Course Notes

COMP130023.01 Theory of Computation

Spring 2024

- Instructor: Yuan Li
- Email: yuanli_li@fudan.edu.cn

TA	$week(n \in \mathbb{N})$
Yiwen Gao	$5n + 1$
Kexin Li	$5n + 2$
Yangjun Li	$5n + 3$
Shun Wan	$5n + 4$
Shuangjun Zhang	$5n + 5$

What do we study?

- What is computation, i.e., computation model
- Finite automaton, context-free grammar
- Turing machine (= algorithm)
- Computability
- Complexity class (P, NP, PSPACE, EXP, L, NL, ...)
- NP completeness, reduction

Contents

1 Lesson 1

1.1 Big-O Notation

***def* 1.1.** *Let $f, g : \mathbb{N} \to \mathbb{R}$*

- *Write $f = O(g)$ if $(\exists c > 0)(\exists N)(\forall n \geq N)(|f(n)| \leq cg(n))$.*
- *Write $f = \Omega(g)$ if $(\exists c > 0)(\exists N)(\forall n \geq N)(|f(n)| \geq cg(n))$.*
- *Write $f = \Theta(g)$ if $(\exists c_1, c_2 > 0)(\exists N)(\forall n \geq N)(c_1 g(n) \leq |f(n)| \leq c_2 g(n))$.*
- *Write $f = o(g)$ if $(\forall \epsilon > 0)(\exists N)(\forall n \geq N)(|f(n)| \leq \epsilon g(n))$.*

Big-O Notation is the most commonly used one among the four.

***e.g.* 1.1.** $f(n) = 6n^4 - 3n^3 + 5 \Rightarrow f(n) = O(n^4)$

Proof. $|6n^4 - 3n^3 + 5| \leq 6n^4 + 3n^4 + 5n^4 = 13n^4$ □

***e.g.* 1.2.** $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$

***exercises* 1.1.** *Write in big-O notation:*

1. $5 + 0.001n^3 + 0.25n$
2. $500n + 100n^{1.5} + 50n \log_{10} n$
3. $n^2 \log_2 n + n(\log_2 n)^2$
4. $3 \log_8 n + \log_2(\log_2 n)$

solution $O(n^3); O(n^1.5); O(n^2 \log n); O(\log n)$ □

***prop* 1.1.**

- $f_1 = O(g_1), f_2 = O(g_2) \Rightarrow f_1 f_2 = O(g_1 g_2)$
- $f(O(g)) = O(fg)$
- $f_1 = O(g_1), f_2 = O(g_2) \Rightarrow f_1 + f_2 = O(max(g_1, g_2))$
- $f_1 = O(g), f_2 = O(g) \Rightarrow f_1 + f_2 = O(g)$
- $f = O(g) \Rightarrow kf = O(g)$

often encountered:

- *constant:* $O(1)$
  - $\sum\limits_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$
  - $\sum\limits_{k=1}^{n} \frac{1}{k^2} = O(1)$
  - $\sum\limits_{k=1}^{n} \frac{1}{k \ln k} = \ln \ln n + O(1)$
- *double logarithmic:* $O(\log \log n)$
- *logarithmic:* $O(\log n)$
- *polylogarithmic:* $O((\log n)^c), c > 0$
- *linear:* $O(n)$
- *quasilinear:* $O(n \log^c n), O(n \log^{O(1)} n)$
- *quadratic:* $O(n^2)$

***def* 1.2.** $\omega(g), \theta(g)$

- $f = \omega(g)$ *if* $(\forall c > 0)(\exists N)(\forall n \geq N)(f(n) \geq cg(n))$
- $g = \theta(g)$ *(or equivalently* $f \sim g$*) if* $(\forall \epsilon > 0)(\exists N)(\forall n \geq N)(|f(n) - g(n)| < \epsilon g(n))$

1.2 Alphabets and Languages

***def* 1.3.** (*alphabet*) *An alphabet is a set of symbols*

- *Roman alphabet* : $a, b, c, d, \cdots, z$
- *binary alphabet* : $0, 1$

***def* 1.4.** *(string and its length)*

*A **string** (over an alphabet) is a finite sequence of symbols from the alphabet.*
***Empty string** is string of no symbols, denoted by* ε.

The set of all string is denoted by Σ^*. *Denote by* Σ^n *the set of all **string of length*** n.
So, $\Sigma^* = \cup_{n \geq 0} \Sigma^n$.

Denote the length of a string w *by* $|w|$

***e.g.* 1.3.** $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, \cdots\}, |\varepsilon| = 0, |0110| = 4$

***def* 1.5.** *(concatenation) Two strings over the same alphabet can be combined by the operation of **concatenation**. The concatenation of* x *and* y *is denoted by* xy.

***def* 1.6.** *(substring, suffix, prefix)*

A string v is a substring of w if $\exists$ strings x and y such that $w = xvy$.

If $w = xv$ for some x, then v is a ***suffix*** *of w.*

If $w = vy$ for some y, then v is a ***prefix*** *of w.*

***def* 1.7.** *("power") The string w^i is defined: $w^0 = \varepsilon, w^{i+1} = w^i w, i \in \mathbb{N}$*

***e.g.* 1.4.** $01^0 = \varepsilon, 01^1 = 01, 01^2 = 0101$

***def* 1.8.** *(reversal) The reversal of a string w,denoted by w^R,is the string "spelled backwards"*

A formal definition can be given by induction on length:

1. *If $w = \varepsilon, w^R = w = \varepsilon$*

2. *If $|w| = n + 1$,where $w = ua, a \in \Sigma$,then $w^R = au^R$*

def* 1.9.** *(language)* ***Language *is a set of strings over an alphabet, That is, $L \subseteq \Sigma^*$.*

For example,$\emptyset, \Sigma^*, \Sigma$ are all languages

- $\sigma = \{0, 1\}$

- $Even = \{0, 10, 100, 110, \cdots ,\}$

- $Odd = \{1, 11, 101, \cdots\}$

- $Prime = \{10, 11, 101, 111, \cdots\}$

- $Palindrome = \{w|w^R = w\} = \{\varepsilon, 0, 1, 00, 11, \cdots\}$

***def* 1.10.** *(complement, binary language operations)*

Let L be a language. The ***complement*** *of L, denoted by $\overline{L}$, is $\Sigma^* - \overline{L}$. So $\overline{\overline{L}} = L$.*

Note that since L is a set, we can define ***union****($\cup$),* ***interchapter**** *($\cap$) and difference.*

The ***concatenation*** *of L_1 and L_2 is defined by $L_1L_2 = w \in \Sigma^*, w = xy, \exists x \in L_1, y \in L_2$*

1.3 Encoding of Problems

1.3.1 Examples

1. ***(Integer multiplication)*** *Given two non-negative integers x, y, compute xy.*

2. ***(Primality testing)*** *Given $n \in \mathbb{N}$, decide if n is a prime.*

3. ***(Hamiltonian cycle)*** *Given an undirected graph G, test if G has a Hamiltonian cycle.*

1.3.2 Analysis

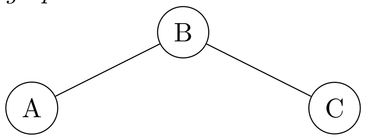
- ***Decision*** *problem: 2,3*
- ***Computation*** *problem: 1*

1.3.3 Conclusion

- *By* ***encoding****, decision problem is language. Any computation problem is a function from Σ^* to Σ^*. Our course only concerns decision problem, namely language.*
- *By* ***preprocessing****, one can switch between encodings.*

***e.g.* 1.5.**

- *graph*

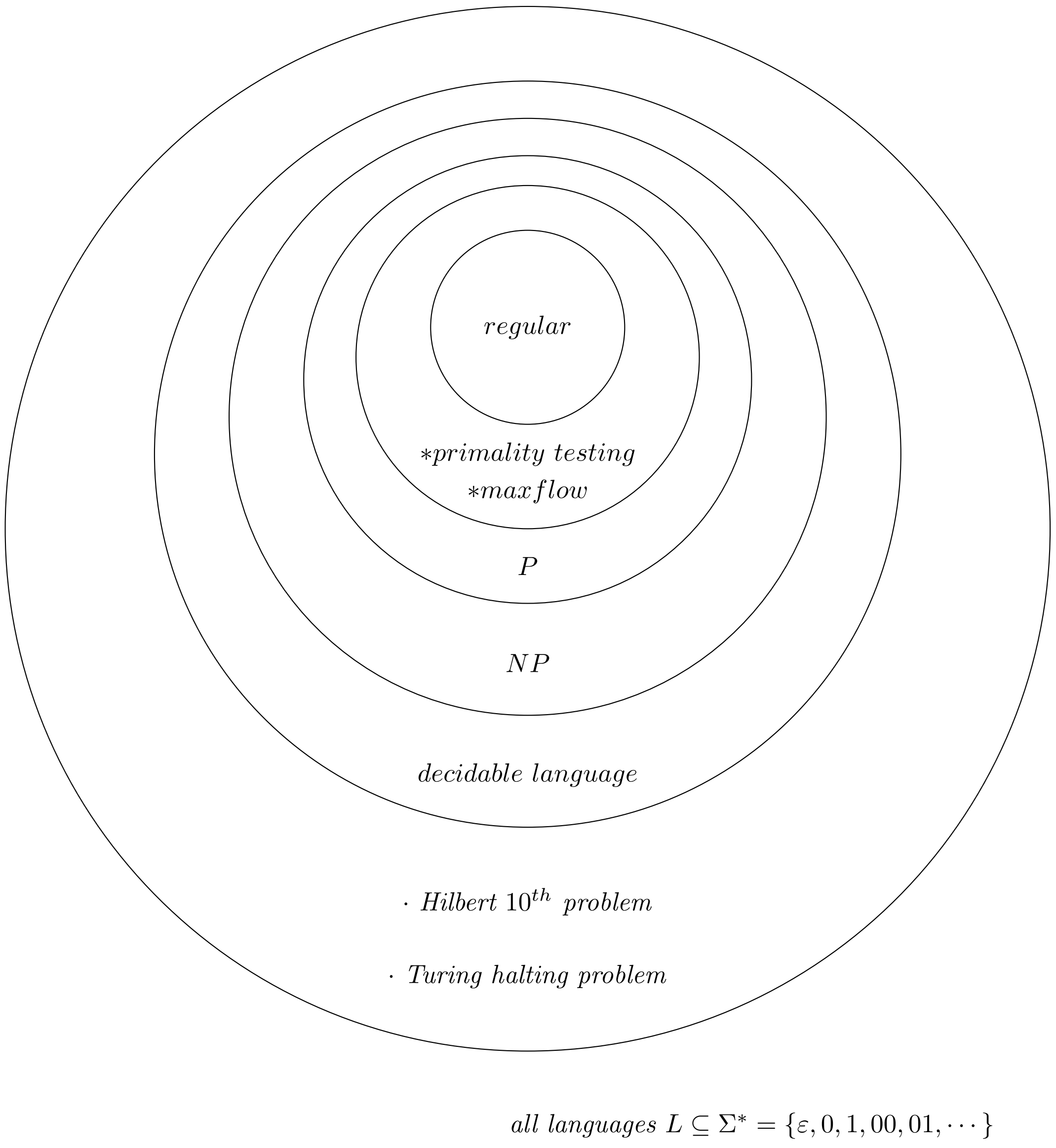


- *adjacency matrix*

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

- *adjacency list*

  (1,2),(1,3),$\cdots$


regular
$*$*primality testing*
$*maxflow$
P
NP
decidable language
· *Hilbert* 10^{th} *problem*
· *Turing halting problem*
all languages $L \subseteq \Sigma^* = \{\varepsilon, 0, 1, 00, 01, \cdots\}$

remark.

NP*: problems that are efficiently verifiable*

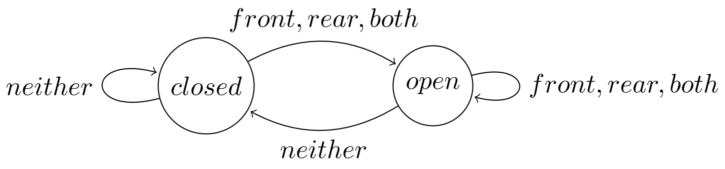
P*: problems that are efficiently solvable, i.e. solvable in* $n^{O(1)}$ *time*

regular language*: problems that are solvable without memory,i.e. solvable by finite automation*

2 Lesson 2

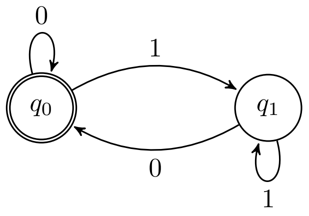
2.1 Finite Automation

***e.g.* 2.1.** *Automatic Door*



	$front$	$rear$	$both$	$neither$
$front$	✓	×	✓	×
$rear$	×	✓	✓	×

***e.g.* 2.2.** $L = \{w \in \{0,1\}^* | w = w_1 w_2 \cdots w_n, w_n = 0\}$



remark. q_0 : *accepted state*

$Q = \{q_0, q_1\}, \Sigma = \{0,1\}, \delta : Q \times \Sigma \to Q$

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

***def* 2.1.** *(finite automation)*

*A **finite automation** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:*

1. *Q is a finite set called the states*
2. *Σ is the alphabet*
3. *$\delta : Q \times \Sigma \to Q$ is the transition function*

4. *q_0 is the start state*

5. *$F \subseteq Q$ is the set of accept states*

***def* 2.2.** *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automation, let $w = w_1 w_2 \cdots w_n$ be a string, where each $w_i \in \Sigma$.*

Then M accept w if there is a sequence of states $r_0, r_1, \cdots, r_n \in Q$, such that:

1. *$r_0 = q_0$*

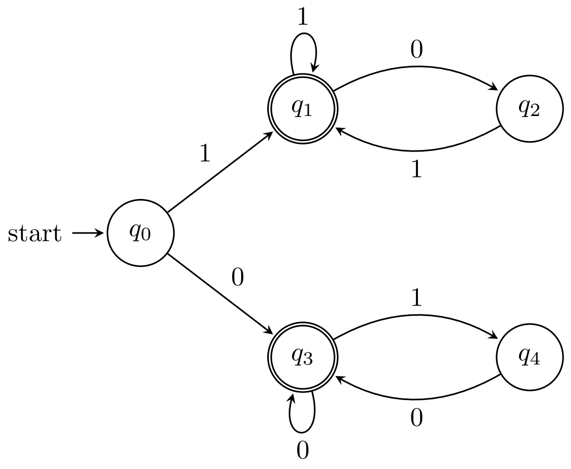
2. *$\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, 2, \cdots, n-1$*

3. *$r_n \in F$*

***def* 2.3.** *If L is the set of strings that M accepts, we say L is the language of M, and write L(M) = L, we say M recognizes/decides/accepts L.*

If M accepts no string, it recognizes one language namely, the empty language.

***e.g.* 2.3.** $L = \{w \in \{0,1\}^* | w = w_1 w_2 \cdots w_n, w_n = w_1\}$



2.2 Regular Language

***def* 2.4.** *(regular language) $L \subseteq \Sigma^*$ is a **regular language** if there is a finite automation that accepts L*

Let $A, B \subseteq \Sigma^*$, define:

- *(union) $A \cup B = \{x \in \Sigma^* | x \in A \text{ or } x \in B\}$*

- *(concatenation) $AB = \{xy | x \in A, y \in B\}$*

- *(star)* $A^* = \{x_1x_2\cdots x_k | k \geq 0, x_1, x_2, \cdots, x_k \in A\}$

***thm* 2.5.** *If A_1, A_2 are regular languages, so is $A_1 \cup A_2$*

Proof. *Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, F_1)$ accepts A_1, $M_2 = (Q_2, \Sigma_2, \delta_2, q_{20}, F_2)$ accepts A_2, construct $M = (Q, \Sigma, \delta, q_0, F)$:*

1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) | r_1 \in Q_1, r_2 \in Q_2\}$
2. $\delta : Q \times \Sigma \to Q$ *is defined as for each* $(r_1, r_2) \in Q$, *and each* $a \in \Sigma$, *let* $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
3. $q_0 = (q_{10}, q_{20})$
4. $F = \{(r_1, r_2) | r_1 \in F_1$ *or* $r_2 \in F_2\}$

□

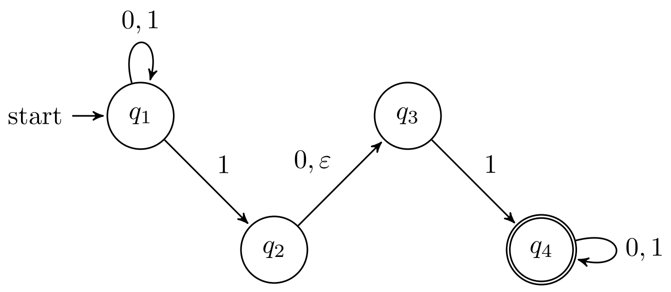
remark. *so is $A \cap B$*

***thm* 2.6.** *If A_1, A_2 are regular languages, so is A_1A_2*

- ***DFA****: deterministic finite automation*
- ***NFA****: nondeterministic...*

If at least one of these processes accepts, then the entire computation accepts.

***e.g.* 2.4.** *NFA:*



input: 010110

***e.g.* 2.5.** *Let A be the language consisting of all strings over $\{0, 1\}$ containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA recognizes A.*

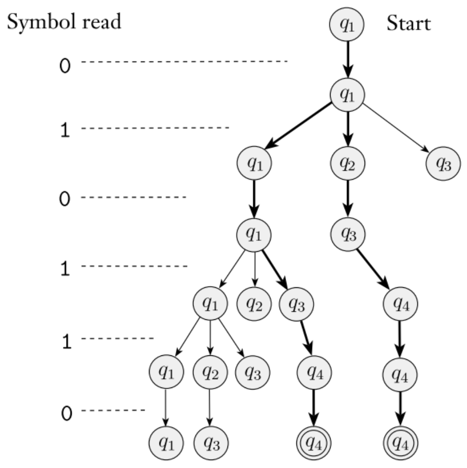


Figure 1: *The computation of NFA on input 010110*

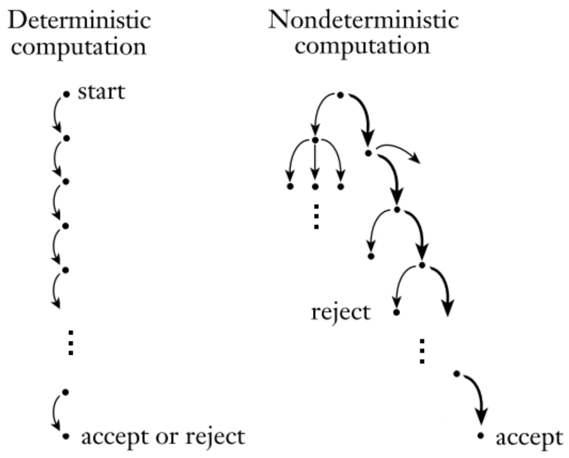
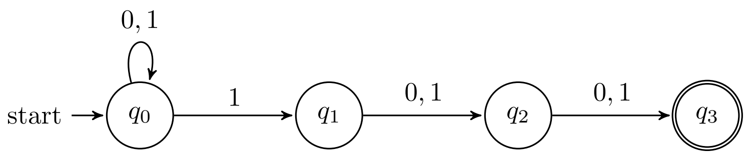
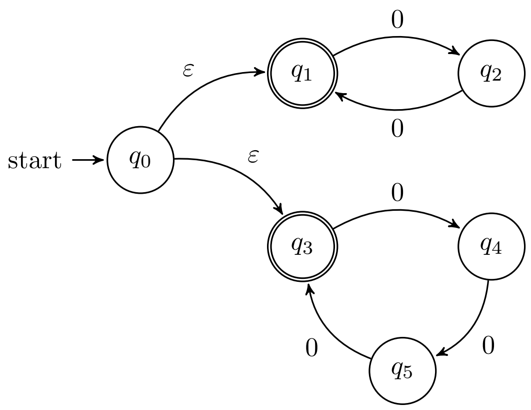


Figure 2: *Deterministic and nondeterministic computations with an accepting branch*



***e.g.* 2.6.** $L = \{0^k, 2|k \text{ or } 3|k\}$



***def* 2.7.** *(NFA)*

*An **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. *$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ is the transive funtion*
4. *$a_0 \in Q$ is the start state*
5. *$F \subseteq Q$ is the set of accept states*

***def* 2.8.**

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and let $w \in \Sigma^*$. Say N accepts w if we can write $w = y_1 y_2 \cdots y_m$, where $y_i \in \Sigma \cup \{\varepsilon\}$, and there exist $r_0, r_1, \cdots, r_m \in Q$, 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. $r_m \in F$

***thm* 2.9.** *Every NFA has an equivalent DFA*

Proof. □