x)|\leq C|g(x)|)$$

Ex 5 $e^x=1+x+\frac{x^2}{2}+O_{x\to 0}(x^3)$

Proof When x approaches 0, by Taylor series

$$
\begin{aligned}
|e^x| &= |1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\cdots| \\
&\leq |1+x+\frac{x^2}{2!}|+|\frac{x^3}{3!}|+|\frac{x^4}{4!}|+|\frac{x^5}{5!}|+\cdots \\
&\leq |1+x+\frac{x^2}{2!}|+|\frac{x^3}{3!}|+|\frac{x^3}{4!}|+|\frac{x^3}{5!}|+\cdots \quad (\text{let } \delta=1,\ |x|\leq 1) \\
&\leq |1+x+\frac{x^2}{2!}|+O(x^3) \qquad \square
\end{aligned}
$$

Prop 1.3

1. If $f_1=O(g_1)$ and $f_2=O(g_2)$ then $f_1f_2=O(g_1g_2)$
2. $f\cdot O(g)=O(fg)$
3. If $f_1=O(g_1)$ and $f_2=O(g_2)$ then $f_1+f_2=O(\max(g_1,g_2))$
   In particular, if $f_1=O(g)$ and $f_2=O(g)$ then $f_1+f_2=O(g)$
4. If $f=O(g)$ and k is a constant, then $k\cdot f=O(g)$

Proof 1. Since $f_1=O(g_1)$ and $f_2=O(g_2)$, by Def 1.1 we have

$$(\exists\, C_1>0)(\exists\, x_1)(\forall\, x\geq x_1)(|f_1(x)|\leq C_1|g_1(x)|)$$

$$(\exists\, C_2>0)(\exists\, x_2)(\forall\, x\geq x_2)(|f_2(x)|\leq C_2|g_2(x)|)$$

Mutiply the two expressions

$$(\exists\, C_3=C_1C_2)(\exists\, x_3=\max(x_1,x_2)(\forall\, x\geq x_3)(|f_1(x)f_2(x)|\leq C_3|g_1(x)g_2(x)|) \qquad \square$$

<u>**Def 1.3**</u> Let $f:\mathbf{R^2}\to\mathbf{R}$ and $g:\mathbf{R}\to\mathbf{R}$. Write $f(x,y)=O_y(g(x))$ iff

$$(\forall\, y)(\exists\, c>0)(\exists\, N)(\forall\, x\geq N)(|f(x,y)|\leq C|g(x)|)$$

Remark `"After fixing y,` $f(x,y)$ `can be bounded by` $O(g(x))$`."`

$O(1)$ is an **absolute constant**, $O_\delta(1)$ is a **constant that depends on** δ.

Ex 6 For $\delta > 0$ we have

$$\frac{1}{1^{1+\delta}} + \frac{1}{2^{1+\delta}} + \cdots + \frac{1}{n^{1+\delta}} = O_\delta(1)$$

Proof By drawing the graph of the function $\frac{1}{x^{1+\delta}}$, we can find

$$\begin{aligned} 1 + \sum_{i=2}^{n} \frac{1}{x^{1+\delta}} &\le 1 + \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{x^{1+\delta}} dx \\ &= 1 + \int_{1}^{n} \frac{1}{x^{1+\delta}} dx \\ &= 1 + (\frac{n^{-\delta}}{-\delta} - \frac{1^{-\delta}}{-\delta}) \\ &\le 1 + \frac{1}{\delta} = O_\delta(1) \end{aligned}$$

The integral comes from $\int x^n dx = \frac{x^{n+1}}{n+1} + C$, let $n = -(1+\delta)$ we have $\int x^{-(1+\delta)} dx = \frac{x^{-\delta}}{-\delta} + C$.
$\square$

<u>**Def 1.4**</u> Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = \widetilde{O}(g(x))$ iff

$$(\exists\, C > 0)(\exists\, N)(\forall\, x \ge N)(|f(x)| \le \log^C x |g(x)|)$$

In other words, tidle hides **polylogarithmic** ($\log^{O(1)} n$) terms.

$O(1)$	constant
$O(\log n)$	logarithmic
$O(\log^c n)$	polylogarithmic
$O(n)$	linear
$O(n \log^c n)$ $(\widetilde{O}(n))$	quasilinear
$O(n^2)$	quadratic
$n^{O(1)}$	polynomial (e.g. n^{100})
$2^{O(n)}$	exporential

Table 1: Complexity classes

<u>Def 1.5</u> Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = \Omega(g(x))$ iff

$$(\exists\, C > 0)(\exists\, N)(\forall\, x \geq N)(|f(x)| \geq C|g(x)|)$$

<u>Def 1.6</u> Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = \Theta(g(x))$ iff

$$(\exists\, C_1, C_2 > 0)(\exists\, N)(\forall\, x \geq N)(C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|)$$

In other words, $f(x) = \Theta(g(x))$ iff $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

<u>Def 1.7</u> Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = o(g(x))$ iff

$$(\forall\, \epsilon > 0)(\exists\, N)(\forall\, x \geq N)(|f(x)| \leq \epsilon|g(x)|)$$

Ex 7 $2n = o(n^2)$

Proof $2n \leq \epsilon n^2$ when $n \geq \frac{2}{\epsilon}$. $\square$

Ex 8 $\frac{1}{\log n} = o(1)$

Proof For any $\epsilon > 0$, we need to find $N = N(\epsilon)$ such that $\frac{1}{\log n} \leq \epsilon$ for any $x \geq N$.

$$\frac{1}{\log n} \leq \epsilon \Leftrightarrow \log n \geq \frac{1}{\epsilon} \Leftrightarrow n \geq 2^{\frac{1}{\epsilon}}$$

Let $N = 2^{\frac{1}{\epsilon}}$ and we prove the exercise. $\square$

<u>Def 1.8</u> Let $f, g : \mathbf{R} \to \mathbf{R}$. Write $f(x) = \omega(g(x))$ iff

$$(\forall\, c > 0)(\exists\, N)(\forall\, x \geq N)(|f(x)| \geq c|g(x)|)$$

For example, we have $n = O(n^2)$, $n^2 = \omega(n)$.

2 Alphabet and Language

Def 2.1 An **alphabet** is a set of symbols.

Roman alphabet $\{a, b, \cdots, z\}$

Binary alphabet $\{0, 1\}$

Def 2.2 A **string** cover an alphabet is a finite sequence of symbols from the alphabet.

An **empty string** is a string with no symbol, denoted by ϵ.

Def 2.3 Let Σ be an alphabet. The set of all strings is denoted by Σ^*.

For example, $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \cdots\}$.

Denote the set of strings of length n by Σ^n. So we have $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$.

The **length** of a stirng is its length as a sequence.

Denote the length of a string ω by $|\omega|$.

For example, $|\epsilon| = 0$ and $|0110| = 4$.

Def 2.4 Two strings over the same alphabet can be combined to form a third by **concatenation**. The concatenation of X and Y is denoted by XY.

Def 2.5 A **language** is a set of strings over an alphabet, denoted by $L \subseteq \Sigma^*$.

For example, $\Sigma = \{0, 1\}$.

$\texttt{Even} = \{0, 10, 100, 110, \cdots\}$, $\texttt{Prime} = \{2, 3, 5, 7, 11, \cdots\} = \{10, 11, 101, 111, 1011, \cdots\}$

Encodings of problem

Examples

1. **Integer Multiplication**
   - `Input: X, Y`
   - `Output: XY`
2. **Primality Test**
   - `Input:` n
   - `Output: Yes/No (Yes if` n `is a prime)`
3. **Hamilton Path**
   - `Input: Undirected Graph` G
   - `Output: Yes/No (Yes if` G `has a Hamilton path)`

Any decision problem can be represented as a language $L \subseteq \{0, 1\}^*$.

Any computation problem can be represented as a function $f : \Sigma^* \to \Sigma^*$.

Often, encoding does not matter. We can switch between encodings by **preprocessing**.

For example, for an undirected graph G with vertices $V = \{1, 2, 3\}$ and edges $E = \{(1, 2), (1, 3), (2, 3)\}$, we can also represent it by an adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

3 Finite Automaton

Computation model with no memory

Example 1: Door State Automaton

door state\input	**neither**	**front**	**rear**	**both**
closed	closed	open	open	open
open	closed	open	open	open

Table 2: Transition Table

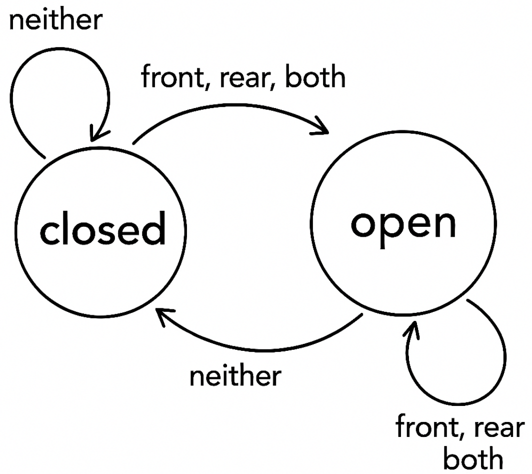

Figure 2: Transition Diagram

For this example, the **alphabet** is defined as $\Sigma = \{b, f, r, n\}$.

Example 2

state\input	**0**	**1**
q_0	q_0	q_1
q_1	q_0	q_1

Table 3: Transition Table

For this example, the **alphabet** is defined as $\Sigma = \{0, 1\}$.

The **initial state** is q_0, the **accept state** is q_0.

<u>**Def 3.1**</u> A **deterministic finite automaton (DFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a **finite set of states**.
2. Σ is the **alphabet**.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**.
4. $q_0 \in Q$ is the **start state**.
5. $F \subseteq Q$ is the **set of accept states**.

In Example 2, by the **transition table**, the **transition function** δ is defined as $\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$, $\delta(q_1, 0) = q_0$, $\delta(q_1, 1) = q_1$.

<u>**Def 3.2**</u> **(M accepts an input)** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $\omega = \omega_1 \cdots \omega_n$ be a string where $\omega_i \in \Sigma$. Then M **accepts** ω iff there is a sequence of states $r_0, r_1, \cdots, r_n \in Q$ such that

1. $r_0 = q_0$
2. $\delta(r_i, \omega_{i+1}) = r_{i+1} \quad (0 \leq i < n)$
3. $r_n \in F$

<u>**Def 3.3**</u> Let $L \subseteq \Sigma^*$. Let M be a DFA. Say M **accepts** L iff L is the set of all strings that M accepts. We also say M **recognizes** L.

In Example 2, the DFA accepts $L = \{\omega : \omega \texttt{ is ended with } 0\}$.

<u>**Def 3.4**</u> **(Regular language)** A language is called a **regular language** iff some DFA accepts it.

Ex 9 Design a DFA: $M = (Q, \Sigma, \delta, q_0, F)$ such that M accepts

$$L = \{\omega \in \{0,1\}^* : \omega = \omega_1 \cdots \omega_n \texttt{ where } \omega_n + 2\omega_{n-1} + \cdots + 2^{n-1}\omega_1 \equiv 0 \pmod 3\}$$

Solution Let $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, **start state** q_0, **accept state** q_0.
Define the **transition function** δ:

$$\delta(q_i, j) = q_k,\ k \equiv 2i + j \pmod 3 \in \{0,1,2\}$$

For $\omega \in \{0,1\}^*$, $\omega = \omega_1 \cdots \omega_n$,

$$\begin{aligned} M \texttt{ accepts } \omega &\iff 2(\cdots 2(2 \times 0 + \omega_1) + \omega_2 \cdots) + \omega_n \equiv 0 \pmod 3 \\ &\iff \omega_n + 2\omega_{n-1} + \cdots + 2^{n-1}\omega_1 \equiv 0 \pmod 3 \end{aligned}$$

which is the description of L. $\square$

Seperating Words Problem

Given distinct $x, y \in \{0,1\}^n$. Let $f_n(x,y)$ denote the smallest m s.t. $\exists$ FA with m states that accepts x but not y. Let

$$f(n) \stackrel{\text{def}}{=} \max_{x \neq y \in \{0,1\}^n} f_n(x,y)$$

Lower bound $f(n) = \Omega(\log n)$

Upper bound $f(n) = \widetilde{O}(n^{\frac{1}{3}})$ **Chase (2022)**

We will show regular languages are closed under operations: **union, intersection, complement, concatenation, star.**

Def 3.5 Let $A, B \subseteq \Sigma^*$. Define

1. **(union)** $A \bigcup B = \{x | x \in A \texttt{ or } x \in B\}$
2. **(intersection)** $A \bigcap B = \{x | x \in A \texttt{ and } x \in B\}$
3. **(complement)** $\overline{A} = \{x | x \notin A\}$
4. **(concatenation)** $A \circ B$ (or AB) $= \{xy | x \in A \texttt{ and } y \in B\}$
5. **(star)** $A^* = \{x_1 x_2 \cdots x_k : k \geq 0 \texttt{ and } x_i \in A\}$

For example, $A = \{\epsilon, 0, 00, 000, \cdots\}$, $B = \{\epsilon, 1, 11, 111, \cdots\}$.
$A \circ B = \{00 \cdots 011 \cdots 1 = 0^i 1^j,\ i, j \geq 0\} = \{\epsilon, 0, 1, 01, 00, 11, \cdots\}$
$A^* = A$, $B^* = B$.

Thm 3.6 If L_1 and L_2 are regular languages, $L_1 \bigcup L_2$ is also a regular language.

Proof Idea: Simulate M_1 and M_2 simultenously.

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1 \subseteq Q_1)$ accept L_1, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2 \subseteq Q_2)$ accept L_2.

Construct $M = (Q, \Sigma, \delta, q, F)$ that accepts $L_1 \bigcup L_2$.

1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) : r_1 \in Q_1 \texttt{ and } r_2 \in Q_2\}$
2. Σ is the same
3. $\delta : Q \times \Sigma \to Q$ is defined as
$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$
where $(r_1, r_2) \in Q$, $a \in \Sigma$.
4. $q_0 = (q_1, q_2)$
5. $F = \{(r_1, r_2) | r_1 \in F_1 \texttt{ and } r_2 \in F_2\}$ $\square$

Thm 3.7 If L_1 and L_2 are regular languages, $L_1 \bigcap L_2$ is also a regular language.

Thm 3.8 If L is a regular language, $\overline{L}$ is also a regular language.

Thm 3.9 If L_1 and L_2 are regular languages, $L_1 \circ L_2$ is also a regular language.

Deterministic Finite Automaton (DFA)

Next state is determined.

Nondeterministic Finite Automaton (NFA)

Several choices may exist - `"parallel universes"`.

ε-transitions allowed.

Accept iff there is at least one path ending in an accept state.

Example Design an NFA that accepts $L = \{0^k : 2|k \texttt{ or } 3|k\}$

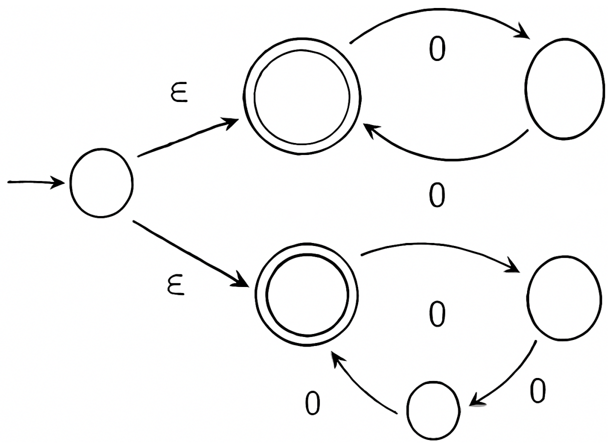

Figure 3: Adequate NFA

<u>**Def 3.10**</u> A **nondeterministic finite automaton (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a **finite set of states**.
2. Σ is the **alphabet**.
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the **transition function**. $\Sigma_\varepsilon \stackrel{\text{def}}{=} \Sigma \cup \{\varepsilon\}$, $\mathcal{P}(Q) \stackrel{\text{def}}{=} \{S : S \subseteq Q\}$ is the **power set**.
4. $q_0 \in Q$ is the **start state**.
5. $F \subseteq Q$ is the **set of accept states**.

<u>**Def 3.11**</u> Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Let $\omega \in \Sigma^*$. We say N **accepts** ω iff we can write $\omega = y_1 y_2 \cdots y_m$, where $y_i \in \Sigma^*$, and $\exists\, r_0, r_1, \cdots, r_m \in Q$ such that

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for $i = 0, 1, \cdots, m-1$
3. $r_m \in F$

Conj	Conjective
Def	Definition
Lem	Lemma
Thm	Theorem
Prop	Proposition
Cor	Corollary

Table 4: Explanation of Abbreviations

Lem 3.12 If L_1 and L_2 are regular languages then $L_1 L_2$ is a regular language.

Proof Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA that accepts L_1, $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be an NFA that accepts L_2. We construct an NFA $N = (Q, \Sigma, \delta, q_1, F_2)$ to accept $L_1 L_2$.

1. $Q = Q_1 \cup Q_2$.
2. q_1 is the initial state.
3. F_2 are the accept states.
4. $\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \backslash F_1 \\ \delta_1(q,a) & q \in F_1 \texttt{ and } a \neq \varepsilon \\ \delta_2(q,a) & q \in Q_2 \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \texttt{ and } a = \varepsilon \end{cases}$ □

Lem 3.13 If L is a regular language. L^* is a regular language.

Proof Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA that accepts L. We construct an NFA $N' = (Q', \Sigma, \delta', q_0', F')$ to accept L^*.

1. $Q' = Q \cup \{q_0'\}$.
2. $F' = F \cup \{q_0'\}$.
3. $\delta'(q,a) = \begin{cases} q_0 & q = q_0' \texttt{ and } a = \varepsilon \\ \delta(q,a) & q \in Q \backslash F \\ \delta(q,a) & q \in F \texttt{ and } a \neq \varepsilon \\ \delta(q,a) \cup \{q_0'\} & q \in F \texttt{ and } a = \varepsilon \end{cases}$ □

Remark Actually, the proof relies on the fact that any regular language can be accepted by an NFA. See the remark on **Cor 3.15**.

Thm 3.14 Let L be a language accepted by an NFA. There is a DFA that accepts L.

Proof Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA that accepts L. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ to accept L^*. To deal with ε arrows, let

$$E(R) \stackrel{\text{def}}{=} \{q : q \texttt{ can be reached from } R \texttt{ by travelling along zero or more } \varepsilon \texttt{ arrows}\}$$

where $R \subseteq Q$.

1. $Q' = \mathcal{P}(Q)$.
2. $q_0' = E(\{q_0\})$.
3. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) \stackrel{\text{def}}{=} \bigcup_{r \in R} E(\delta(r, a))$
4. $F' = \{R \in Q' : R \cap F \neq \varnothing\}$ $\square$

Cor 3.15 A language is regular if some NFA accepts it.

Remark More generally, a language is regular iff some NFA accepts it.

Given a computation model, we are interested in its limitations.

That is, **what is computable by DFA?**

Thm 3.16 Almost all languages are **nonregular**.

Proof $\#$ `all languages` $= 2^{\aleph_0}$ `is uncountable` $(2^{\aleph_0} > \aleph_0)$

$\#$ `regular languages` $\leq \#$ `DFAs` $\leq \#$ `01 Strings` $=$ `countable` $\square$

Lem 3.17 (Pumping Lemma) Let Σ be an alphabet. Let $L \subseteq \Sigma^*$ be a regular language. $\exists\, p \in \mathrm{N}$ s.t. for any string $S \in L$ of length $\geq p$, $\exists\, x, y, z \in \Sigma^*$ s.t. $S = xyz$ and

1. $xy^iz \in L \quad \forall\, i \geq 0$
2. $|y| > 0$
3. $|xy| \leq p$

Remark y^i means repeating the string y exactly i times.

Proof Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA that accepts L. Let $p \stackrel{\text{def}}{=} |Q|$. Let $S = S_1 \cdots S_n \in L$, where $n \geq p$. Let $r_0, r_1, \cdots, r_n \in Q$ be the sequence of states where

1. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, s_i) \quad \forall\, i = 0, 1, \cdots, n-1$

Since $n + 1 > |Q|$, by PHP, two states must be the same. Call the first r_j, the second r_l. Let

$$x \stackrel{\text{def}}{=} S_1 S_2 \cdots S_j,\ y \stackrel{\text{def}}{=} S_{j+1} \cdots S_l,\ z \stackrel{\text{def}}{=} S_{l+1} \cdots S_n \qquad \square$$

Ex 10 $\{0^n 1^n : n \geq 0\}$ is nonregular.

Proof Let $p \in \mathbf{N}$ be sufficiently large. Let $S = 0^p 1^p \in L$. By the Pumping Lemma, $\exists x, y, z$ s.t.

1. $s = xyz$
2. $|xy| \leq p \implies y = 0^j\ (j > 0)$
3. $xy^iz \in L \quad \forall\, i \geq 0 \implies 0^{p+ji} 1^p \in L$ `Contradiction!` $\square$

4 Turing Machine

In 1936, Turing gave a vigorous definition of computation for the first time.

<u>**Def 4.1**</u> **(Turing Machine)** A TM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\texttt{accept}}, q_{\texttt{reject}})$, where

1. Q is the **set of states**.
2. Σ is the **input alphabet**, where $\sqcup \notin \Sigma$.
3. Γ is the **tape alphabet**, where $\triangleright, \sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is the **transition function**.
5. $q_0 \in Q$ is the **start state**.
6. $q_{\texttt{accept}} \in Q$ is the **accept state**.
7. $q_{\texttt{reject}} \in Q$ is the **reject state**.

Remark A TM **accepts** an input when it enters an accept state, and the machine halts.

Ex 11 Describe a TM of adding one to a binary number.

- **Input:** Binary number n (LSB first)
- **Output:** Binary number $n + 1$ (LSB first)

For example,

Input	$1\,1\,\sqcup$	$1\,0\,1\,\sqcup$
Output	$0\,0\,1$	$0\,1\,1$

Solution Let $Q = \{q_0, q_{\texttt{end}}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup, \triangleright\}$. Let the start state and the accept state be q_0 and $q_{\texttt{end}}$. Define the transition function δ

$$\delta(q_0, \triangleright) = (q_0, \triangleright, R),\ \delta(q_0, 1) = (q_0, 0, R),\ \delta(q_0, 0) = (q_{\texttt{end}}, 1, S),\ \delta(q_0, \sqcup) = (q_{\texttt{end}}, 1, S)$$

The source code of TM is as follows. □

```
name: Adding one to a binary number
init: q_0
accept: q_end

q_0,1
q_0,0,>

q_0,0
q_end,1,-

q_0,_
q_end,1,-
```

Ex 12 Describe a TM that decides $L = \{0^{2^n} : n \geq 0\}$

Solution We have the following idea.

1. If the tape contains only a single 0, accept.
2. Scan from the left to right, crossing off every other 0.
3. If $\#0's$ is odd, reject.
4. Move the head back to the left and repeat from Step 1.

Let $\Sigma = \{0\}$ and $\Gamma = \{0, x, \llcorner\!\lrcorner, \triangleright\}$. The state diagram is as follows.

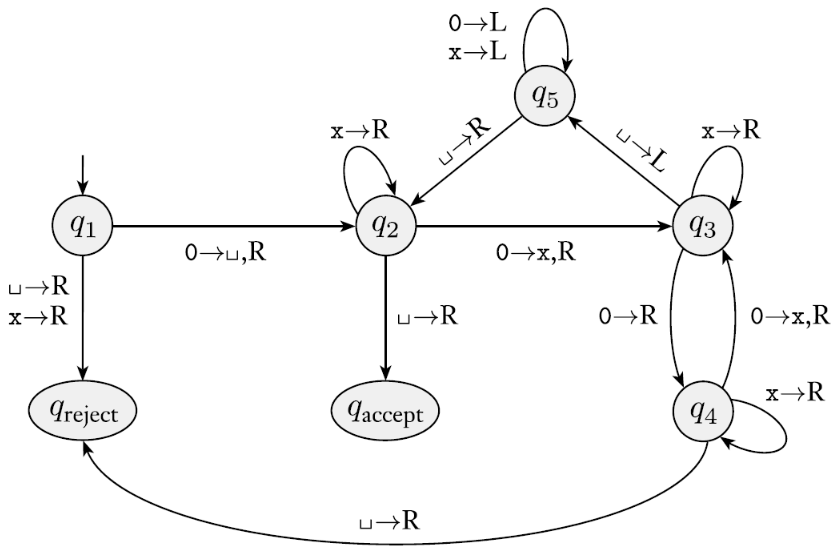

Figure 4: State Diagram

Remark Assume the input length is $m\,(= 2^n)$, the running time is $O(m \log m)$.

Ex 13 Describe a TM that decides $L = \{w\#w : w \in \{0,1\}^*\}$.

Solution Let $\Sigma = \{0, 1, \#\}$ and $\Gamma = \{0, 1, \#, x, \llcorner\!\lrcorner, \triangleright\}$. The state diagram is as follows.

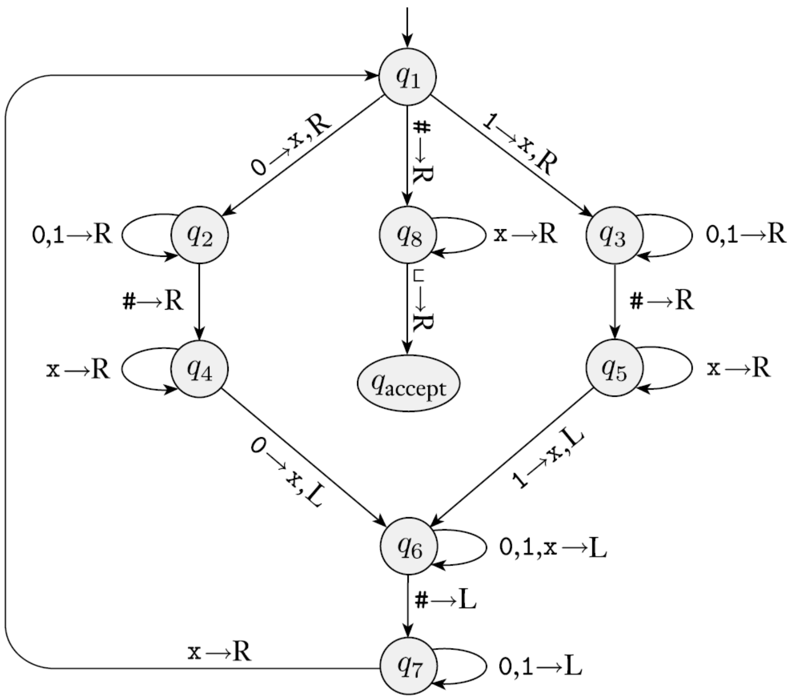

Figure 5: State Diagram

<u>**Def 4.2**</u> Let $L \subseteq \{0,1\}^*$, Say Turing machine M **decides** L in time $T(n)$ iff for every input $x \in \{0,1\}^*$,

1. M halts in $T(|x|)$ steps.
2. If $x \in L$, then M accepts x.
3. If $x \notin L$, then M rejects x.

<u>**Def 4.3**</u> Call a language **(Turing) decidable** iff there is a TM that decides it.

<u>**Def 4.4**</u> The set of strings that a Turing machine M accepts is the language **recognized** by M, denoted by $L(M)$.

<u>**Def 4.5**</u> Let $L \subseteq \{0,1\}^*$. Call L **(Turing) recognizable** iff there is a TM M s.t. the language recognized by M is L, i.e., $L(M) = L$.

<u>**Def 4.6**</u> Let $f : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$. Say a TM M computes f in time $T(n)$ iff for every $x \in \Sigma^*$ such that $f(x) \neq \perp$, M halts with $f(x)$ on its tape within $T(|x|)$ steps.

What is an algorithm?

An algorithm is a TM.

Variants of TM

The original model and its reasonable variants all have the **same power**.

Their running time differs by a **polynomial factor**.

Lem 4.7 (Change of alphabet) Let $L \subseteq \{0,1\}^*$. If L is decidable by a TM on alphabet Γ in time $T(n)$, then L is decidable in time $O(\log|\Gamma|\, T(n))$ on alphabet $\Sigma = \{0,1\}$.

Proof Encode a symbol in Γ using $k \overset{\text{def}}{=} \lceil \log_2 |\Gamma| \rceil$ bits. To simulate one step of M, the new machine M' will

1. Spend k steps to read the symbol $a \in \Gamma$ and determine the new symbol $b \in \Gamma$ to be written.
2. Overwrite a by b.
3. Move the head to the next symbol.
4. Transit to the next state.

In total, it takes $k + k + k + 1 = O(k)$ steps. $\square$

<u>**Def 4.8**</u> **(Multitape TM)** A k-tape TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\texttt{accept}}, q_{\texttt{reject}})$, where

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$$

Typically, the first tape is the input tape.

Lem 4.9 Let $L \subseteq \{0,1\}^*$. If L is decidable by a k-tape TM in time $T(n)$, then it is decidable in time $O(k\,T(n)^2)$ by a single-tape TM.

Proof Use positions $0, k, 2k, \cdots$ to simulate the first tape.

Use positions $1, k+1, 2k+1, \cdots$ to simulate the second tape.

Generally, use position $i-1, k+i-1, \cdots$ to simulate the i^{th} tape, where $i = 1, 2, \cdots, k$.

For every $a \in \Gamma$, introduce $a, \hat{a} \in \Gamma'$ with $\hat{a}$ marking the location of the head. To simulate one step of M, the machine M' will

1. Sweep form left to right to read k symbols (marked by $\hat{}$).
2. Use $M's$ transition function to determine the new state and actions.
3. Sweep back from right to left to update the symbols and adjust the head positions (i.e., move the hats).

It takes $O(k\,T(n)) + 1 + O(k\,T(n))$ to simulate one step. In total, $O(k\,T(n)^2)$. $\square$

Bidirectional TM is a TM whose tape is infinite in both directions.

Lem 4.10 Let $L \subseteq \{0,1\}^*$. If L is decidable by a bidirectional TM in time $T(n)$, then L is decidable by a single-tape TM in time $O(T(n))$.

Def 4.11 (Random access memory TM) A RAM TM is a TM with random access memory.

1. M has an infinite memory tape A indexed by $\mathbf{N}$.
2. One tape is the address tape.
3. Σ contains two special symbols R (read) and W (write).
4. Q has some special states $Q_{\text{access}} \subseteq Q$.

Whenever M gets into a state $q \in Q_{\text{access}}$.

- If the address tape contains iR, the value $A[i]$ is written to the cell next to R.
- If the address tape contains $iW\sigma$, then $A[i]$ is set to the value σ.

Lem 4.12 Let $L \subseteq \{0,1\}^*$. If L is decidable by a RAM TM in time $T(n)$, then it is decidable by a multitape TM in time $O(T(n)^3)$. Moreover, if the address used has bounded length $O(1)$, then L can be decided by a multitape TM in time $O(T(n)^2)$.

Proof Use an extra tape as the memory, which contains pains like $(i, A[i])$ for all addresses i that have been referenced.

If the RAM TM M is in an access state, the machine M' will

1. Scans through tape A to find the entry whose address in i, where i is the integer stored on the address tape.
2. If no such entry exists, create a new entry $(i, A[i])$ on tape A.
3. Read from or write to $A[i]$ depending on the symbol (W or R) on the address tape.

One-step simulation takes

$$O(|A|) = O(\#\texttt{address} \times \texttt{length of each address}) = O(T(n)) \times T(n) = O(T(n)^2)$$

In total, it takes $T(n) \times O(T(n)^2) = O(T(n)^3)$. $\square$

$$\begin{array}{ccccccccc} \texttt{C++} & \longrightarrow & \texttt{Assembly Language} & \longrightarrow & \texttt{RAM TM} & \xrightarrow{\textbf{Lem 4.12}} & \texttt{Multitape TM} & \xrightarrow{\textbf{Lem 4.9}} & \texttt{Single-tape TM} \\ T(n) & & O(T(n)) & & O(T(n)) & & O(T(n)^2) & & O(T(n)^4) \end{array}$$

Every assembly language instruction can be implemented by a RAM TM in $O(1)$ steps.

Church-Turing Thesis

Every function that can be physically computed can be computed by a Turing machine.

Strong CT Thesis

Every function that can be physically computed can be computed by a Turing machine with polynomial time overhead.

Possible Exception: Quantum Computers

Def 4.13 Let $T : \mathbf{N} \to \mathbf{N}$. $L \subseteq \{0,1\}^*$ is in $\texttt{DTIME}[T(n)]$ if there is a multitape TM that decides L in time $O(T(n))$.

Def 4.14

$$\mathbf{P} \stackrel{\text{def}}{=} \bigcup_{k\geq 1} \texttt{DTIME}[n^k] = \texttt{DTIME}[n^{O(1)}] = \texttt{DTIME}[\mathbf{poly}(n)]$$

Williams (2025) For any $T(n) \geq n$, $\texttt{DTIME}[T(n)] \subseteq \texttt{SPACE}[\sqrt{T(n)\log T(n)}]$

5 Universal TM

A **universal TM (UTM)** is a theoretical machine that can simulate any other TM. It takes as input the description of a machine and its input and mimics the behavior.

Encoding of TM

Website: turingmachinesimulator.com

$$\left.\begin{array}{ll} \texttt{initial state:} & q_0 \\ \texttt{accept state:} & q_{\text{accept}} \\ q_0, 0 & \\ q_0, 1, > & \end{array}\right\} \xrightarrow{\textbf{ASCII Code}} \alpha \in \{0,1\}^*$$

Assume our encoding satisfies the following properties:

1. Every string $\alpha \in \{0,1\}^*$ represents some TM, denoted by M_α.
2. Every TM is represented by infinitely many strings in $\{0,1\}^*$.

Thm 5.1 There is a 3-tape TM $\mathcal{U}$ s.t. $\forall\, x, \alpha \in \{0,1\}^*$, $\mathcal{U}(x,\alpha) = M_\alpha(x)$. Moreover, if M_α halts within T steps, then $\mathcal{U}(x,\alpha)$ halts in $O_k(T \log T)$ steps, where k is the number of tapes in M_α.

Proof Combine k tapes into one tape by working over Γ^k. Assume tapes are two-way infinite.

Instead of moving heads, $\mathcal{U}$ moves the tape. However, the simulation takes $O_k(T^2)$.

The idea is to allocate additional buffer space so that each move has an amortized cost of $O_k(\log T)$.

We split each of $\mathcal{U}$'s parallel tapes into zones, denoted $R_0, L_0, R_1, L_1, \cdots, R_{\lceil \log T \rceil}, L_{\lceil \log T \rceil}$. The center cell is not in any zone. $|R_i| = |L_i| = 2^{i+1}$ maintains the following invariants.

1. Each of the zones is either empty, full, or half-full. That is $\#\boxtimes$ cells are 2^{i+1}, 0 or 2^i.

2. The number of $\boxtimes$ in $L_i \cup R_i$ is 2^{i+1}. That is, if L_i is full, then R_i is empty; if L_i is empty, then R_i is full; if L_i is half-full, then R_i is half-full.

Initially, all zones are half-full. It takes $\geq 2^{i_0} - 1$ shifts to make $R_0, R_1, \cdots, R_{i_0-1}$ empty. So, the first $2^{i_0} - 1$ shifts have index $< i_0$.

How to perform a shift (left shift for example)

1. Find the smallest i_0 s.t. R_{i_0} is not empty.

2. Put the leftmost non-$\boxtimes$ symbol of R_{i_0} to position 0, and shift the remaining leftmost $2^{i_0}-1$ non-$\boxtimes$ symbols from R_{i_0} into $R_0, R_1, \cdots, R_{i_0-1} = 2^1+2^2+\cdots+2^{i_0} = 2^{i_0+1}-2 = 2(2^{i_0}-1)$.

$\mathcal{U}$ performs the symmetric operation to the left. That is, we move $\frac{1}{2}(|L_0| + \cdots + |L_{i_0-1}|) + 1$ non-$\boxtimes$ cells to L_{i_0}. Now, $R_0, L_0, \cdots, R_{i_0-1}, L_{i_0-1}$ are half-full. R_{i_0} is empty or half-full; L_{i_0} is half-full or full.

We call i_0 the **index** of this shift operation.

Once we perform a shift with index i_0, the next $2^{i_0} - 1$ shifts will have index $< i_0$. Thus, at most $\frac{1}{2^{i_0}}$ fraction of shifts have index i_0.

Denote $\texttt{index}(i)$ as the index of the i^{th} operation.

$$\begin{aligned}
k\sum_{i=1}^{T} O(2^{\texttt{index}(i)}) &\leq k \sum_{j=0}^{O(\log T)} O(2^j) \cdot \#\{i : \texttt{index}(i) = j\} \\
&\leq k \sum_{j=0}^{O(\log T)} O(2^j)\frac{T}{2^j} \\
&= k \sum_{j=0}^{O(\log T)} O(T) = O(kT\log T) \qquad \square
\end{aligned}$$

Thm 5.2 Almost all languages are undecidable.

Proof $\#$ `all languages` $= 2^{\aleph_0}$ `is uncountable` $(2^{\aleph_0} > \aleph_0)$

$\#$ `decidable languages` $\leq \#$ `TMs` $\leq \#$ `01 Strings` $=$ `countable` $\square$

Turing Halting Problem

$L_{\texttt{halt}} \stackrel{\text{def}}{=} \{(\alpha, x) : M_\alpha \texttt{ halts on input } x\} \subseteq \{0,1\}^*$

$L_{\texttt{flip}} \stackrel{\text{def}}{=} \{\alpha \in \{0,1\}^* : M_\alpha \texttt{ does not accept } \alpha\texttt{, i.e., } M_\alpha \texttt{ rejects } \alpha \texttt{ or loops forever}\}$

$L_{\texttt{accept}} \stackrel{\text{def}}{=} \{(\alpha, x) : M_\alpha \texttt{ accepts } x\}$

$L_{\texttt{empty}} \stackrel{\text{def}}{=} \{\langle M \rangle : M \texttt{ is a TM that does not accept any input, i.e., } L(M) = \varnothing$

$L_{\texttt{regular}} \stackrel{\text{def}}{=} \{\alpha : L(M_\alpha) \texttt{ is a regular language}\}$

Lem 5.3 $L_{\texttt{flip}}$ is undecidable.

Proof Assume for contradiction that $L_{\texttt{flip}}$ is decided by some TM M_α. Thus, $L(M_\alpha) = L_{\texttt{flip}}$.

1. $\alpha \in L_{\texttt{flip}}$. By the definition of $L_{\texttt{flip}}$, M_α does not accept α. So, $\alpha \notin L(M_\alpha) = L_{\texttt{flip}}$. Contradiction.

2. $\alpha \notin L_{\texttt{flip}}$. By definition, M_α accepts α. So, $\alpha \in L(M_\alpha) = L_{\texttt{flip}}$. Contradiction. □

Lem 5.4 $L_{\texttt{halt}}$ is undecidable.

Proof Assume for contradiction that $L_{\texttt{halt}}$ is decidable, i.e., $\exists$ TM $M_{\texttt{halt}}$ that decides $L_{\texttt{halt}}$. Create a new TM $M_{\texttt{flip}}$ as follows.

On input α, run $M_{\texttt{halt}}$ on input (α, α). If $M_{\texttt{halt}}$ rejects (α, α), then $M_{\texttt{flip}}$ accepts α. If $M_{\texttt{halt}}$ accepts (α, α), simulate M_α on input α, and flip the output, that is, $M_{\texttt{flip}}(\alpha) = \neg\mathcal{U}(\alpha, \alpha)$.

Clearly, $M_{\texttt{flip}}$ decides $L_{\texttt{flip}}$. Contradiction. □

Lem 5.5 $L_{\texttt{accept}}$ is undecidable.

Proof Assume for contradiction $L_{\texttt{accept}}$ is decidable, i.e., $\exists$ TM $M_{\texttt{accept}}$ that decides $L_{\texttt{accept}}$. We construct a TM $M_{\texttt{halt}}$ as follows.

On input (α, x), create a new TM M_β, which always accepts whenever M_α halts. If M_α loops forever, M_β also loops forever. Run $M_{\texttt{accept}}$ on input (β, x), forward the output.

Clearly, $M_{\texttt{halt}}$ decides $L_{\texttt{halt}}$. Contradiction. □

Lem 5.6 $L_{\texttt{empty}}$ is undecidable.

Proof Assume for contradiction that $L_{\texttt{empty}}$ is undecidable, i.e., $\exists$ TM $M_{\texttt{empty}}$ that decides $L_{\texttt{empty}}$. We construct a new TM $M_{\texttt{halt}}$.

On input (α, x), construct a new TM M_β as follows. M_β's input is $y \in \{0,1\}^*$.

1. Simulate M_α on input x.

2. If Step (1) halts, M_β always accepts y.

Clearly, $L(M_\beta) = \varnothing$ if M_α does not halt on x. Otherwise, $L(M_\beta) = \{0,1\}^*$.

Run $M_{\texttt{empty}}$ on β and flip its output.

Clearly, $M_{\texttt{halt}}$ decides $L_{\texttt{halt}}$. Contradiction. □

Lem 5.7 $L_{\texttt{regular}}$ is undecidable.

Proof Assume for contradiction that $L_{\texttt{regular}}$ is undecidable, i.e., $\exists$ TM $M_{\texttt{regular}}$ that decides $L_{\texttt{regular}}$. We will show $L_{\texttt{accept}}$ is decidable. Construct a TM $M_{\texttt{accept}}$, which on input (α, x), as follows:

1. Construct the following TM M_β, where the input of M_β is y.
   - If y has the form $0^n 1^n$, accept.
   - If y is not of the form, simulate M_α. On input x, accept y if M_α accepts x.

2. Run $M_{\texttt{regular}}$ on β. Forward its output.

Observe

$$L(M_\beta) = \begin{cases} \{0^n 1^n : n \geq 0\}, \texttt{ If } M_\alpha \texttt{ does not accept } x \\ \{0,1\}^*, \texttt{ Otherwise} \end{cases}$$

If M_α accepts x, $M_{\texttt{accept}}$ accepts (α, x). Otherwise, $M_{\texttt{accept}}$ rejects (α, x).

Clearly, $M_{\texttt{accept}}$ decides $L_{\texttt{accept}}$. Contradiction. $\square$

<u>**Def 5.8**</u> **(Nontrivial property of languages)** Property $\mathcal{P} \subseteq \{0,1\}^*$ is about the language recognized by TMs iff: whenever $L(M) = L(N)$, $\langle M \rangle \in \mathcal{P} \iff \langle N \rangle \in \mathcal{P}$. $\mathcal{P}$ is **nontrivial** iff $\mathcal{P} \neq \varnothing$ and $\mathcal{P} \neq \{0,1\}^*$.

Thm 5.9 (Rice's theorem) Any nontrivial property about the language recognized by TMs is undecidable.

Proof WLOG $\mathcal{P}$ does not contain the language $\varnothing$. Assume for contradiction that $\mathcal{P}$ is decided by a TM M. Since $\mathcal{P} \neq \varnothing$, pick an arbitrary $\beta \in \mathcal{P}$. Construct a TM $M_{\texttt{accept}}$.

1. On input (α, x), construct a TM M_γ as follows. Simulate M_α on input x.
   - If M_α does not accept x, M_γ loops forever.
   - If M_α accepts x, simulate M_β.
2. Run M on input γ. Forward its output.

We verify $M_{\texttt{accept}}$ decides $L_{\texttt{accept}}$. If M_α accepts x, then $L(M_\gamma) = L(M_\beta)$. Since $\beta \in \mathcal{P}$, $M_{\texttt{accept}}$ will accept (α, x). If M_α does not accept x, then $L(M_\gamma) = \varnothing$. So $\gamma \notin \mathcal{P}$, M will reject. $\square$

Post Correspondence Problem

Domino

$$\left[\frac{b}{ca}\right]$$

A collection of dominos

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

A match

$$\left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{a}{ab}\right], \left[\frac{abc}{c}\right]$$

$\mathrm{P} = \left\{ \left[\frac{t_1}{b_1}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\}$

$\mathrm{PCP} = \{\langle \mathrm{P} \rangle : \texttt{P is an instance with a match}\}$

Thm 5.10 (Emil Post 1946) PCP is undecidable.

6 Reduction

Roughly speaking, problem A is reducible to B, denoted by $A \leq B$, if an algorithm for solving B (efficiently) can be used as a subroutine to solve A (efficiently).

Examples

1. **Many-one reduction:** $L_1 \leq_m L_2$
2. **Turing reduction:** $L_1 \leq_T L_2$
3. **Polynomial-time many-one reduction (Karp reduction):** $L_1 \leq_p L_2$
4. **Polynomial-time Turing reduction (Cook reduction):** $L_1 \leq_T^P L_2$
5. **Log-space reduction**

Conjecture

There is a language which is NP-complete under Cook reduction, but is not NP-complete under Karp reduction. (mathoverflow.com)

Def 6.1 (Many-one reduction) Let $L_1, L_2 \subseteq \{0,1\}^*$. Say L_1 is **many-one reducible** to L_2, denoted by $L_1 \leq_m L_2$, if $\exists$ a **computable** function $\varphi : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$, $x \in L_1$ iff $\varphi(x) \in L_2$.

Prop 6.2

1. $(\forall L \subseteq \{0,1\}^*)\,(L \leq_m L)$
2. $L_1 \leq_m L_2$ iff $\overline{L_1} \leq_m \overline{L_2}$
3. **Transitivity:** If $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$

Def 6.3 (Karp reduction) Say L_1 is **Karp reducible** to L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a **polynomial-time computable** function $\varphi : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$, $x \in L_1$ iff $\varphi(x) \in L_2$.

Def 6.4 (Oracle TM) A Turing machine with an oracle for language L is a TM that has two additional kinds of states: query states $Q_{\texttt{query}} \subseteq Q$ and response states $Q_{\texttt{response}} \subseteq Q$. There is an extra tape called the oracle tape. When it enters a query state, the following are performed, all in one step.

1. String z that is written on the oracle tape is erased.
2. If $z \in L$, then 1 is written on the leftmost cell; Otherwise, 0 is written.
3. The oracle tape head is moved to the leftmost cell.
4. The machine enters a response state.

Def 6.5 (Turing reduction) L_1 is **Turing reducible** to L_2, denoted by $L_1 \leq_T L_2$, iff there is an **oracle** TM with an oracle for L_2 that decides L_1.

Def 6.6 (Cook reduction) L_1 is **Cook reducible** to L_2, denoted by $L_1 \leq_T^P L_2$, iff there is a **polynomial-time oracle** TM with an oracle for L_2 that decides L_1.

Def 6.7 Let $T : \mathbf{N} \to \mathbf{N}$. $L \subseteq \{0,1\}^*$ is in $\texttt{DTIME}\,[T(n)]$ if there is a multitape TM that decides L in time $O(T(n))$.

Def 6.8

$$\mathbf{P} \stackrel{\text{def}}{=} \bigcup_{k\geq 1} \texttt{DTIME}\,[n^k] = \texttt{DTIME}\,[n^{O(1)}] = \texttt{DTIME}\,[\mathbf{poly}(n)]$$

Remark Repeated mention of **Def 4.13** and **Def 4.14**. We also have a definition

$$\mathbf{EXP} \stackrel{\text{def}}{=} \bigcup_{c\geq 1} \texttt{DTIME}\,[2^{n^c}] = \texttt{DTIME}\,[2^{n^{O(1)}}]$$

The class NP

$$\begin{cases}\textbf{N: Non-deterministic}\\ \textbf{P: Polynomial-time}\end{cases}$$

NP is the set of languages that can be **verified** in polynomial time.

P is the set of languages that can be **decided** in polynomial time.

Def 6.9 Language $L \subseteq \{0,1\}^*$ is in **NP** if $\exists$ polynomial $P : \mathbf{N} \to \mathbf{N}$ and a polynomial-time TM M (called a verifier for L) such that for every $x \in \{0,1\}^*$, if $x \in L$, $\exists\, w \in \{0,1\}^{P(|x|)}$, M accepts (x, w); if $x \notin L$, $\forall w \in \{0,1\}^{P(|x|)}$, M rejects (x, w). Such w is called a **certificate/witness** for x.

Examples

1. **Graph Isomorphism Problem (GI)**

$$\text{GI} = \{\langle G, H\rangle : \texttt{undirected graphs } G \texttt{ and } H \texttt{ are isomorphic}\} \in \mathbf{NP}$$

2. **Clique Problem (CLIQUE)**

$$\text{CLIQUE} = \{\langle G, k\rangle : G \texttt{ contains a } K_k \texttt{ subgraph}\} \in \mathbf{NP}$$

3. **Traveling Salesman Problem (TSP)**

$$\text{TSP} = \{\langle G, k\rangle : \exists \texttt{ a closed circuit that visits each vertex exactly once, and has total length} \leq k\} \in \mathbf{NP}$$

Thm 6.10 $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$

Proof $\mathbf{P} \subseteq \mathbf{NP}$: For $L \in \mathbf{P}$, L is decided by a TM M efficiently. We can let $P(n) = 0$, $w = \epsilon$ and use M as the verifier. So $L \in \mathbf{NP}$.

$\mathbf{NP} \subseteq \mathbf{EXP}$: Enumerate all $w \in \{0,1\}^{P(|x|)}$, $\#\texttt{possible certificates} = 2^{P(n)}$. For $L \in \mathbf{NP}$, L can be decided in time $2^{P(n)} \cdot \mathbf{poly}(n) = 2^{n^{O(1)}}$. So $L \in \mathbf{EXP}$. $\square$

Nondeterministic Turing Machine

An NTM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\texttt{accept}}, q_{\texttt{reject}})$, where

1. Q is the **set of states**.
2. Σ is the **input alphabet**, where $\sqcup \notin \Sigma$.

3. Γ is the **tape alphabet**, where $\triangleright, \sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.
4. $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$ is the **transition function**.
5. $q_0 \in Q$ is the **start state**.
6. $q_{\texttt{accept}} \in Q$ is the **accept state**.
7. $q_{\texttt{reject}} \in Q$ is the **reject state**.

M accepts input x if **at least one** of the possible computation paths leads to an accept state.

Say M runs in time $T(n)$ if for every $x \in \{0,1\}^n$ and every sequence of nondeterministic choice, M reaches an end state in $T(n)$ steps.

Remark The difference between NTM and TM (**Def 4.1**) is the **transition function**.

Prop 6.12 Every NTM has an equivalent deterministic TM.

Def 6.13 Let $T : \mathbf{N} \to \mathbf{N}$. $L \subseteq \{0,1\}^*$ is in $\texttt{NTIME}[T(n)]$ if $\exists$ an NTM that runs in time $O(T(n))$ and decides L.

Thm 6.14

$$\mathbf{NP} = \bigcup_{c \geq 1} \texttt{NTIME}[n^c] = \texttt{NTIME}[n^{O(1)}]$$

Proof $\mathbf{NP} \subseteq \texttt{NTIME}[n^{O(1)}]$: Let $L \in \mathbf{NP}$. By definition, $\exists$ polynomial P and a poly-time TM M such that

1. If $x \in L$, $\exists\, w \in \{0,1\}^{P(|x|)}$ s.t. M accepts (x, w).
2. If $x \notin L$, $\forall\, w \in \{0,1\}^{P(|x|)}$ s.t. M rejects (x, w).

Construct an NTM on input x.

1. Nondetermenistically guess $w \in \{0,1\}^{P(|x|)}$.
2. Simulate M on (x, w). Accept if M accepts.

The NTM runs in polynomial time. So $L \in \texttt{NTIME}[n^{O(1)}]$.

$\texttt{NTIME}[n^{O(1)}] \subseteq \mathbf{NP}$: Let $L \in \texttt{NTIME}[n^{O(1)}]$. So L is decided by a binary choice NTM in time dn^c. We want to prove $L \in \mathbf{NP}$. Let $P(n) = dn^c$. The certificate $w \in \{0,1\}^{P(n)}$ indicates which transition function to apply. The verifier checks if x can be accepted. So $L \in \mathbf{NP}$. $\square$

Def 6.15 Language $L \subseteq \{0,1\}^*$ is **NP-hard** iff for any $K \in \mathbf{NP}$, $K \leq_p L$.

Def 6.16 Language $L \subseteq \{0,1\}^*$ is **NP-complete** iff L is **NP** and **NP-hard**.

Satisfiability Problem (SAT)

Boolean variable: `True(1)`, `False(0)`

Boolean operation: `AND`$(\wedge)$, `OR`$(\vee)$, `NOT`$(\neg)$

Remark We use the overbar as a shorthand for the $\neg$ symbol, so $\overline{x} = \neg x$.

A **Boolean formula** is an expression involving Boolean variables and operations. For example,

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1 `(True)`. For example, the preceding formula ϕ is satisfiable because when $x = z = 0$, $y = 1$, we have $\phi = 1$.

Now we can define the **Satisfiability Problem (SAT)**:

$$\mathbf{SAT} = \{\langle\phi\rangle : \phi \texttt{ is a satisfiable Boolean formula}\}$$

Thm 6.17 (Cook-Levin Theorem) **SAT** is **NP-complete**.

A **literal** is a Boolean variable(x) or a negated Boolean variable($\overline{x}$).

A **clause** is several literals connected with $\vee$s, e.g., $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$.

A Boolean formula is in **Conjunctive Normal Form**, called a **CNF**, iff it comprises several clauses connected with $\wedge$s, e.g.

$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$$

A **CNF** is a **3CNF** iff all the clauses have 3 literals, e.g.

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

Now we can define **3SAT**:

$$\mathbf{3SAT} = \{\langle\phi\rangle : \phi \texttt{ is a satisfiable 3CNF}\}$$

Remark Actually, **SAT** and **3SAT** have equivalent statements:

$$\mathbf{SAT} = \{\langle\phi\rangle : \phi \texttt{ is a satisfiable CNF}\}$$

$$\mathbf{3SAT} = \{\langle\phi\rangle : \phi \texttt{ is a satisfiable CNF which clauses have} \leq 3 \texttt{ literals}\}$$

Thm 6.18 $\mathbf{SAT} \leq_p \mathbf{3SAT}$

Proof Let ϕ be a satisfiable CNF. For clause ≤ 3 materials, done. For clause > 3 materials, we replace it by an equivalent 3CNF.

Let $C = l_1 \vee l_2 \vee \cdots \vee l_k$ be a clause with $k > 3$ materials, where $l_i \in \{x_1, \overline{x_1}, x_2, \overline{x_2}, \cdots$. Replace C by C', where

$$C' = (l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge (\overline{z_2} \vee l_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{k-3}} \vee l_{k-1} \vee l_k)$$

So ϕ can be written as a satisfiable CNF which clauses have ≤ 3 literals, $\mathbf{SAT} \leq_p \mathbf{3SAT}$. $\square$

Integer Programming (IP)

An **integer programming** problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. A special case is **0-1 integer linear programming**, in which unknowns are binary, and only the restrictions must be satisfied. **0-1 integer linear programming** is one of Karp's 21 NP-complete problems.

Thm 6.19 3SAT $\leq_p$ IP

Proof Let ϕ be a satisfiable 3CNF of n variables $x_1, x_2, \cdots, x_n$. Each variable $x_i \in \{0,1\}$. Each clause, e.g., $x_i \vee \overline{x_j} \vee \overline{x_k}$ is equivalent to

$$x_i + (1 - x_j) + (1 - x_k) \geq 1$$

So ϕ can be written as a satisfiable 0-1 integer linear programming problem, **3SAT** $\leq_p$ **IP**. □

An **independent set** is a set of vertices where no two vertices are connected by an edge. Let

$$\textbf{INDSET} = \{\langle G, k\rangle : G \texttt{ has an independent set of size } k\}$$

Thm 6.20 3SAT $\leq_p$ INDSET

Proof Let ϕ be a satisfiable 3CNF of n variables $x_1, x_2, \cdots, x_n$ and m clauses $C_1, C_2, \cdots, C_m$. We construct a graph G as follows.

For each clause C_j in ϕ, create 3 vertices corresponding to 3 literals in C_j. Thus, G has exactly $3m$ vertices.

Add edges to form a triangle(a clique of 3 vertices) among three vertices corresponding to each clause C_j. This ensures that at most 1 literal can be selected from the same clause. For every pair of vertices that correspond to complementary literals(i.e., x and $\overline{x}$), add an edge between them. This ensures that a variable and its negation cannot both be selected.

The graph G constructed has an independent set of size m, so **3SAT** $\leq_p$ **INDSET**. □

References

Chase, Z. (2022). A new upper bound for separating words. *arXiv preprint*, arXiv:2007.12097.

Williams, R. (2025). Simulating time with square-root space. *arXiv preprint*, arXiv:2502.17779.