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 Introduction

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:* $O(1)$
  - $\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$
  - $\sum_{k=1}^{n} \frac{1}{k^2} = O(1)$
  - $\sum_{k=1}^{n} \frac{1}{k \ln k} = \ln \ln n + O(1)$
- *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 $L_1 L_2 = w \in \Sigma^*, w = xy, \exists x \in L_1, y \in L_2$*

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

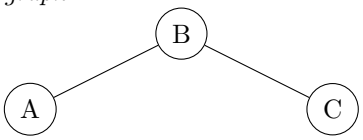
- ***Decision*** *problem: 2,3*
- ***Computation*** *problem: 1*

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*

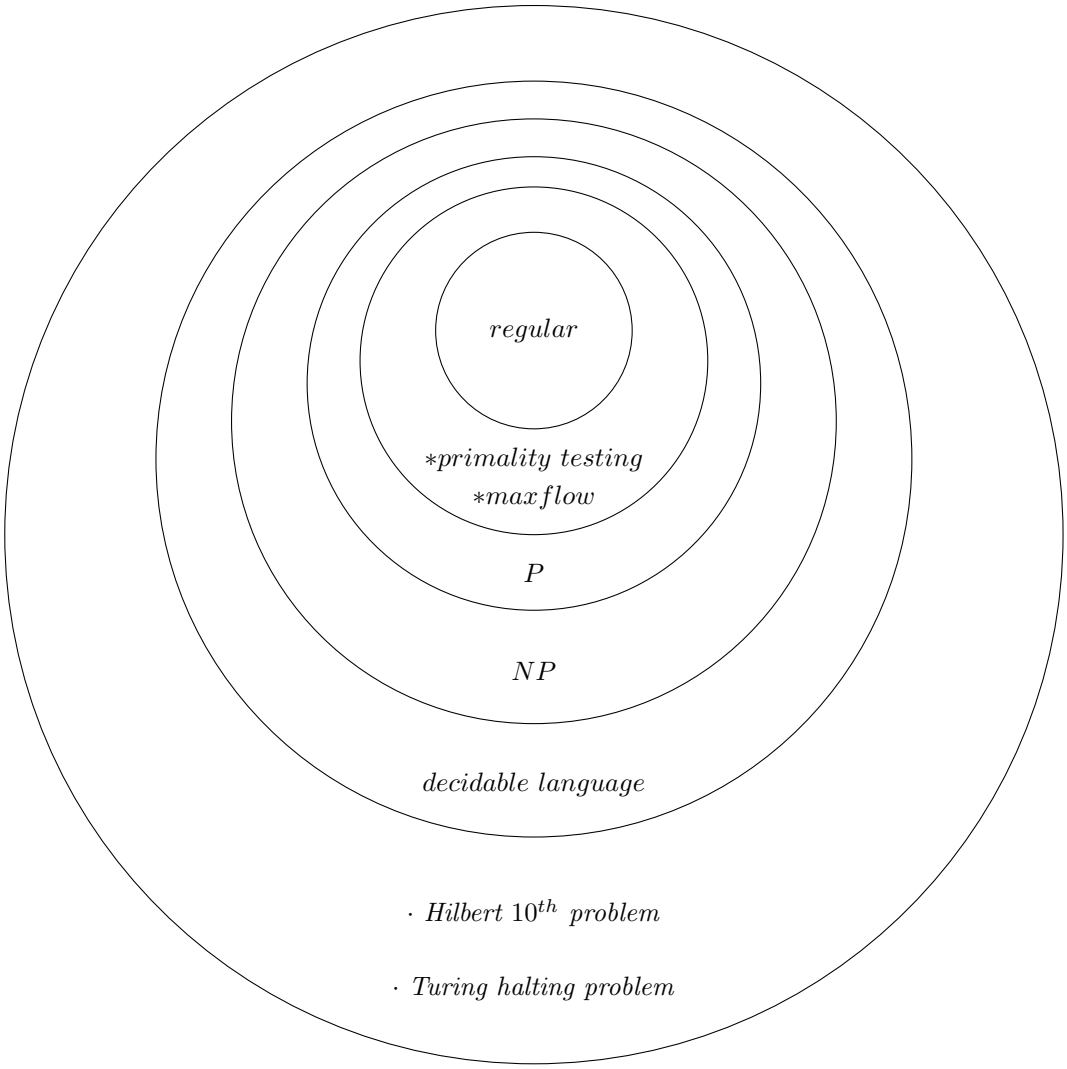


- *adjacency matrix*

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

- *adjacency list*

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



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*

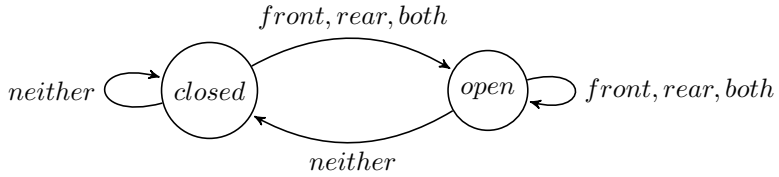
number of regular languages: $\aleph_0$

number of all languages: $\aleph_1$

2 Automata and Language

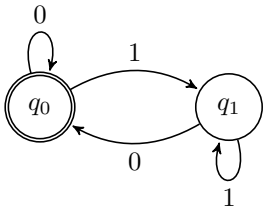
2.1 Finite Automaton

***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 automaton)*

A ***finite automaton*** *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 automaton, 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. $r_0 = q_0$

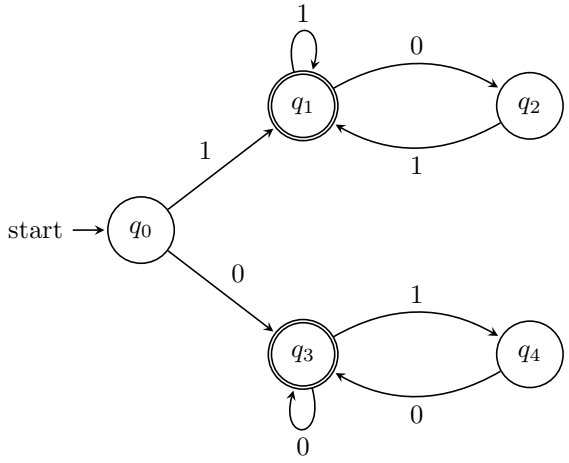
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, *for* $i = 0, 1, 2, \cdots, n-1$

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 automaton 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)* $A^* = \{x_1 x_2 \cdots x_k | k \geq 0, x_1, x_2, \cdots, x_k \in A\}$

***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. $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$

□

remark. *so is $A \cap B = \overline{\overline{A} \cup \overline{B}}$*[1]

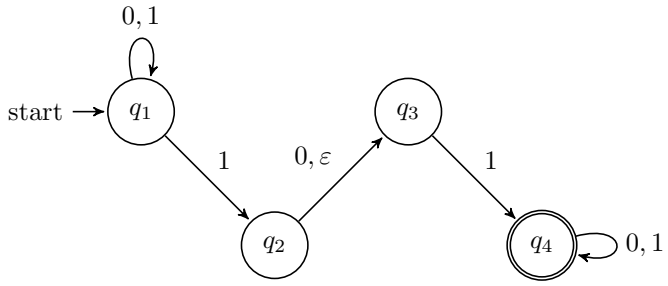
2.3 DFA and NFA

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

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

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

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



input: 010110

[1] *proof of the closure under complement will be mentioned later*

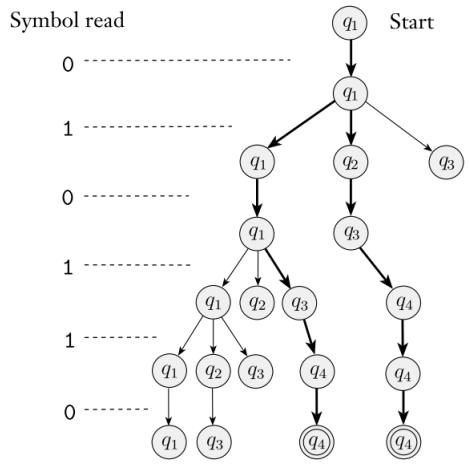


Figure 1: *The computation of NFA on input 010110*

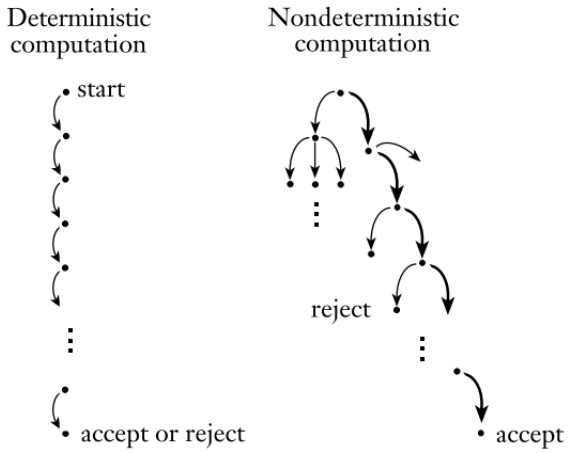
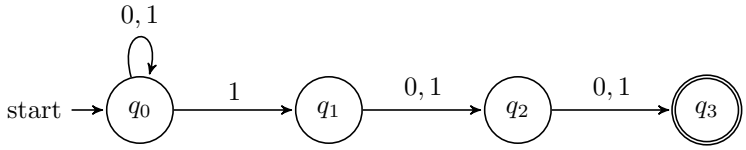


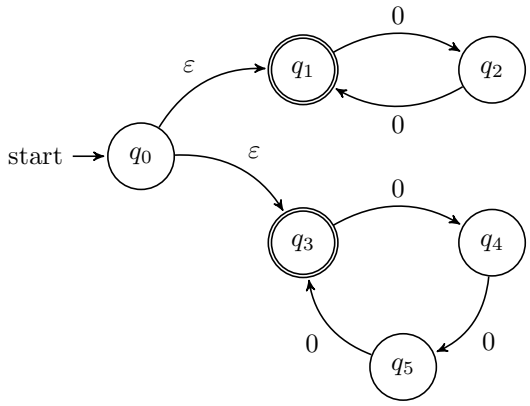
Figure 2: *Deterministic and nondeterministic computations with an accepting branch*

remark. *If a language can be accepted by DFA, then its time complexity is O(n), space complexity is O(1).*

***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.*



***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 function*
4. $q_0 \in Q$ *is the start state*
5. $F \subseteq Q$ *is the set of accept states*

remark. $P(Q)$ *is the collection of all the subsets of Q (power set)*

- *variant1:* $\delta : Q \times \Sigma^* \to Q$

  We guarantee there is at most one applicable transition.

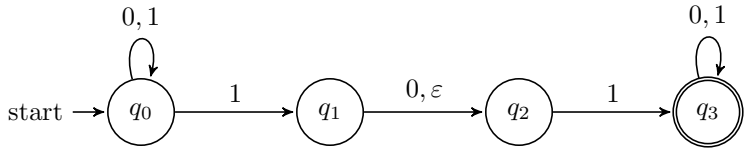
- *variant2: If there are multiple applicable transitions,non-deterministically choose one.*

***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. $r_m \in F$

***e.g.* 2.7.** $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}, F = \{q_3\}, \delta : Q \times \Sigma \to Q$



$q \backslash \Sigma$	0	1	ε
q_0	$\{q_0\}$	$\{q_0, q_1\}$	$\emptyset$
q_1	$\{q_2\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\{q_3\}$	$\{q_3\}$	$\emptyset$

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

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing A. Construct a DFA $M = (Q', \Sigma, \delta', q_0', F)$ recognizing A:

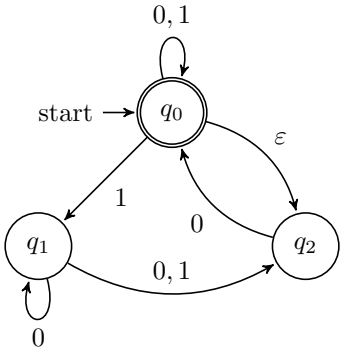
Let $E(R) = \{q \in Q |$ q can be reached from R by traveling along zero or more ε arrows$\}$

Define M as follows:

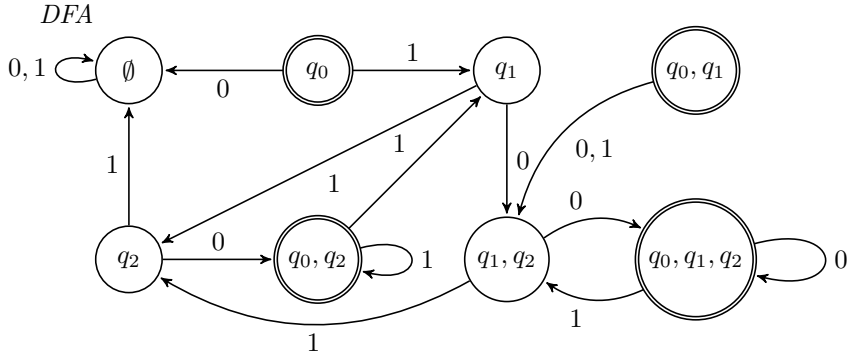
1. $Q' = P(Q)$
2. *For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q : q \in E(\delta(r, a))$ for some $r \in R\} = \bigcup_{r \in R} E(\delta(r, a))$*
3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q', R \cap F \neq \emptyset\}$

□

***e.g.* 2.8.** *NFA*



DFA



***corollary* 2.10.** *A language is a regular language iff an NFA recognizes it.*

***thm* 2.5.** *The class of regular languages is closed under union.*

Second proof

Proof. See figure 3.

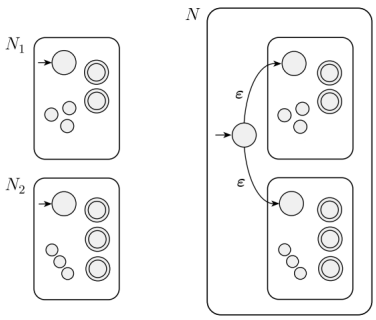


Figure 3: *Construction of an NFA N to recognize $A_1 \cup A_2$*

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recignize* A_1 *and* $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *recognize* A_2
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup Q_2 \cup \{q_0\}$
2. q_0 *is the start state*
3. $F = F_1 \cup F_2$
4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

$$\textit{Let } \delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q \\ \delta_2(q,a) & q \in Q \\ \{q_1, q_2\} & q = q_0 \textit{ and } a = \varepsilon \\ \emptyset & q = q_0 \textit{ and } a \neq \varepsilon \end{cases}$$

□

***thm* 2.11.** *The class of regular languages is closed under concatenation.*

Proof. See figure 4. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recignize* A_1 *and* $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *recognize* A_2
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup Q_2$
2. q_1 *is the start state*
3. $F = F_2$

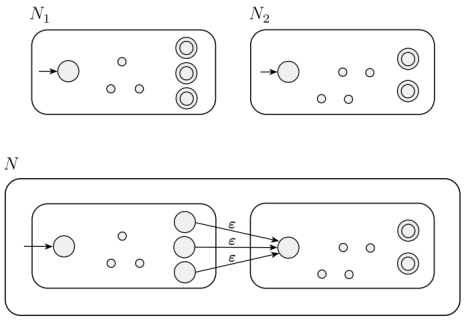


Figure 4: *Construction of an NFA* N *to recognize* A_1A_2

4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

$$\textit{Let } \delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \textit{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \textit{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \textit{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

***thm* 2.12.** *The class of regular languages is closed under star.*

Proof. See figure 5.

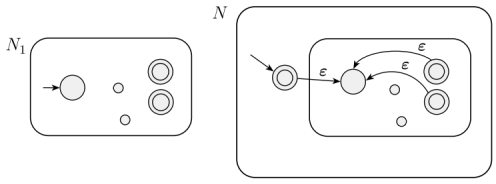


Figure 5: *Construction of an NFA N to recognize* A_1^*

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recognize* A_1
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup \{q_0\}$
2. q_0 *is the start state*
3. $F = \{q_0\} \cup F_1$
4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

$$\textit{Let } \delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \textit{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \textit{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \textit{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \textit{ and } a = \varepsilon \\ \emptyset & q = q_0 \textit{ and } a \neq \varepsilon \end{cases}$$

□

***thm* 2.13.** *The class of regular languages is closed under complement.*

Proof. *Let DFA* $M = (Q, \Sigma, \delta, q_0, Q_{accept})$ *recognizing* A*, then* $\overline{A} = \Sigma^* - A$*,*
construct $M' = (Q, \Sigma, \delta, q_0, Q'_{accept}), Q'_{accept} = Q - Q_{accept}$*, it's easy to prove* $L(Q') = \overline{A}$*.* □

2.4 Regular Expression

***e.g.* 2.9.** *Consider students' ID number, with boys' ending with an odd number, while girls' ending with even number.*

$boys : (0 \cup 1 \cup \cdots \cup 9)^*(1 \cup 3 \cup 5 \cup 7 \cup 9)$

***def* 2.14.** *(regular expression)*

R is a ***regular expression*** *if R is:*

1. a *for some* $a \in \Sigma$
2. ε
3. $\emptyset$
4. $R_1 \cup R_2$*, where* R_1, R_2 *are regular expressions*
5. $R_1 R_2$*, where* R_1, R_2 *are regular expressions*
6. R^**, where* R *is a regular expression*

***e.g.* 2.10.**

1. 0^*10^*
2. $\Sigma^*1\Sigma^* = \{w | w = \cdots 1 \cdots\}$
3. $\Sigma^*001\Sigma^* = \{w | w = \cdots 001 \cdots\}$
4. $1^*(01^+)^* = \{w \in \{0,1\}$*, every 0 in w is followed by at least 1*$\}$
5. $(\Sigma\Sigma)^* = \{w |$*the length of w is even*$\}$
6. $1^*\emptyset = \emptyset$

***exercises* 2.1.**

1. *w contains substring 110:* $\{\Sigma^*110\Sigma^*\}$
2. *w doesn't contain 00 as substring:* $(0 \cup \varepsilon)(1 \cup 10)^*$

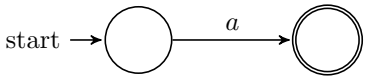
3. *the number of 1s is a multiple of 3:* $\{(0^*10^*10^*10^*)^*\}$
4. *w contains at least two 1s and one 0:* $(\Sigma^*1\Sigma^*1\Sigma^*0\Sigma^*) \cup (\Sigma^*1\Sigma^*0\Sigma^*1\Sigma^*) \cup (\Sigma^*0\Sigma^*1\Sigma^*1\Sigma^*)$

***thm* 2.15.** *A language is regular iff some regular expression described it*

***lemma* 2.16.** ($\Leftarrow$) *If a language is decribed by a regular expression, then it's regular*

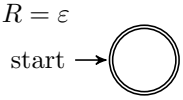
Proof. Let's convert R into an NFA N

1. $R = a, a \in \Sigma$

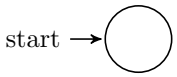
   Note that this machine fits the definition of an NFA but not that of a DFA because it has some states with no exiting arrow for each possible input symbol. Of course, we could have presented an equivalent DFA here; but an NFA is all we need for now, and it is easier to describe. Formally, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$*, where we describe* δ *by saying that* $\delta(q_1, a) = \{q_2\}$ *and that* $\delta(r, b) = \emptyset$ *for* $r \neq q_1$ *or* $b \neq a$*.*

2. $R = \varepsilon$

   Formally, $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$*, where we describe* δ *by saying that* $\delta(q_1, a) = \emptyset$ *for* $\forall r, b$*.*

3. $R = \emptyset$

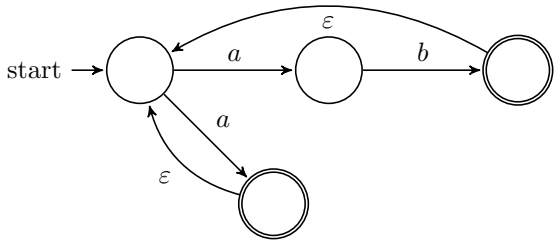
   Formally, $N = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$*, where we describe* δ *by saying that* $\delta(q_1, a) = \emptyset$ *for* $\forall r, b$*.*

4. $R = R_1 \cup R_2$
5. $R = R_1 R_2$
6. $R = R_1^*$

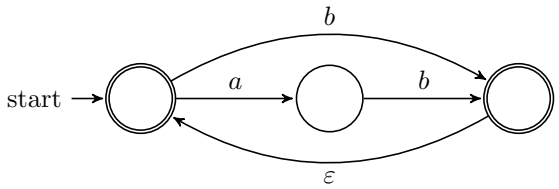
   For the last three cases, we use the constructions given in the proofs that the class of regular languages is closed under the regular operations. In other words, we construct the NFA for R from the NFAs for R_1 *and* R_2 *(or just* R_1 *in case 6) and the appropriate closure construction.*

□

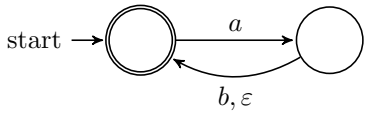
***e.g.* 2.11.** $(ab \cup a)^*$



It can be simplified as below:



Or as this:



Now let's turn to the other direction of the proof of Theorem 2.15.

***lemma* 2.17.** (⇒) *If a language is regular, then it is described by a regular expression.*

PROOF IDEA *We need to show that if a language A is regular, a regular expression describes it. Because A is regular, it is accepted by a DFA. We describe a procedure for converting DFAs into equivalent regular expressions.*

Proof. We break this procedure into two parts, using a new type of finite automaton called a ***generalized nondeterministic finite automaton***, ***GNFA***.

DFA → GNFA → regular expressions □

***def* 2.18.** *A generalized nondeterministic finite automaton is a 5-tuple,* $(Q, \Sigma, \delta, q_{start}, q_{accept})$*, where:*

1. Q *is a finite set of states*
2. Σ *is the input alphabet*
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$ *is the transive function*

4. q_{start} *is the start state*

5. q_{accept} *is the accept state*

remark. GNFA *requires:*

1. *The start state has transition arrows going to every other state but no arrows coming in from any other state.*

2. *There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.*

3. *Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.*

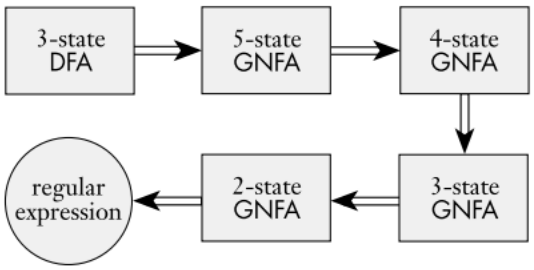


Figure 6: *Typical stages in converting a DFA to a regular expression*

Let's finish the proof for Lemma 2.17

Proof.

1. *Add a new start state with* ε *arrow to the old start state*

2. *Add a new accept state with* ε *arrow(s) from old accept states*

3. *Replace multiple lables or arrows between two states with a single arrow whose label is the union of the previous lables*

4. *Add arrows with* $'\emptyset'$ *between states that had no arrows*

Now consider convert ***GNFA*** *to* ***regular expression***

We use the procedure ***CONVERT(G)***

1. *Let k be the number of states in G*
2. *If $k = 2$, return $\delta(q_{start}, q_{accept})$*
3. *If $k > 2$, choose $q_{rip} \in Q \backslash \{q_{start}, q_{accept}\}$*

   Let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where:

   (a) $Q' = Q - q_{rip}$

   (b) $\forall q_i \in Q' - \{q_{qccept}\}, \forall q_j \in Q' - \{q_{start}\}$

   Then $\delta'(q_i, q_j) = \delta(q_i, q_j) \cup (\delta(q_i, q_{rip})\delta(q_{rip}, q_{rip})^*\delta(q_{rip}, q_j))$

   (see Figure7)

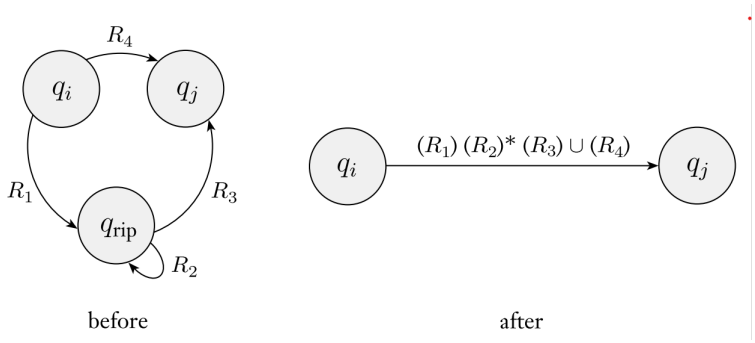


Figure 7: *Constructing an equivalent GNFA with one fewer state*

remark. *$q_i = q_j$ is possible*

4. *compute convert(G') and return*

claim* 2.19.** *For any GNFA G,* ***CONVERT(G) *is equivalent to G.*

Proof. We prove this claim by induction on k, the number of states of the GNFA.

Basis:*Prove the claim true for $k = 2$ states. If G has only two states, it can have only a single arrow, which goes from the start state to the accept state. The regular expression label on this arrow describes all the strings that allow G to get to the accept state. Hence this expression is equivalent to G.*

Induction step: *Assume that the claim is true for k - 1 states and use this assumption to prove that the claim is true for k states. First we show that G and G′ recognize the same language. Suppose that G accepts an input w. Then in an accepting branch of the computation, G enters a sequence of states:* $q_{start}, q_1, q_2, \cdots, q_{accept}$

If none of them is the removed state q_{rip}*, clearly G′ also accepts w. The reason is that each of the new regular expressions labeling the arrows of G′ contains the old regular expression as part of a union.*

If q_{rip} *does appear, removing each run of consecutive* q_{rip} *states forms an accepting computation for G′. The states* q_i *and* q_j *bracketing a run have a new regular expression on the arrow between them that describes all strings taking qi to* q_j *via* q_{rip} *on G. So G′ accepts w.*

Conversely omitted.

The induction hypothesis states that when the algorithm calls itself recursively on input G′ , the result is a regular expression that is equivalent to G′ because G′ has k - 1 states. Hence this regular expression also is equivalent to G, and the algorithm is proved correct □

Thus ***Lemma 2.17*** *and* ***Theorem 2.15*** *are proved* □

e.g. **2.12.**

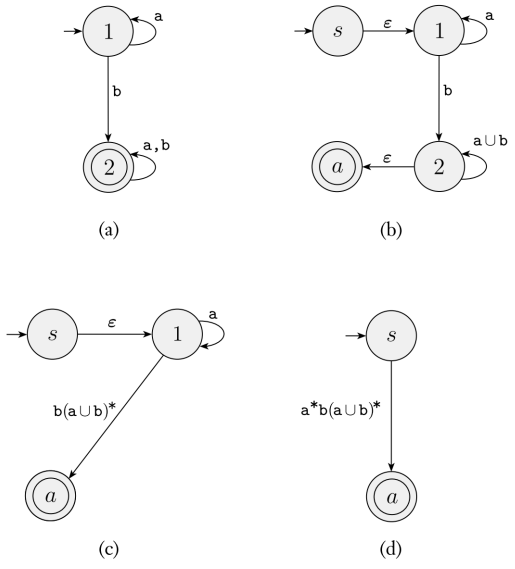


Figure 8: *Converting a two-state DFA to an equivalent regular expression*

2.5 Nonregular Languages

***e.g.* 2.13.** *Consider a language* $B = \{0^n 1^n | n \geq 0\}$

If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input. Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

"Almost all languages are non-regular"

lemma* 2.20.** ***(pumping lemma)

If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$*, satisfying the following conditions:*

1. *for each* $i \geq 0, xy^i z \in A$
2. $|y| > 0$
3. $|xy| \leq p$

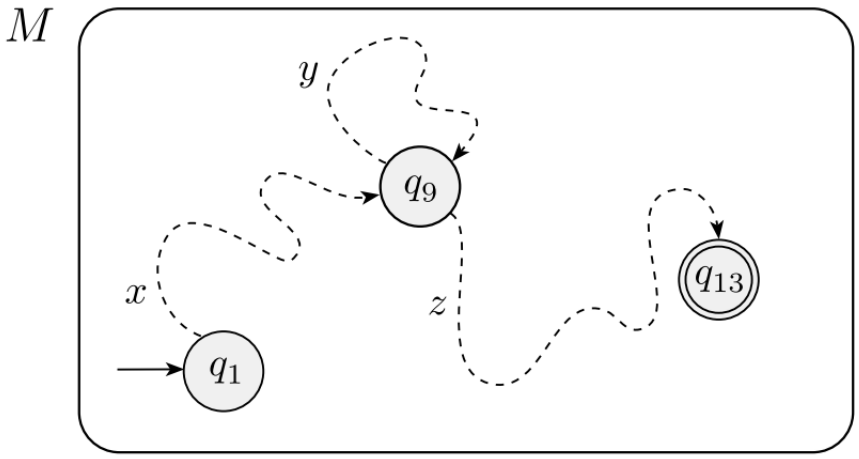


Figure 9: *Example showing how the strings x, y, and z affect M*

Proof. Let $M = (Q, \Sigma, \delta, q_1, F)$ recognizing A and p be the number of states of M.

Let $s = s_1 s_2 \cdots s_n$ be a string in A of length n, where $n \geq p$. Let $r_1, r_2, \cdots, r_{n+1}$ be the sequence of states that M enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. This sequence has length $n + 1$, which is at least $p + 1$. Among the first $p + 1$ elements in the sequence, two must be the same state, by the ***pigeonhole principle****.We call the first of these r_j and the second r_l. Because r_l occurs among the first $p + 1$ places in a sequence starting at r1, we have $l \leq p + 1$.*
Now let

$$\begin{cases} x = s_1 \cdots s_{j-1} \\ y = s_j \cdots s_{l-1} \\ z = s_l \cdots s_n \end{cases}$$

it's easy to prove this satisfy 3 conditions □

***exercises* 2.2.**

1. *$L = \{0^n 1^n | n \geq 0\}$ is non-regular*

   Proof. Let $n = p$,consider $s = 0^n 1^n$. By ***pumping lemma****, write:*
   $s = xyz, |xy| \leq p = n$, then $x, y = 0^*, xy^i z \notin L, \forall i \neq 1$, contradiction! □

3 Algorithm and Turing Machines

3.1 Turing machines

An algorithm is a mechanical process to be followed in calculations or other problem-solving operation.

***e.g.* 3.1.**

1. *addition,substraction,multiplication,division*
2. ***Euclidean algorithm*** *for gcd:* $gcd(210, 25) = gcd(45, 30) = gcd(30, 15) = gcd(15, 0)$
3. *selection quicksort*
4. *Dijkstra's algorithm for shortest path*

In 1936 for the first time, Alan Turing rigorously defined algorithms

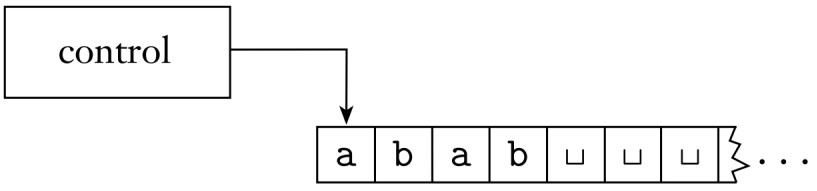


Figure 10: *Schematic of a Turing machine*

1. *It uses an infinite tape as its unlimited memory*
2. *It has a tape head that can read and write symbols and move around on the tape*

***def* 3.1.** *(Turing Machine)*

A ***Turing machine*** *is a 7-tuple,* $(Q, \Sigma, \Gamma, \delta, q_0, \underbrace{q_{accept}, q_{reject}}_{q_{end}})$*, where:*

1. Q *is a set of states*
2. Σ *is the input alphabet not containing the blank symbol* $\sqcup$
3. Γ *is the tape alphabet, where* $\sqcup \in \Gamma$ *and* $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \{L, R\}$ *is the transive function*
5. $q_0 \in Q$ *is the start state*

6. $q_{accept} \in Q$ *is the accept state*

7. $q_{reject} \in Q$ *is the reject state, where* $q_{accept} \neq q_{reject}$

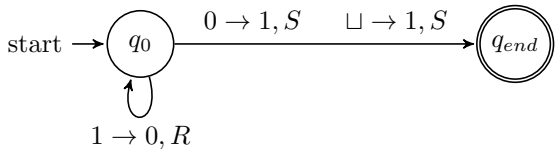
***e.g.* 3.2.**

1. *Input a binary number n (least significant bit first), output n+1*

   Input: $101 \sqcup \sqcup$

   Output: $011 \sqcup \sqcup$

   remark. *this is a computation problem(6-tuple)*



2. *decide if* $w \in \{0,1\}^*$ *is a palindrome, i.e.* $w = w^R$

3. *decide* $L = \{0^{2^n}, n \geq 0\}$*, where* $\Sigma = \{0\}$

   IDEA

   (a) *If there is a single 0, accept it*

   (b) *Sweep left to right across the tape, crossing off every other 0.*

   (c) *If the tape contained more than a single 0 and the number of 0s was odd, reject*

   (d) *Return the head to the left-hand end of the tape*

   (e) *go to step1*

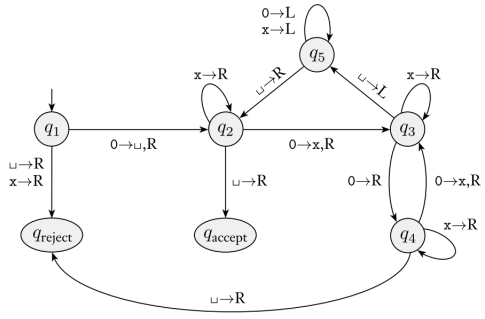


Figure 11: *State diagram for Turing machine M2*

4. $L = \{w\#w, w \in \{0,1\}^*\}$

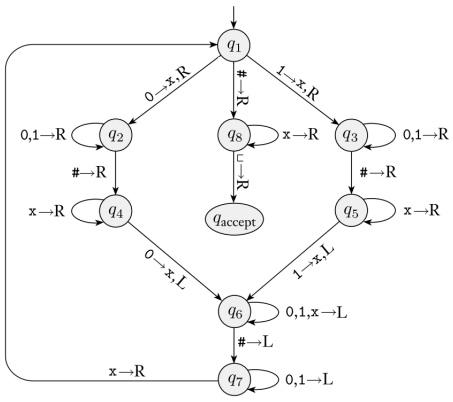


Figure 12: *State diagram for Turing machine M1*

***exercises* 3.1.**

1. *Binary comparison.* $L = \{x, y | x, y \in \{0,1\}^* x \geq y$, *most significantbit first, no leading 0 unless the number is 0*$\}$
2. *Binary add 1*
3. $L = \{0^n 1^n\}$

***def* 3.2.** *Let* $L \subset \{0,1\}^*$, *M be a Turing Machine. Say M decides L in time T(n) if for* $\forall x \in \{0,1\}^*$*:*

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

***def* 3.3.** *Let* $L \subseteq \{0,1\}^*$. *Call L(Turing Machine) decidable if there is a Turing Machine decides it*

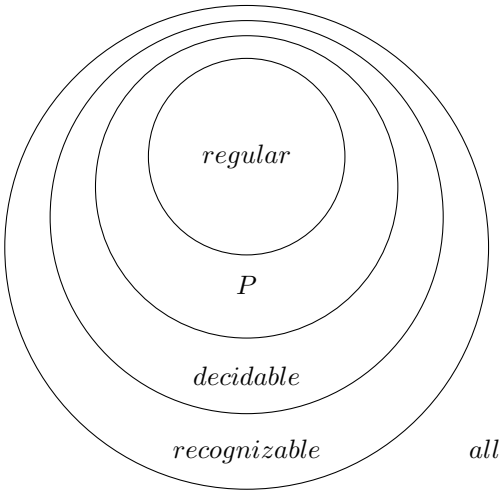
remark. *On an input x, a TM may accept, reject, or loop forever*

In ***definition 3.2****, the machine should never loop forever*

***def* 3.4.** *Let M be a TM, the set of strings that M accepts is the language recognized by M, denoted by L(M)*

***def* 3.5.** *Let* $L \subseteq \{0,1\}^*$. *Call L(Turing) recognizable if there is some TM recognizes it*

remark. *Obviously, every decidable language is recognizable. While the converse is not true, e.g.* $L = \{< M, x >, M \text{ halts on } x\}$



***def* 3.6.** *Let* $f : \{0,1\}^* \to \{0,1\}^* \cup \{undefined\}$. *Say TM M computes f in time T(n) if for* $\forall x \in \{0,1\}^*$, *with* $f(x) \neq undefined$, *M halts with f(x) on its tape in at most* $T(|x|)$ *steps*

What is algorithm? An algorithm is A Turing Machine. Despite its simplicity, it is capable of implementing any computer algorithm.
"Everything should be made as simple as possible, but no simpler."

3.2 Variants of Turing Machines

***lemma* 3.7.** *If language* $L \subseteq \{0,1\}^*$ *is decidable in time T(n) by a Turing Machine on alphabet* Γ, *then it is decidable in time* $O(log|\Gamma|T(n)) = O_\Gamma(T(n))$ *by a Turing Machine on alphabet* $\Gamma = \{0, 1, \sqcup, \triangleright\}$

Proof. Encode any symbol in Γ *using* $k = \lceil log_2|\Gamma| \rceil = O(log|\Gamma|)$ *bits. To simulate one step of M, the new Turing Machine will:*

1. *use k steps to read a symbol* $a \in \Gamma$
2. *transit to next step q', and get the new symbol b(to overwrite a)*
3. *overwrite a by b*
4. *go left or right for k steps, or stay*

In total, the simulation (for one step) takes less than k+1+k+k = O(k) □

***def* 3.8.** *A k-tape Turing Machine M is a 7-tuple* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$*, where:*

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

Usually, the first tape is the input tape, the last tape is the output tape, and the remaining tapes are work tapes.

A multiple Turing Machine is an O(1)-tape Turing Machine.

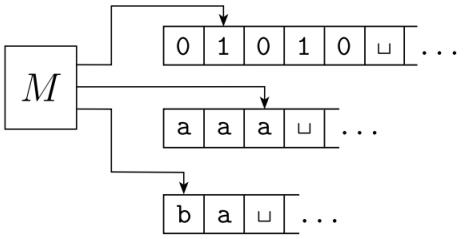


Figure 13: *3-tape Turing Machine*

***lemma* 3.9.** *Let* $L \subseteq \Sigma^*$*, If L is decidable by a k-tape Turing Machine in time T(n), then it is decidable by a single-tape Turing Machine in time* $O(kT(n)^2) \neq O(T(n)^2)$

Proof. Use location $i-1, k+i-1, 2k+i-1, ...$ *to store the contents of the* i^{th} *tape, where* $i = 1, 2, ..., k$*. For* $\forall a \in \Gamma$*, introduce* $a, \widehat{a} \in \Gamma$*, where* $\widehat{a}$ *denotes the location of the head.*

To simulate one step of M, single-tape Turing Machine M' will:

1. *sweep the tape from left to right to read k symbols marked by* $\widehat{\ }$
2. *apply M's transition function* δ *to determine the next state*
3. *sweep back from right to left to update k symbols, if needed, and move* $\widehat{\ }$ *if needed*

In total, 1.2.3. take T(n)+1+O(kT(n)) = O(kT(n)) □

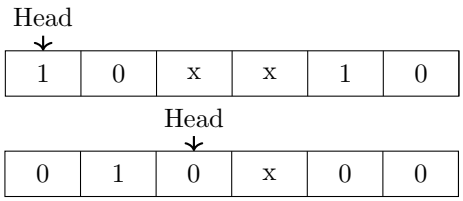


Figure 14: A two tape Turing Machine

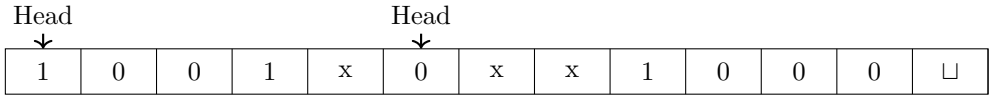


Figure 15: Representing two tapes with one

*A **Bidirectional-tape Turing Machine** is a Turing Machine whose tape is infinite in both directions*

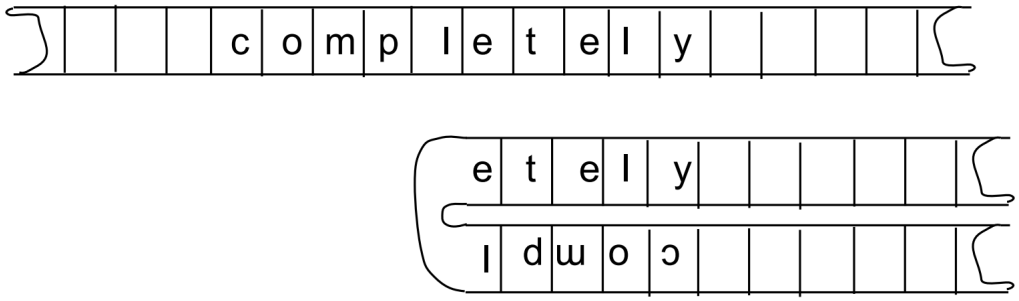


Figure 16: *A bidirectional-tape Turing Machine*

***lemma* 3.10.** *Let $L \subseteq \Sigma^*$. If L is decidable by a bidirectional-tape Turing Machine in T(n), then it is decidable by a single-tape Turing Machine in O(T(n)) time.*

Proof. Index the biderectional tape by $\mathbb{Z}$, map location i to

$$\begin{cases} 2i, i \geq 0 \\ -2i - 1, i < 0 \end{cases}$$

For every step of M, M' will

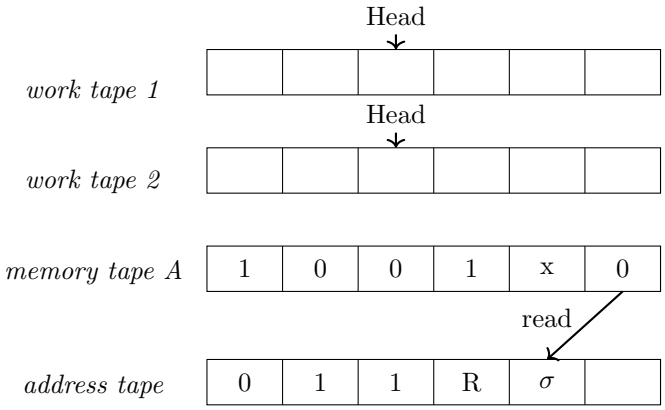
1. *read the symbol*
2. *transit to the next state*
3. *update the symbol*
4. *move left or right for two steps, if needed*

It takes O(1) to simulate one step. In total, the running time is O(T(n)) □

***def* 3.11.** *Random access memory(RAM) Turing machine is a TM with random access memory:*

1. *M has an infinite memory tape A indexed by N*
2. *One of M's tapes is the address tape*
3. *Γ contains two speed symbols R(read) and W(write)*
4. *Q has some special states $Q_{access} \subseteq Q$*
   Whenever M gets into a state $q \in Q_{access}$
   (a) *If the address tape contains iR, the value $A[i]$ is written to the cell next to R*
   (b) *If the address tape contains $iW\sigma$, then set $A[i]$ to symbol σ*

Assume the TAM Turing Machine M has k work tapes and an address.
Then $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}, Q_{access})$, $\delta : Q \times \Gamma^{k+1} \to Q \times \Gamma^{k+1} \times \{L, R, S\}^{k+1}$



***lemma* 3.12.** *Let $L \subseteq \{0,1\}^*$. If L is decidable by an RAM Turing Machinein time T(n), then it is decidable by a multi-tape Turing Machine in time $O(T(n)^3)$. Moreover, if the length of the address is O(1), then L is decidable by a multi-tape Turing Machine in $O(T(n)^2)$*

Proof. Use an extra work tape as memory that contains pairs $(i, A[i])$, where i is an integer in binary, $A[i] \in \Gamma$, for all memory addresses that have been referred to.

To simulate one step of M, if M is in an access state, the new multi-tape Turing Machine M' will:

1. scans tape A to find an address that matches i in the address tape
2. if i does not exist, add a new pair $(i, A[i])$
3. read or write $A[i]$ accordingly

1,2,3 take $O(T(n)^2) + O(T(n)) + O(T(n)) = O(T(n)^2)$

In total, M' runs in time $T(n) * O(T(n)) = O(T(n)^3)$

If address length is $O(1)$, then $\#pairs \leq T(n)$, length of each pair is $O(1)$,1,2,3 take O(T(n)) steps to simulate □

remark. *Ignoring polynomial factors, all Turing Machine variants are equivalent.*

$$\underbrace{C++}_{T(n)} \to \underbrace{AssemblyLanguage}_{O(T(n))} \to \underbrace{RAMTM}_{O(T(n))} \to \underbrace{multitapeTM}_{O(T(n^3))(O(T(n^2)))} \to \underbrace{single-tapeTM}_{O(T(n^6))(O(T(n^4)))}$$

***def* 3.13.** *Let $T : \mathbb{N} \to \mathbb{N}$, language $L \subseteq \{0,1\}^*$ is in **DTIME**(T(N)) iff there exists a multi-tape Turing Machine M that decides L in time O(T(n))*

***def* 3.14.** $P = \bigcup_{c \geq 1} DTIME(n^c)$

4 Computability

Encoding of a multi-tape Turing Machine*: Assume our encoding of the TMs satisfy the following properties*

1. *Every string $\alpha \in \{0,1\}^*$ represents some TM (On invalid encoding α, M_α always reject)*
2. *Every Turing Machine is represented by infinitely many strings*
   TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

***thm* 4.1.** *(Universal Turing Machine)*

There exists a multitape Turing Machine $\mathcal{U}$, s.t. $\forall x, \alpha \in \{0,1\}^*$,$\mathcal{U}(x, \alpha) = M_\alpha(x)$. Moreover, if M_α halts on input x within T steps, then $\mathcal{U}(x, \alpha)$ halts in $O_M(T \log T)$ steps (weaker version $O_{M_\alpha}(T(n))^2$)

***lemma* 4.2.** *Almost all languages are undecidable*

Proof. $\#languages = 2^{\aleph_0} = \aleph_1$

$\#\{L \subseteq \{0,1\}^*\}$

$\#TMs = \aleph_0$ □

thoughts: ***diagonalization***

$def\ L_{flip} = \{\alpha : M_\alpha \text{ does not accept } \alpha\}$

***lemma* 4.3.** *L_{flip} is undecidable*

Proof. Assume for contradiction that L_{flip} is decided by a TM M_β, which implies that $L(M_\beta) = L_{flip}$

- *case1: $\beta \in L_{flip}$. By definition M_β does not accept β,i.e, M_β rejects β. So, $\beta \notin L(M_\beta) = L_{flip}$. Contradiction!*
- *case2: $\beta \notin L_{flip}$. By definition, M_β accepts β. So, $\beta \in L(M_\beta) = L_{flip}$. Contradiction!*

□

Turing halting problem

$L_{halt} = \{(\alpha, x), M_\alpha \text{ halts on } x\}$

Fermat's Last Theorem

$(\forall m \geq 3)(\forall a, b, c \geq 1)(a^m + b^m \neq c^m)$

M_α {
- $T = 2$
- *while true*
- $T = T + 1$
- *for* $d = 3$ *to* T
- *for* $a, b, c \in \{1, 2, ..., T\}$
- $if(a^d + b^d = c^d) \rightarrow exit$

}

FLT iff $(M_\alpha, \varepsilon) \notin L_{halt}$

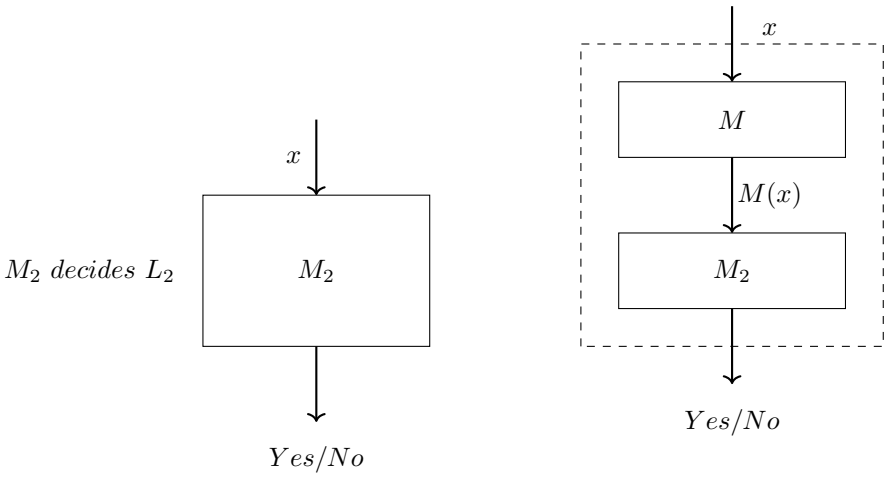
****reduction***

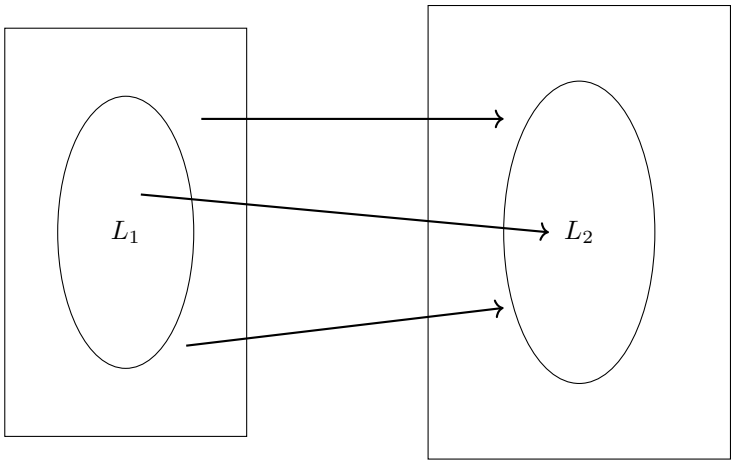
***def* 4.4.** *Let* $L_1, L_2 \subseteq \{0,1\}^*$. *Write* $L_1 \leq L_2$ *if there is a reduction from* L_1 *to* L_2 *that, there exists a TM* $M : \{0,1\}^* \rightarrow \{0,1\}^*$ *(On any input x, M always halts and outputs a string M(x)), s.t.*

1. $(\forall x \in L_1)(M(x) \in L_2)$
2. $(\forall x \notin L_1)(M(x) \notin L_2)$

Let $L_1 \leq L_2$ *if* L_2 *is decidable, then* L_1 *is decidable.*

contrpositive: If L_1 *is undecidable, then* L_2 *is undecidable.*





If $x \in L_1$, then $M(x) \in L_2$, so M_2 accepts M(x)

If $x \notin L_1$, then $M(x) \notin L_2$, so M_2 rejects M(x)

***thm* 4.5.** *L_{halt} is undecidable*

Proof. we will prove $L_{flip} \leq L'_{halt}$

Assuming L_{halt} is decidable by a TM M_{halt}, we will prove L_{flip} is decidable, which would be a contradiction.

Create a TM M_{flip} as follows:

run M_{halt} on input (α, α)

1. *If M_{halt} rejects (α, α), let M_{flip} accept α*
2. *If M_{halt} accepts α, α, simulate M_α on input α (using a UTM), and flip the output*

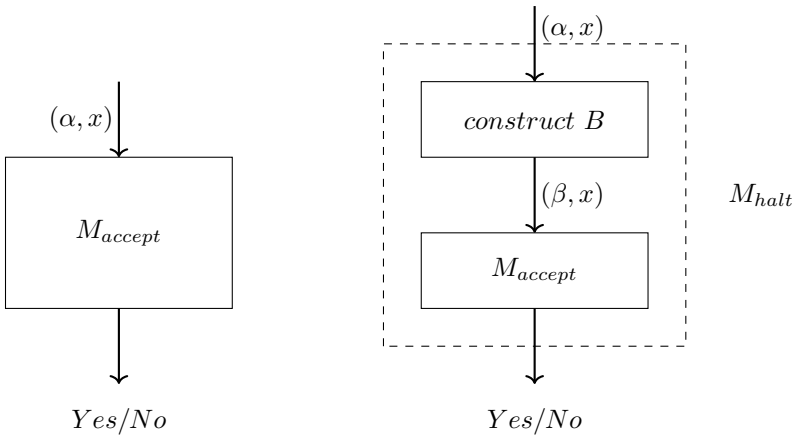
It is easy to verify M_{flip} decides L_{flip}. Contradiction! □

***lemma* 4.6.** *$L_{accept} = \{(\alpha, x)$, M_α accepts $x\}$ is undecidable*

Proof. we will prove $L_{halt} \leq L_{accept}$. Assuming for contradiction that L_{accept} is decidable, i.e. there exists a TM M_{accept} that decides L_{accept}, we construct a TM M_{halt} that decides L_{halt} as follows:

1. *On input (α, x), create a new TM M_β, which simulates M_α on input x, and always accepts whenever M_alpha halts (If M_α loops forever, M_β loops forever as well)*
2. *Run M_{accept} on input (β, x), and forward its output. Clearly, M_{halt} decides L_{halt}. Contradiction!*

□



decides $if (\alpha, x) \in L_{accept}$

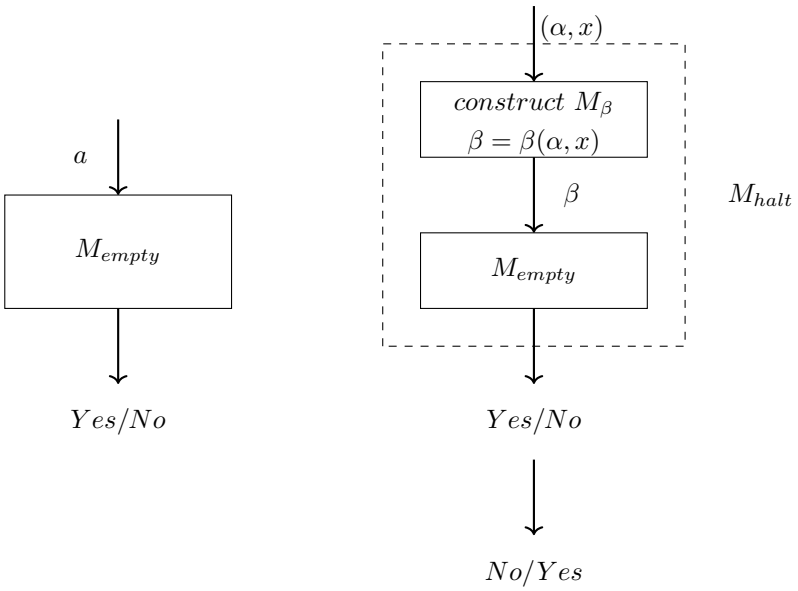
***lemma* 4.7.** *Let* $L_{empty} = \{< M >,$ *M does not accept any input, i.e.* $L(M) = \emptyset\}$

Proof. We will prove $L_{halt} \leq L_{empty}$*. Assuming for contradiction that* L_{empty} *can be decided by a TM* M_{empty}*, we construct a TM* M_{halt} *as follows:*
On input (α, x)

1. *We construct a new TM* M_β*, whose input is* $y \in \{0,1\}^*$*, as follows*
   (a) *simulate* M_alpha *on input* x
   (b) *if step (b) halts, always accept* y

   Clearly, $L(M_\beta) = \emptyset$ *if* M_α *does not halt on* x*. Otherwise,* $L(M_\beta) = \{0,1\}^*$
2. *Run* M_{empty} *on input* β *and flip the output. We can verify that* M_{halt} *decides* L_{halt}*. Contradiction!*

□



***thm* 4.8.** *Let $L_{regular} = \{< M >, M \text{ is a TM, s.t. } L(M) \text{ is a regular language}\}$, it is decidable*

Proof. Assume for contradiction that $L_{regular}$ is decidable, i.e. $\exists$ a TM $M_{regular}$ that decides $L_{regular}$. We will prove L_{accept} is decidable.

On input (α, x), construct a TM as follows:

1. *Construct a TM M_β, where $\beta = \beta(\alpha, x)$, and the input of M_β is denoted by y*
   (a) *If $y \in \{0^n 1^n, n \geq 0\}$, accept*
   (b) *Otherwise, simulate M_α on x, and accept iff M_α accepts x.*
2. *Run $M_{regular}$ on β, and forward its output*
   - *case1 $(\alpha, x) \notin L_{accept}$, i.e., M_{alpha} does not accept x*
     So, $L(M_\beta) = \{0^n 1^n, n \geq 0\}$, which is not a regular language
     Thus, $M_{regular}$ rejects β, which implies that M_{halt} rejects β
   - *case2 $(\alpha, x) \in L_{accept}$. So $L(M_\beta) = \{0, 1\}^*$, which is regular*
     As such, $M_{regular}$ accepts β, and so does M_{accept}

□

***lemma* 4.9.** *Let $L_{equal} = \{(< M_1 >, < M_2 >), M_1, M_2 \text{ are TMs, s.t. } L(M_1) = L(M_2)\}$, it's undecidable*

Proof. *Assuming for contradiction that L_{equal} is decidable, L_{equal} is decided by a TM M_{equal}*

On input $< M >$ construct a TM M_{empty} as follows

1. *Run M_{equal} on input $(< M >, < M_0 >)$, where M_0 rejects immediately ($L(M_0) = \emptyset$)*
2. *Forward the above output*

□