Contents

I Sets, Relations, and Languages

1. 框架

划分问题的难度.

1.1 Automata Theory

讨论问题前需要有清晰的数学定义. Definition of problems and computing models.

finite automata ->pushdown automata -> Turing machines

有个 Church-Turing Thesis, 断言 Turing machines 是最终极的模型.

1.2 Computability Theory

可计算性问题.

1.3 Complexity Theory

证明 hardness, 需要证明问题属于某个 complexity class (复杂类).

2. Problem Definition

2.1 Optimization problem

e.g. Given $G : (V, E, w)$, what is the minimum spanning tree.

2.2 Search problem

e.g. Given G and an integer k, find a spanning tree whose weight is at most k or tells such a tree not exist.

2.3 Decision problem

e.g. Given G and k, is there a spanning tree with weight at most k.

2.4 Counting problem

e.g. Given G and k, what is # (the number of) spanning tree with weight at mots k?

Decision problem 最简单, 因为能解决任意其他三个问题, 就能解决 decision problem. 所有课程着眼于 decision problem, 接下来问题都是 decision problem.

3. 抽象问题为数学形式语言

Decision 结果为 yes-instance or no-instance.
For computer 上述问题等价为: Given a string w, is $w \in$

$$\{\text{encode } (G, k) : (G, k) \text{ is a yes-instance}\}$$

问题由集合唯一决定, 称集合为 a language.

4. Language Definition

Definition I.1 (Alphabet)**.** *A alphabet (字符集) is a* ***finite*** *set of symbols.*

e.g. $\Sigma = \{0, 1\}$, $\Sigma = \{\}$ (空集).

Definition I.2 (String)**.** *A string is a* ***finite*** *sequence of symbols from some alphabet.*

e.g. $\Sigma = \{1, 2, 3\},\ 1, 3, 23, 333$
Length $|w|=$ # symbols in w.
Empty string e with $|e| = 0$. (特殊定义)
$\Sigma^i =$ the set of all string of length i over Σ.
e.g. $\Sigma = \{0, 1\}$, $\Sigma^o = \{e\}$,

$$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$$

$$\Sigma^+ = \bigcup_{i \geq 1} \Sigma^i$$

Definition I.3 (Concatenation)**.** *Given* $u = a_1 \cdots a_n$, $v = b_1 \cdots b_m$, $w = uv = a_1 \cdots a_n b_1 \cdots b_m$

Definition I.4 (exponentiation(幂))**.** $w^i = \underbrace{w \dots w}_{i\ times}$

Definition I.5 (reversal)**.** $w = a_1 \dots a_n$, $w^R = a_n \dots a_1$.

Definition I.6 (Language)**.** *A language over* Σ *is a subset* $L \subseteq \Sigma^*$.

e.g. $\Sigma = \{0, 1\}$, language is ϕ, Σ^*, $\{0^n 1^n : n \geq 0\}$.
decision problem 与 language 一一对应. decision problem $\Leftrightarrow$ language.

Proof. decision problem $\Rightarrow$ { encoding of yes-instance }.
Given a string w, is $w \in L \Leftarrow$ language Q.E.D.

II Finite Automata

有限状态机. state diagram

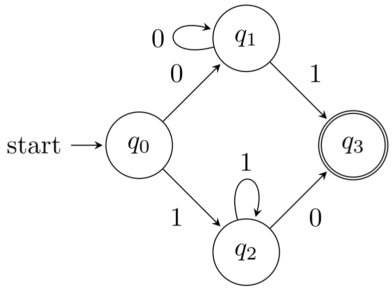


Figure II.1: example state diagram

initial state is unique, Finial state may be several.

1. Definition

Definition II.1. *A finite automata $M = (K, \Sigma, \delta, s, F)$.*

- *Σ: input alphabet*
- *K: a **finite** set of states*
- *$s \in K$: initial state*
- *$F \subseteq K$: the set of finial states*
- *δ: transition function*

$$\delta : K \times \Sigma \to K$$

K current state, Σ symbol, K next state.

之前的 symbol 对之后的结果不会有影响.

Definition II.2 (configuration)**.** *A configuration is any element of $K \times \Sigma^*$ (current state 与 unread input 作用)*

Definition II.3 (yields in one step)**.** *$(q, w) \vdash_M (q', w')$ if $w = aw'$ for some $a \in \Sigma$ and $\delta(q, a) = q'$.*

每走一步, 所携带的输入会减少.

Definition II.4 (yields)**.** *$(q, w) \vdash_M^* (q', w')$ if $(q, w) \vdash_M (q', w')$ or $(q, w) \vdash_M \cdots \vdash_M (q', w')$*

Definition II.5. *M accepts $w \in \Sigma^*$ if $(s, w) \vdash_M^* (q, e)$ for some $q \in F$*

Definition II.6.

$$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$$

M accepts $L(M)$.

M accepts L iif

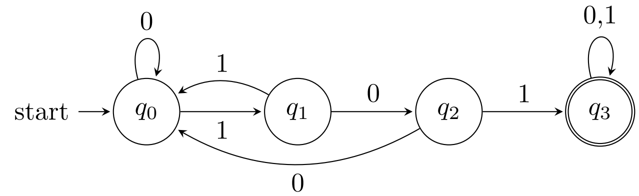
$$\begin{cases} w \in L & M \text{ accepts } w \\ w \notin L & M \text{ doesn't accept } w \end{cases}$$

Definition II.7. *A language is regular if it's accept by some finite automata.*

e.g.

$$\{w \in \{0,1\}^* : w \text{ contains 101 as a substring}\}$$

is regular? yes.



2. Regular Operations

Definition II.8 (Regular Operations)**.**

- *Union $A \cup B = \{w : w \in A \text{ or } w \in B\}$*
- *Concatenation $A \cdot B = \{ab : a \in A \text{ and } b \in B\}$*
- *Star $A^* = \{w_1 w_2 \cdots w_k : w_i \in A \text{ and } k \geq 0\}$*

regular language 在这些操作下是封闭的.

Theorem II.9. *If A and B are regular, so is $A \cup B$.*

Proof. $\exists M_A = (K_A, \Sigma_A, \delta_A, s_A, F_A)$ accepts A, $\exists M_B = (K_B, \Sigma_B, \delta_B, s_B, F_B)$ accepts B.

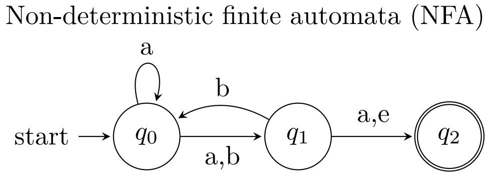
For $M_U = (K_U, \Sigma_U, \delta_U, s_U, F_U)$,

- $\Sigma_U = \Sigma_A \cup \Sigma_B$
- $K_U = K_A \times K_B$
- $s_U = (s_A, s_B)$
- $F_U = \{(q_A, q_B) \in K_A \times K_B : q_A \in F_A \text{ or } q_B \in F_B\}$
- δ_U: $\forall\, q_A \in K_A$, $q_B \in K_B$, $a \in \Sigma_U$, $\delta_U((q_A, q_B), a) = (\delta_A(q_A, a), \delta_B(q_B, a))$

Q.E.D.

3. Non-determinism

deterministic finite automata (DFA) 给 (s, w), 输出唯一.
Non-deterministic finite automata (NFA)



1) several choise for next state
2) e-transition

NFA 是 Δ, 对应的是关系.

$$\Delta = \{(q_0, a, q_1), (q_1, e, q_2), \ldots\}$$

Definition II.10. *A NFA is a 5-tuple* $(K, \Sigma, \Delta, s, F)$*.*
transition relation $\Delta \subseteq K \times \Sigma \cup \{e\} \times K$*.*

NFA's configuration and step is same as DFA.
NFA 路径有很多条, 存在一条读完的路就算接受.

Definition II.11. M *accepts* w *if* $(s, w) \vdash_M^* (q, e)$ *for some* $q \in F$

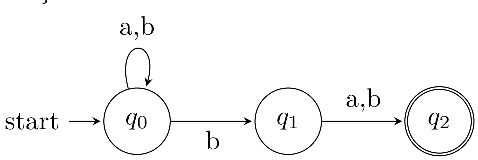
Definition II.12.

$$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$$

M *accepts* $L(M)$.

NFA 类似于 Parallel, 分支就是分裂进程. Magic: Always make the right guess.

e.g. $L = \{w \in \{a, b\}^* :$ the second symbol from the end of w is $b\}$



Theorem II.13.

$$\forall \textit{ DFA } M \Rightarrow \exists \textit{ NFA } M' \textit{ s.t. } L(M) = L(M')$$
$$\forall \textit{ NFA } M \Rightarrow \exists \textit{ DFA } M' \textit{ s.t. } L(M) = L(M')$$

第一条显然. Idea: 构造 DFA, 可以模拟 NFA 分支计算. 第二条:

Proof.

$$\forall \text{ NFA } M = (K, \Sigma, \Delta, s, F),$$
$$\exists \text{ DFA } M' = (K', \Sigma', \delta', s', F')$$

- $\Sigma' = \Sigma$
- $K' = 2^K = \{Q : Q \subseteq K\}$
- $F' = \{Q \subseteq K : Q \cap F \neq \emptyset\}$
- $s' = E(s)$,
  $\forall q \in K, E(q) = \{p \in K : (q, e) \vdash_M^* (p, e)\}$
- $\forall Q \subseteq K$, $a \in \Sigma^*$,

$$\delta'(Q, a) = \bigcup_{q \in Q} \bigcup_{p:(q,a,p) \in \Delta} E(p)$$

Q.E.D.

$E(q)$ 相当于在 q 零步之内 (不耗费输入的字符) 能到达的结点.

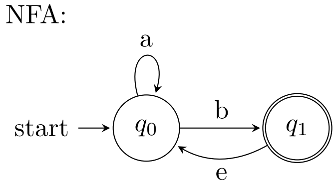
Theorem II.14. *DFA* M' *accepts* $w \iff$ *NFA* M *accepts* w*.*

Claim II.14.1. $\forall$ $p, q \in K$ *and* $w \in \Sigma^*$, $(p, w) \vdash_M^* (q, e) \iff (E(p), w) \vdash_{M'}^* (Q, e)$ *for some* $Q \ni q$

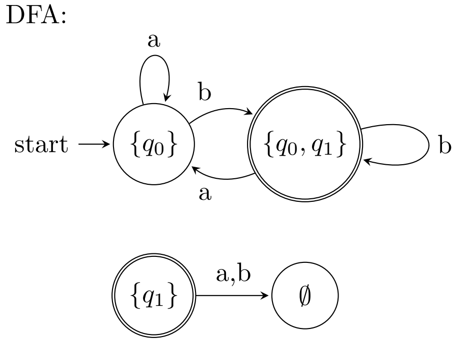
Proof. by induction on $|w|$. Q.E.D.

Proof. NFA M accepts
$\iff (s, w) \vdash_M^* (q, e)$ with $q \in F$.
$\iff (E(s), w) \vdash_{M'}^* (Q, e)$ with $Q \ni q (Q \in F')$
$\iff$ DFA M' accepts w. Q.E.D.

e.g.
NFA:



DFA:



$\{q_1\}$ 与 $\emptyset$ 可省略.

Definition II.15. *A language is regular if it's accept by some non-deterministic finite automata.*

Theorem II.16. *If* A *and* B *are regular, so is* $A \cdot B$

Proof. $\exists M_A = (K_A, \Sigma_A, \Delta_A, s_A, F_A)$ accepts A, $\exists M_B = (K_B, \Sigma_B, \Delta_B, s_B, F_B)$ accepts B.

For $M^\circ = (K^\circ, \Sigma^\circ, \Delta^\circ, s^\circ, F^\circ)$,

- $\Sigma^\circ = \Sigma_A \cup \Sigma_B$
- $K^\circ = K_A \cup K_B$
- $s^\circ = s_A$
- $F^\circ = F_B$
- $\Delta^\circ = \Delta_A \cup \Delta_B \cup \{(q, e, s_B) : q \in F_A\}$

Q.E.D.

Theorem II.17. *If A is regular, so is A^**

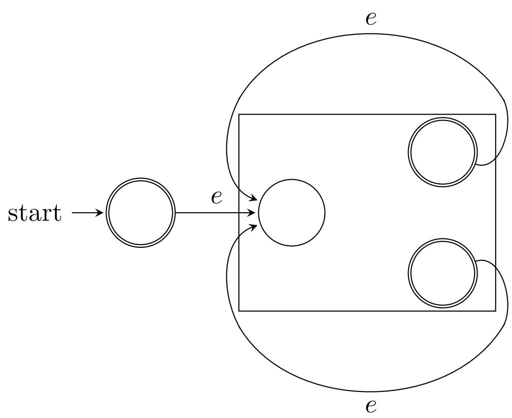


Figure II.2: A^*

正则对 $\cup$, $\cap$, $^{-}$, $\circ$, $*$, $-$ 都封闭.

III Regular Expression

e.g. $R = (a \cup b^*a)$, $L(R) = (\{a\} \cup \{b\})^* \circ \{a\}$.

Definition III.1 (Regular Expression)**.** *Atomic:*

- ϕ*:* $L(\phi) = \phi$
- $a \in \Sigma$*:* $L(a) = \{a\}$

Composite $\cup, \circ, *$*.*

- $R_1 \cup R_2$*:* $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
- R_1R_2*:* $L(R_1R_2) = L(R_1) \circ L(R_2)$
- R^**:* $L(R^*) = (L(R))^*$

Precedence: $* > \circ > \cup$*.*

e.g.

- $a^*b \cup b^*a = ((a^*)b) \cup ((b^*)a)$
- $\{e\} = \phi^*$
- $\{w \in \{a \cup b\}^* : w$ starts with a and end with $b\} = a(a \cup b)^*b$
- $\{w \in \{a \cup b\}^* : w$ has at least two occurrence of $a\} = (a \cup b)^*a(a \cup b)^*a(a \cup b)^*$

Theorem III.2. *A language B is regular iff there is some regular exp R with $L(R) = B$.*

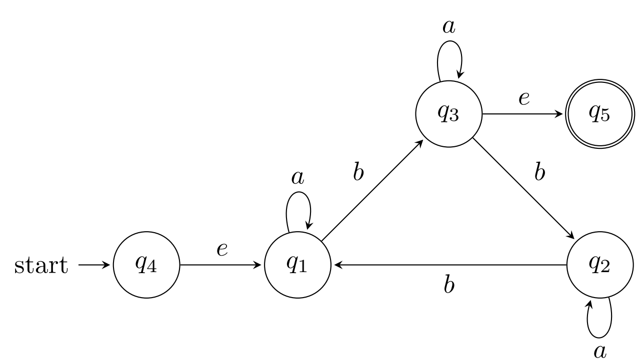
Idea: Regular Expression $\iff$ NFA ($L(R) = L(M)$). $R \Longrightarrow M$ 直接转换. $R \Longleftarrow M$ 需要一个算法:

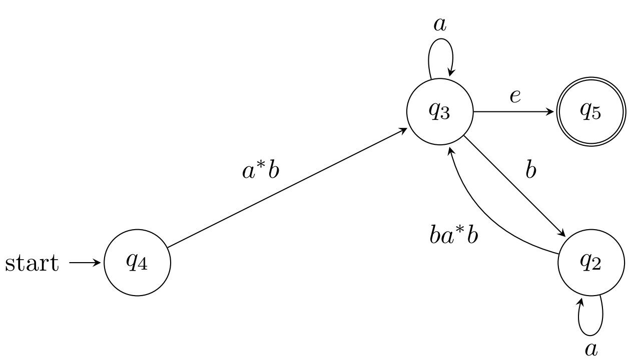
1) simplity M so that

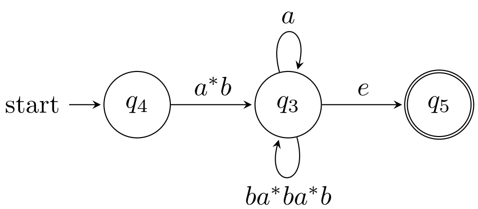
   a. no arc enters the initial state. (创建一个新的 initial state)

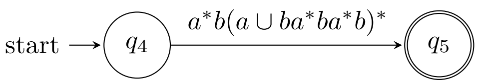
   b. only one finial state and no atc leaves it. (新建 finial state)

2) eliminate states: 仅处理长度为 2 以内的情景, 递归处理更长的情景.









Proof. $R \Longleftarrow M$

NFA $M = (K, \Sigma, \delta, s, F)$.

1) $K = \{q_1, \ldots, q_n\}, s = q_{n-1}, F = \{q_n\}$.
2) $\forall q \& a, (p, a, q_{n-1}) \notin \Delta$
3) $\forall q \& a, (q_n, a, p) \notin \Delta$

R s.t. $L(R) = L(M)$.

Subproblem: for $i, j \in [1, n]$, for $k \in [1, n]$, define R_{ij}^k,

$$L(R_{ij}^k) = \{w \in \Sigma^* : w \text{ drives } M \text{ for } q_i \text{ to } q_j \\ \text{with no intermediate state having index} > k\}$$

Goal: $R_{(n-1)n}^{n-2}$
Base case: $(k = 0)$, $a_i = (q_i, a, q_i) \in \Delta$
for $i, j \in [1, n]$ **do**
  if $i = j$ **then**
    $R_{ij}^0 = \phi^* \cup a_1 \cup a_2 \cup \cdots \cup a_m$
  end if
  if $i \neq j$ **then**
    $R_{ij}^0 = a_1 \cup a_2 \cup \cdots \cup a_m$
  end if
end for
Recurrence: $R_{ij}^k = R_{ij}^{k-1} \cup R_{1k}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}$

Q.E.D.

Theorem III.3 (Pumping theorem)**.** *Let L be a regular language. There exists an integer $p \geq 1$ (pumping length), such that $w \in L$ with $|w| \geq p$ can be divided into 3 parts $w = xyz$, satisfying:*

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

Proof. If L is finite,

$$p = \max_{w \in L} |w| + 1$$

Assume L is infinite, $\exists$ DFA M accepting L, $p :=$ # state of M.
Let w be any string in L with $|w| \geq p$, $w = a_1 \ldots a_n$, $\exists 0 \leq i < j \leq p, q_i = q_j$ and

$$x := a_1 \ldots a_i$$
$$y := a_{i+1} \ldots a_j$$
$$z := a_{j+1} \ldots a_n$$

1) $\forall k \geq 0, xy^kz \in L$
2) $|y| = j - i > 0$
3) $|xy| = j \leq p$

Q.E.D.

e.g. $L = \{0^n 1^n : n \geq 0\}$ is not regular.

Proof. Assume it is regular. Let p be the pumping length given by the pumping theorem. $\forall w \in L$ with $|w| \geq p$,

$$w = 0^p 1^p$$

w can be written $w = xyz$ s.t.

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

(2)(3)
$\Rightarrow y = 0^k$ for some $k \geq 1$
$\Rightarrow xy^0z = 0^{p-k}1^p \notin L$
$\nRightarrow$ (1) Q.E.D.

e.g.
$L = \{w \in \{0,1\}^* : w$ has equal number of 0's and 1's$\}$ is not regular.

Assume regular $\Rightarrow L \cap 0^*1^* = \{0^n 1^n : n \geq 0\}$ is regular. contradictory.

IV Context-free Language

Context-free Grammar
e.g.

- start symbol: S
- rule: $S \to aSb$, $S \to A$, $A \to c$, $A \to e$
  - non-terminal: S, A
  - terminal: a, b, c

1. Definition

Definition IV.1. *A content-free grammar $G = (V, \Sigma, S, R)$,*

- *V a **finite** set of symbols*
- *$\Sigma \subseteq V$: the set of terminals*
  $V - \Sigma$: the set of non-terminals
- *$S \in V - \Sigma$: start symbol*
- *$R \subseteq (V - \Sigma) \times V^*$: a **finite** set of rules*

Definition IV.2 (derive in one step)**.** *$\forall x, y, u \in V^*$, $\forall A \in V - \Sigma$,*

$$xAy \Rightarrow_G xuy \text{ if } (A, u) \in R$$

Definition IV.3 (a derivation from w to u)**.** *$\forall w, u \in V^*$,*

$$w \Rightarrow_G^* u \text{ if } w = u \text{ or } w \Rightarrow_G \cdots \Rightarrow_G u$$

Definition IV.4. *G generates $w \in \Sigma^*$ if $S \Rightarrow_G^* w$.*

$$L(G) = \{w \in \Sigma^* : G \text{ generates } w\}$$

G generates $L(G)$, called content-free language

e.g. Show that $\{w \in \{a, b\}^* : w = w^R\}$ (回文串) is content-free.
$S \to e|a|b|aSa|bSb$.
need to proof:

- if $w \in L(G)$, $w = w^R$. (归纳 w 替换的次数)
- if $w = w^R$, $w \in L(G)$. (归纳 $|w|$)

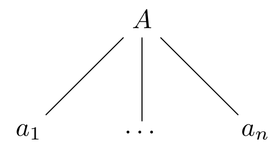
e.g. have $S \to SS$, $S \to (S)$, $S \to e$, generates ()().

- $S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()()$ (leftmost derivation)
- $S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ()()$(rightmost derivation)

parse tree is same.

Definition IV.5 (parse tree)**.**

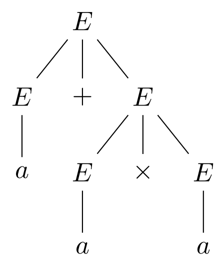
- *internal node: non-terminal*
- *leaves: terminal or e, e must be the only child of its parent*
- *edge: $A \to a_1, \ldots, a_n$*

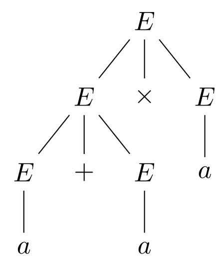


- *the yield of parse tree: e.g. a_1, a_n*

e.g.

- $E \to E + E$,
- $E \to E \times E$,
- $E \to a$

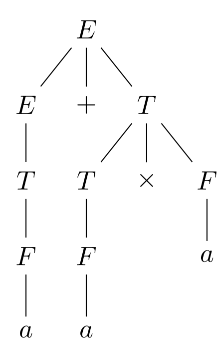




ambiguous
but

- $E \to E + T$,
- $E \to T$,
- $T \to T \times F$,
- $T \to F$,
- $F \to (E)$,
- $F \to a$

no ambiguous



Fact: $\{a^i b^j c^k : i = j \text{ or } j = k\}$ inherently language (任意生成此的 language 都有歧义)

2. Chomsky nrom form (CNF)

Definition IV.6. *A CFG is in Chomsky nrom form (CNF) if every rule is of one of the following forms:*

1) $S \to e$
2) $A \to BC$, $B, C \in V - \Sigma - \{S\}$
3) $A \to a$, $a \in \Sigma$

Observation: Suppose that G is a CFG in CNF. If G generates a string of length $n \geq 1$, # length if derivation is $2n-1$.

Theorem IV.7. *Every CFG has an equivalent CFG in CNF*

sketch of proof: 不满足的几种情况

1) S appears on the right side of a rule
   solution: create new start symbol S_0, let $S_0 \to S$
2) $A \to e$ for $A \neq S$
   solution: e.g. $A \to e$, $B \to ACA$, delete $A \to e$, add $B \to AC$, $B \to CA$, $B \to C$. 所有类似的规则作类似的处理.
3) $A \to B$, for some $B \in V - \Sigma$
   solution: If $A \neq S$, the same as $A \to e$. If $A = S$: e.g. $S \to B$, $B \to AC$, $B \to CD$, delete all, add $S \to AC$, $S \to CD$
4) $A \to u_1, \ldots, u_k$, $k \geq 3, u_i \in V$
   solution: delete it, add $A \to u_1 V_2$, $V_2 \to u_2, \ldots, u_k$ and so on.
5) $A \to u_1 u_2$, at least one $u_i \in \Sigma$
   solution: e.g. $A \to bC$, delete it, add $A \to BC$, $B \to b$

3. Pushdown Automata(PDA)

PDA $\Longleftrightarrow$ CFG. PDA = NFA + stack
have input tape, state control, stack

Definition IV.8 (Pushdown Automata)**.** *A PDA is a 6-tuple* $P = (K, \Gamma, \Sigma, \Delta, s, F)$

- Γ*: stack alphabet*
- Δ*: transition relation: a* ***finite*** *subset of*

$$(K \times (\Sigma \cup \{e\}) \times \Gamma^*) \times (K \times \Gamma^*)$$

  - *1st* K*: current state*
  - $\Sigma \cup \{e\}$*: current symbol that is read*
  - *1st* Γ^**: pop string at the top of stack*
  - *2nd* K*: next state*
  - *2nd* Γ^**: push a string onto the stack*

e.g. $((p, a, 123), (q, 45))$, 在状态 p 时, 若读到 a, 且 stack 顶有 123, 将 123 pop, 然后压入 45, 状态 $p \to q$. $((p, a, e), (q, B))$, 这个 stack 一直满足条件, 顶一定有 e.

Definition IV.9. *A configuration of* P *is a member of* $K \times \Sigma^* \times \Gamma^*$

- *current state*
- *unread input*
- *stack control*

Definition IV.10. $(p, x, \alpha) \vdash_P (q, y, \beta)$ *if* $\exists ((p, a, r), (q, \eta)) \in \Delta$ *s.t.*

$$x = ay$$
$$\alpha = r\tau \text{ and } \beta = \eta\tau$$

for some $\tau \in \Gamma^*$

Definition IV.11. $(p, x, \alpha) \vdash_P^* (q, y, \beta)$ *if* $(p, x, \alpha) = (q, y, \beta)$ *or* $(p, x, \alpha) \vdash_P \cdots \vdash_P (q, y, \beta)$

Definition IV.12. P *accepts* $w \in \Sigma^*$ *if*

$$(s, w, e) \vdash_P^* (q, e, e)$$

for some $q \in \Gamma$

1) *finial state*
2) *consume all the input*
3) *empty stack*

Definition IV.13. $L(P) = \{w \in \Sigma^* : P \text{ accepts } w\}$. *P accepts* $L(P)$

e.g. $\{w \in \{0,1\}^* : \# \text{ 0's} = \# \text{ 1's in } w\}$ 设计 PDA, accepts it.

- $K = \{s\}$
- $F = \{s\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1\}$

$$\begin{aligned}\Delta = \{&((s,0,1),(s,e))\\ &((s,0,e),(s,0))\\ &((s,1,0),(s,e))\\ &((s,1,e),(s,1))\}\end{aligned}$$

PDA 是非确定的, 也有猜的能力.
CFG $\Longleftrightarrow$ PDA
CFG $\to$ PDA
Idea:

1) in stack, non-deterministic generates a string from S
2) compute it to the input
3) accepts if they match

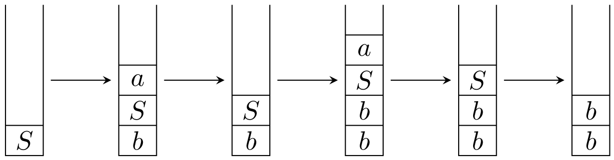


Figure IV.1: Idea of CFG → PDA. e.g. $S \to aSb|e$, aabb

Proof. CFG → PDA

$G = (V, \Sigma, S, R) \to M = (K, \Sigma, \Gamma, \Delta, s, F) \iff L(M) = L(G)$

- $\Gamma = V$
- $K = \{s, f\}$
- $F = \{f\}$
-

$$\begin{aligned}\Delta = \{&((s,e,e),(f,S)),\\ &((f,e,A),(f,u)) \text{ for each } (A,u) \in R,\\ &((f,a,a),(f,e)) \text{ for each } a \in \Sigma\}\end{aligned}$$

Q.E.D.

PDA → simple PDA → CFG

Definition IV.14. *A PDA $M = (K, \Sigma, \Gamma, \Delta, s, F)$ is simple if*

1) $|F| = 1$
2) for each transition $((p,a,\alpha),(q,\beta)) \in \Delta$
either $\alpha = e$ and $|\beta| = 1$
or $|\alpha| = 1$ and $\beta = e$

PDA → simple PDA

1) If $|F| > 1$, create a new state f', for each $q \in F$, add a new transition $((q,e,e),(f',e))$, $F := \{f'\}$.
2) only four cases, as $((p,a,\alpha),(q,\beta))$

a. $|\alpha| \geq 1$ and $|\beta| \geq 1$

create a new state r, replace the transition with

$$((p,a,\alpha),(r,e))$$
$$((r,e,e),(q,\beta))$$

b. $|\alpha| > 1$ and $\beta = e$

$\alpha = c_1, c_2, \ldots, c_k$, create new state $r_1, \ldots, r_{k-1}$. replace the transition with

$$((p,a,c_1),(r_1,e))$$
$$((r_1,e,c_2),(r_2,e))$$
$$\vdots$$
$$((r_{k-1},e,c_k),(q,e))$$

c. $\alpha = e$ and $|\beta| > 1$

same as 2

d. $\alpha = e$ and $\beta = e$

create a new state r, pick a b from Γ,

$$((p,a,e),(r,b))$$
$$((r,e,b),(q,e))$$

simple PDA → CFG

a simple PDA $P = (K, \Sigma, \Gamma, \Delta, s, F) \to G = (V, \Sigma, S, R)$

- non-terminal: $\{A_{pq}: \text{ for } p, q \in K\}$
  Goal:
  $$\begin{aligned}&A_{pq} \to w \in \Sigma^*\\ \iff & w \in \{w \in \Sigma^* : (p,u,e) \vdash_p^* (q,e,e)\}\end{aligned}$$
- $S = A_{sf}$, $F = \{f\}$(simple)
- $R =$

  1) $\forall p \in K$, $A_{pp} \to e$
  2) $\forall p, q \in K$,
     $A_{pq} \to A_{pr}A_{rq}$ $\forall r \in K$
     $A_{pq} \to aA_{p'q'}b$
     $\forall((p,a,e),(p',\alpha)) \in \Delta$ and $((q',b,\alpha),(q,e)) \in \Delta$ for some $\alpha \in \Gamma$

proof idea:

- ⇒ by induction on length of derivation from A_{pq} to w
- ⇐ by induction on # steps of computation.

PDA → define CFG

Theorem IV.15. *Every regular language is content-free.*

4. CFL properties

CFG closure properties: $\cup, \circ, *$ is closure, $\cap, \bar{A}$ isn't closure.

$G_A = (V_A, \Sigma_A, S_A, R_A)$, $G_B = (V_B, \Sigma_B, S_B, R_B)$

- $G_{A\cup B}$: $S \to S_A|S_B$

- $G_{A \circ B}$: $S \to S_A S_B$
- G_{A^*}: $S \to e|SS_A$

$A = \{a^i b^j c^k : i = j\}, B = \{a^i b^j c^k : j = k\} \to A$ and B are content-free.

- $A \cap B = \{a^n b^n c^n : n \geq 0\}$ not content-free.
- $A \cap B = \overline{\bar{A} \cup \bar{B}}$ not content-free.

so CFL is not closed under $\cap$ or $\bar{A}$

Theorem IV.16 (Pumping theorem for CFL)**.** *Let L be a content-free language. There exists an integer $p > 0$ such that any $w \in L$ with $|w| \geq p$ can be divided into $w = uvxyz$ satifiying:*

1) $uv^i xy^i z \in L$ *for any* $i \geq 0$
2) $|v| + |y| > 0$
3) $|vxy| \leq p$

e.g. $\{a^n b^n : n \geq 0\}, p = 2$, $u = x = z = e, v = a, y = b$

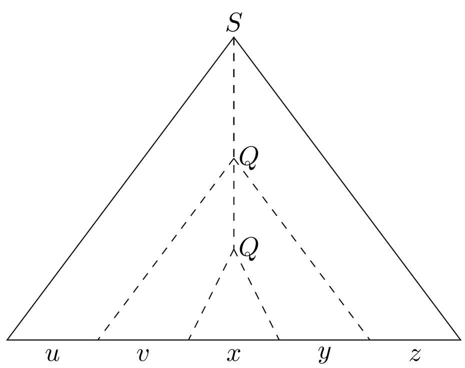


Figure IV.2: Idea's parse tree

non-terminal $|V - \Sigma|$ finite. Q exists twice in a long enough path, draw Q subtree.

- $S \Rightarrow^* uQz$
- $Q \Rightarrow^* vQy$
- $Q \Rightarrow^* x$

$S \Rightarrow^* uv^i xy^i z \in L$, $\forall i \geq 0$.

Proof. L is content-free $\Rightarrow \exists G = (V, \Sigma, S, R)$ with $L(G) = L$.

Let $b = \max\{|u| : (A, u) \in R\}$, fanout $\leq b$

Fact: If a tree with famout $\leq b$ has n leaves, its height(# edges on the longest descending path) $\geq \log_b n$.

$p = b^{|V - \Sigma| + 1}$, $w \in L$ with $|w| \geq p$. Let T be the parse tree of w, (with smallest # nodes), height of $T \geq \log_b p = |V - \Sigma| + 1$.

- # edges $\geq |V - \Sigma| + 1$,
- # nodes $\geq |V - \Sigma| + 2$,
- # non-terminal on the path $\geq |V - \Sigma| + 1$

some non-terminal Q appears twice on the path, as shwon in **Figure** IV.2.

1) $uv^i xy^i z \in L$ for any $i \geq 0$ 显然
2) $|u| + |y| > 0$

If $v = y = e, w = uxz$, parse tree smaller than T, but T is the smallest, contradiction.

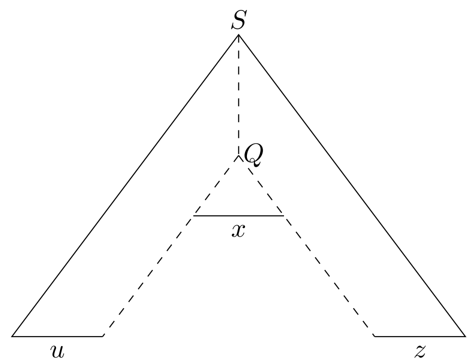


Figure IV.3: 2)'s parse tree

3) $|vxy| \leq p$

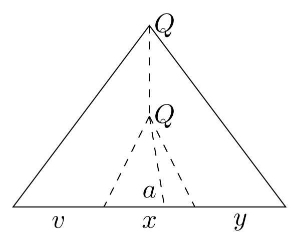


Figure IV.4: 3)'s parse tree

The height of subtree $\leq |V - \Sigma| + 1$, so $|vxy| = \#$ leaves $\leq b^{|V - \Sigma| + 1} = p$.
If choose Q, # nodes of $QQa \leq 2 + |V - \Sigma|$.
If choose the lowest twice non-terminal, the length $\leq 1 + |V - \Sigma|$.

Q.E.D.

e.g. $\{a^n b^n c^n : n \geq 0\}$ is not content-free

Proof. Assume it's content-free. Let p be the pumping length. Pick $a^p b^p c^p \in L$. By pumping theorem, $a^p b^p c^p = uvxyz$, s.t.

1) $uv^i xy^i z \in L$ for any $i \geq 0$
2) $|v| + |y| > 0$
3) $|vxy| \leq p$

(3) $\Rightarrow$ at least one of a and c doesn't appear in v or y $\Rightarrow$ $uv^0xy^0z \notin L$, contradiction. Q.E.D.

V Turing Machine

1. Definition

1) $\leftarrow, \rightarrow$
2) raed and write

- left end symbol $\triangleright$
- blank symbol $\sqcup$

Definition V.1. *A Turing machine is a 5-tuple* $M = (K, \Sigma, \delta, s, H)$

- K*: a finite set of states*
- Σ*: tape alphabet (left end symbol and blank symbol)*
- $s \in K$*: initial state*
- $H \subseteq K$*: a set of halting state*
- δ*:*

$$\underbrace{(K-H)}_{\text{current state}} \times \underbrace{\Sigma}_{\text{symbol read by the head}} \rightarrow \underbrace{K}_{\text{next state}} \times \underbrace{(\overbrace{\{\leftarrow, \rightarrow\}}^{\text{moving}} \cup \overbrace{(\Sigma - \{\triangleright\})}^{\text{writing}})}_{\text{head action}}$$

s.t. $\forall q \in K - H$, $\delta(q, \triangleright) = (p, \rightarrow)$ *for some* $p \in K$

Definition V.2 (configuration)**.** *A configuration is a member of*

$$K \times \triangleright(\Sigma - \{\triangleright\})^* \times (\{e\} \cup (\Sigma - \{\triangleright\})^*(\Sigma - \{\triangleright, \sqcup\}))$$

Definition V.3 (yield one step)**.**
$(q_1, \triangleright w_1 \underline{a_1} u_1) \vdash_M (q_2, \triangleright w_2 \underline{a_2} u_2)$ *if*

1) writing: $\delta(q_1, a_1) = (q_2, a_2)$, $w_1 = w_2$ *and* $u_1 = u_2$
2) moving left: $\delta(q_1, a) = (q_2, \leftarrow)$, $w_1 = w_2 a_2$, *and* $u_2 = a_1 u_1$ *(*$u_2 = e$ *if* $a_1 = \sqcup, u_1 = e$*)*
3) moving right: $\delta(q_1, a) = (q_2, \rightarrow)$, $w_2 = w_1 a_1$, *and* $u_1 = a_2 u_2$

Definition V.4 (yields)**.**
$(q_1, \triangleright w_1 \underline{a_1} u_1) \vdash_M^* (q_2, \triangleright w_2 \underline{a_2} u_2)$ *if*

1) $(q_1, \triangleright w_1 \underline{a_1} u_1) \vdash_M (q_2, \triangleright w_2 \underline{a_2} u_2)$
2) $(q_1, \triangleright w_1 \underline{a_1} u_1) \vdash_M \cdots \vdash_M (q_2, \triangleright w_2 \underline{a_2} u_2)$

- $(q, \triangleright w \underline{a} u)$ is a halting configuration if $q \in H$.
- initial configuration

Fix Σ

1) symbol writing machine M_a, $(a \in \Sigma - \{\triangleright\})$.
$M_a = (\{s, h\}, \Sigma, \delta, s, \{h\})$
   - $\forall b \in \Sigma - \{\triangleright\}$, $\delta(s, b) = (h, a)$
   - $\delta(s, \triangleright) = \delta(s, \rightarrow)$

2) heading moving machine $M_{\rightarrow}$, $M_{\leftarrow}$
$M_{\leftarrow} = (\{s,h\}, \Sigma, \delta, s, \{h\})$

- $\forall b \in \Sigma - \{\triangleright\}$, $\delta(s,b) = (h, \leftarrow)$
- $\delta(s,\triangleright) = \delta(s,\rightarrow)$

basic machine: $M_a, M_{\leftarrow}, M_{\rightarrow}$, also called a, L, R.

$$\begin{tikzcd} \text{start} \arrow[r] & M_1 \arrow[r, "0"] \arrow[d, "1"] & M_2 \\ & M_3 & \end{tikzcd}$$

Figure V.1: 组合图灵机

Algorithm V.1 组合图灵机

run M_1 until it halts
if the symbol read by head is 0 **then**
  run M_2
else if the symbol read by head is 1 **then**
  run M_3
else
  halt
end if

$$\text{start} \longrightarrow R \xrightarrow{\quad\Sigma\quad} R$$

(a) R^2

$$\text{start} \longrightarrow R \xrightarrow{\quad a \neq \sqcup\quad} Ra$$

(b)

$$\text{start} \longrightarrow R \circlearrowleft \overline{\sqcup}$$

(c) $R_{\sqcup}$

$$\text{start} \longrightarrow L \circlearrowleft \sqcup$$

(d) $L_{\overline{\sqcup}}$

$$\text{start} \longrightarrow R \circlearrowleft \sqcup$$

(e) $R_{\overline{\sqcup}}$

$$\text{start} \longrightarrow L \circlearrowleft \overline{\sqcup}$$

(f) $L_{\sqcup}$

Figure V.2: Example

Left shifting machine: $S_{\leftarrow}$. $\forall w \in (\Sigma - \{\triangleright, \sqcup\})^*$,

$$\triangleright \sqcup \sqcup w \underline{\sqcup} \rightarrow \triangleright \sqcup w \underline{\sqcup}$$

$$\begin{tikzcd} \text{start} \arrow[r] & L_{\sqcup} \arrow[r] & R \arrow[r, "a \neq \sqcup"] \arrow[d, "\sqcup"] & \sqcup L a R \arrow[l, bend right] \\ & & L & \end{tikzcd}$$

Figure V.3: Left shifting machine: $S_{\leftarrow}$.

1.1 Recognize Language

- input alphabet: $\Sigma_0 \subseteq (\Sigma - \{\triangleright, \sqcup\})$
- initial configuration $\{s, \triangleright\underline{\sqcup}w\}$

Definition V.5. *M semidecides $L(M)$, called recursively enumerable (递归可枚举)/recognizable*

$$L(M) = \{w \in \Sigma_0^* : (s, \triangleright\underline{\sqcup}w) \vdash^* (h, \dots) \text{ for some } h \in H\}$$

实际上没法用, 因为不保证可接受时间内停机.

Definition V.6. *Let $M = (K, \Sigma_0, \Sigma, \delta, s, \{y, n\})$ be a Turing Machine. We say M decides a language $L \subseteq \Sigma^*$ if*

1) $\forall w \in L$

$$(s, \triangleright\underline{\sqcup}w) \vdash^* (y, \dots)$$

M accepts w

2) $\forall w \in \Sigma^* - L$

$$(s, \triangleright\underline{\sqcup}w) \vdash^* (n, \dots)$$

M rejects w

A language is recursive/decidable if it is decided by some Turing Machine.

Theorem V.7. *If L is recursive, L must is recursively enumerable.*

Idea: 构造 n 不停机, y 停机.

2. Compute Function

Definition V.8. $\forall w \in \Sigma_0^*$, *if* $(s,) \vdash^* (h, \triangleright\underline{\sqcup}y)$ *for some* $h \in H$ *and some* $y \in \Sigma_0^*$.

Figure V.4: output

y: output of M on w. $y = M(w)$.
Let $f : \Sigma_0^* \rightarrow \Sigma_0^*$. We say M computes f if

$$\forall w \in \Sigma_0^*, M(w) = f(w)$$

f is called recursive/computable

e.g. $\{a^n b^n c^n : n \geq 0\}$ is recursive.

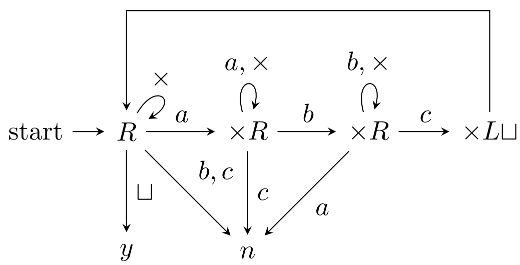


Figure V.5: $\{a^n b^n c^n : n \geq 0\}$

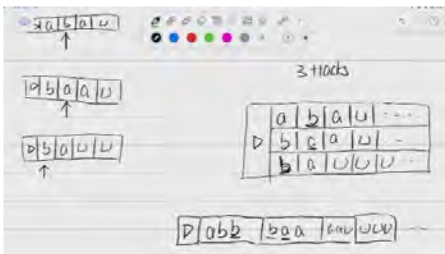


Figure V.6: Multiple tape TM

3. Extensions of Turing machines

3.1 Multiple tape TM

$$\delta : (K - H) \times \Sigma^k \to K \times (\Sigma \cup \{\leftarrow, \rightarrow\})^k$$

Idea: 以三带为例, 将一条纸带分为三个 tracks, 也可以等价于把纸带拉平, 其 symbol 长度为 3.

3.2 Two-way Infinite Tape

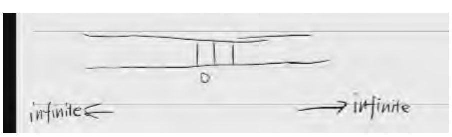


Figure V.7: Two-way Infinite Tape

向两侧无限延申.
可以先使用双带模拟, 然后用标准 TM 模拟双带.

3.3 Multiple Heads

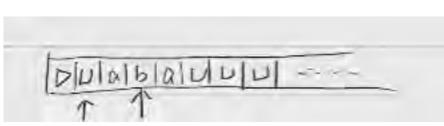


Figure V.8: Multiple Heads

一个纸带但多个读写头.

用下划线标记这 k 个读写头, 先找到 k 个下划线位置, 然后做操作.

3.4 Two-Dimensional Tape

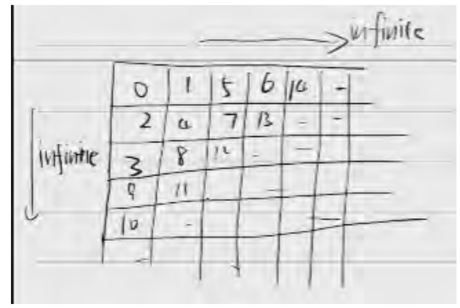


Figure V.9: Two-Dimensional Tape

二维纸带.
将所有格子做编号, 然后拉平.

3.5 Random Access TM

head 可以跳格子.
把跳拆开成单步走即可.

4. Non-deterministic TM

Definition V.9. *A non-deterministic Turing Machine is 5-tuple* $M = (K, \Sigma, \Delta, s, H)$

- K
- Σ
- s
- H
- Δ*: a finite set of*

$$((K - H) \times \Sigma) \times (K \times \Sigma \cup \{\leftarrow, \rightarrow\})$$

可以看作是一颗树.
Also can define configuration and $\vdash_M, \vdash_M^*, \vdash_M^N$ (走 N 步).

Definition V.10. *A NTM M with input alphabet Σ_0, semidecides $L \subseteq \Sigma_0^*$, if $\forall w \in \Sigma_0^*$, $w \in L$ iff $(S, \triangleright \sqcup w) \vdash_M^* (h, \dots)$ for some h.*

Definition V.11. *Let $M = (K, \Sigma, \Delta, s, \{y, n\})$ be a NTM with input alphabet Σ_0. M decides a language $L \in \Sigma_0^*$ if*

1) there is a constant N (independent of the M), $\forall w \in \Sigma_0^*$, there is no configuration C, satisfy $(s, \triangleright \underline{\sqcup} w) \vdash_M^N C$. (要求每条分支在 N 步内停止)
2) $w \in L$ iff $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (y, \dots)$

即 NTM 树高有限, 且

- if $w \in L$, some branch accepts w
- if $w \notin L$, every branch rejects w

e.g. Let $C = \{100, 110, 1000, \dots\}$ (所有合数二进制编码集合)

Idea: 猜其两个分解 p, q, if $p \times q = w$ accepts it, else rejects it.

Theorem V.12. *Every NTM can be simulated by a DTM.*

Proof. (sketch) 以 semidecides 为例.

a NTM semidecides $L \rightarrow$ a DTM semidecides L.

3-tape DTM to simulate NTM (using BFS)

- store the input w
- simulate NTM
- enumerate hints 类似程序栈, 使用 bfs

Q.E.D.

VI Undecidability

1. Church-Turing Thesis

Church-Turing Thesis

Intuition of algorithm	equals	Turing machines that halt on every input
solve		decide
decision porblem	equals	language

Intuition of algorithm $\Longleftrightarrow$ TM that halt of every input

Description of TM

1) formal definition: $M = (K, \Sigma, \delta, s, H)$
2) Implement-level desc: diagram
3) high-level: "pseudo code"

Fact:

1) Any finite set can encoded.
2) Any finite collection of finite sets can be encoded

Object $O \rightarrow$ "O" (用双引号表示 encode)

2. Decidable Problem

R_1 $A_{DFA} = \{$"D" "w" $: D$ is a DFA that accepts $w\}$.

M_{R_1} = on input "D" "w".

Algorithm VI.1 M_{R_1}

```
run D on w.
if D accepts w then
    accepts "D" "w"
else
    rejects "D" "w"
end if
```

R_2 $A_{NFA} = \{$"N" "w" $: N$ is a NFA that accepts $w\}$

M_{R_2} = on input "N" "w".

Algorithm VI.2 M_{R_2}

```
N → equivalent DFA D
run M_R1 on "D" "w"
output the result of M_R1
```

a reduction(归约) from A_{NFA} to A_{DFA}. 将左输入映射为右输入, 保证映射前后答案一致.

R_3 $A_{REX} = \{$"R" "w" $: R$ is a regular expression with $w \in L(R)\}$

M_{R_3} = on input "R" "w".

Algorithm VI.3 M_{R_3}

R $\to$ an equivalent NFA N
run M_{R_2} on "N" "w"
output the result of M_{R_2}

R_4 $E_{DFA} = \{\text{"}D\text{"} : D \text{ is a DFA and } L(D) = \emptyset\}$
M_{R_4} = on input "D"

Algorithm VI.4 M_{R_4}

if D has no finial state **then**
  accepts
else
  run DFS on BFS starting with the initial state on the diagram
  if $\exists$ path from s to finial **then**
    reject
  else
    accept
  end if
end if

R_5 $EQ_{DFA} = \{\text{"}D_1\text{" "}D_2\text{"} : D_1 \text{ and } D_2 \text{ are DFAs with } L(D_1) = L(D_2)\}$

Definition VI.1 (symmetric difference)**.**

$$A \oplus B = \{x \in A \cup B \cap x \notin A \cap B\}$$

$$\begin{aligned} A = B &\iff A \oplus B = \emptyset \\ A \oplus B &= A \cup B - A \cap B \\ &= (A \cup B) \cap (\overline{A \cap B}) \\ &= (A \cup B) \cap (\overline{A} \cup \overline{B}) \end{aligned}$$

M_{R_5} = on input "D_1" "D_2"

Algorithm VI.5 M_{R_5}

construct D_3 with $L(D_3) = L(D_1) \oplus L(D_2)$
run M_{R_4} on "D_3"
output the result of M_{R_4}

C_1 $A_{CFG} = \{\text{"}G\text{" "}w\text{"} : G \text{ is a CFG that generates } w\}$
M_{C_1} = on input "G""w"

Algorithm VI.6 M_{C_1}

$G \to$ an equivalent CFG G' in CNF.
enumerate all the derivations of length $2|w|-1$(the number of derivations $\leq |R|^{2|w|-1}$).
if any of them generates w **then**
  accepts "G""w"
else
  rejects "G""w"
end if

C_2 $A_{PDA} = \{\text{"}P\text{""}w\text{"} : P \text{ isa PDA that accepts } w\}$
M_{C_2} = on input "P""w"

Algorithm VI.7 M_{C_2}

$P \to$ an equivalent CFG G
run M_{C_1} on "G""w"
return the result of M_{C_1}

C_3 $E_{CFG} = \{\text{"}G\text{"} : G \text{ is a CFG with } L(G) = \emptyset\}$
M_{C_3} = on input "G"

Algorithm VI.8 M_{C_3}

mark terminals and e
If right is marked, mark left, until it can't be marked.
if S is marked **then**
  rejects "G"
else
  accepts "G"
end if

C_4 $E_{PDA} = \{\text{"}P\text{"} : P \text{ is a PDA with } L(P) = \emptyset\}$
M_{C_4} = on input "P"

Algorithm VI.9 M_{C_4}

$P \to$ an equivalent CFG G
run M_{C_3} on "G""w"
return the result of M_{C_3}

Definition VI.2. *Let A and B be two languages over Σ_A and Σ_B respectively. A reduction from A to B is a recursive(computable) function $f : \Sigma_A^* \to \Sigma_B^*$, such that $\forall x \in \Sigma_A^*$,*

$$x \in A \iff f(x) \in B$$

called $A \leq B$

Theorem VI.3. *Suppose that $\exists$ a reduction from A to B. If B is recursive, A is recursive. (If A is not recursive, then B is not recursive.)*

Proof. B is recursive $\to \exists M_B$ decides B.

Let M_A = on input "x",

Algorithm VI.10 M_A

```
compute f(x)
run M_B on f(x)
return the result of M_B
```

Q.E.D.

Theorem VI.4. *Suppose that we have a reduction f from A to B. If B is recursively enumerable, A is recursively enumerable.*

Proof. Since B is recursive enumerable, there is a Turing machine M_B that semidecides B. We can construct M_A = on input "x"

Algorithm VI.11 M_A

```
compute f(x)
run M_B on f(x)
```

M_A halts on $x \iff M_B$ halts on $f(x) \iff f(x) \in B \iff x \in A$. So M_A semidecides A. Q.E.D.

Theorem VI.5. *If $A \le B$, then $\overline{A} \le \overline{B}$*

Proof. Let f be a reduction from A to B. By definition, for any $x \in \Sigma^*$,$x \in A$ iff $f(x) \in B$. In other words, $x \in \overline{A}$ iff $f(x) \in \overline{B}$. So f is also a reduction to $\overline{A}$ to $\overline{B}$. Q.E.D.

Theorem VI.6. *If A and $\overline{A}$ are recursively enumerable, A is recursive.*

Proof. $\exists M_1, M_2$ semidecides $A, \overline{A}$ respectively.

Target: construct M_3 to decide A

M_3 = on input x

Algorithm VI.12 M_3

```
run M_1 and M_2 on x in parallel
if M_1 halts on x then
    accept x
else if M_2 halts on x then
    reject x
end if
```

Q.E.D.

3. Countable

Definition VI.7. *Two sets A and B are equinumerous(等势), if $\exists$ bijection $f : A \to B$.*

Lemma VI.8. *A is countable if and only if $\exists$ injection $f : A \to \mathcal{N}$*

Proof.

$\to$ is simple.

$\leftarrow$ if A is finite, trivially

else sort A in increasing order of $f(a)$. Let $g(a)$ = rank of a. $g(a)$ is bijection.

Q.E.D.

Corollary VI.9. *Any subset of a countable set is countable.*

Proof. A is a countable set.
$\Rightarrow \exists$ injection $f : A \to \mathcal{N}$
$\Rightarrow \exists$ injection $f' : A' \to \mathcal{N}$, $A' \subseteq A$
$\Rightarrow A'$ is countable. Q.E.D.

Lemma VI.10. *Let Σ be an alphabet. Σ^* is countable.*

Proof idea: 从短到长排列所有字符串，并编号，这是个 bijection.

Corollary VI.11. *$\{M : M$ is a Turing Machine$\}$ is countable*

Lemma VI.12. *Let Σ be some non-empty alphabet. Let $\mathcal{L}$ be the set of all languages over Σ. $\mathcal{L}$ is uncountable.*

Proof. Suppose that $\mathcal{L}$ is countable. The languages in $\mathcal{L}$ can be labelled as $L_1, L_2, \ldots$

Since Σ^* is countable, the strings in Σ^* can be labelled as $s_1, s_2, \ldots$

Let $D = \{s_i : s_i \notin L_i\}$, $\forall i$, $s_i \in D \iff s_i \notin L_i$.
$\Rightarrow D \neq L_i$
$\Rightarrow D \notin \mathcal{L}$, contradiction. Q.E.D.

It's same as Diagonalization.(用来证实数不可数的)

From **Corollary** VI.11 and **Lemma** VI.12, $\exists$ language is not recursively enumerable.

4. Halting Problem

$H = \{$"M""w" $: M$ is a TM that halts on $w\}$

Theorem VI.13. *H is recursively enumerable.*

Proof. Let U = on input "M""w"

Algorithm VI.13 U

run M on w

U halts on "M""w" $\iff$ M halts on w $\iff$ "M""w" $\in H$. U is called universal Turing Machine Q.E.D.

Theorem VI.14. *H is not recursive.*

Proof. $H_d = \{$"M" : M is a TM that doesn't halt on "M"$\}$

1) If H is recursive, then H_d is recursive.
H is recursive $\Rightarrow \exists M_H$ decides H.
M_d = on input "M".

Algorithm VI.14 M_d

run M_H on "M""M"
if M_H accepts "M""M" **then**
  rejects "M"
else
  accepts "M"
end if

$\Rightarrow M_d$ decides H_d
$\Rightarrow H_d$ is recursive.
2) H_d is not recursively enumerable.
Assume H_d is recursively enumerable.
$\Rightarrow \exists D$ semidecides H_d

$$D \text{ on input "}M\text{"} \begin{cases} \text{halt} & \text{if } M \text{ doesn't halt on "}M\text{"} \\ & (\text{"}M\text{"} \in H_d) \\ \text{looping} & \text{if } M \text{ halts on "}M\text{"} \\ & (\text{"}M\text{"} \notin H_d) \end{cases}$$

$$D \text{ on input "}D\text{"} \begin{cases} \text{halt} & \text{if } D \text{ doesn't halt on "}D\text{"} \\ & (\text{"}D\text{"} \in H_d) \\ \text{looping} & \text{if } D \text{ halts on "}D\text{"} \\ & (\text{"}D\text{"} \notin H_d) \end{cases}$$

$\Rightarrow$ contradiction

So H is not recursive. Q.E.D.

e.g.

1) $L_1 = \{$"M" : M is a TM that halts on $e\}$ is not recursive.
"M""w" $\in H \iff$ "M^*" $\in L_1$.
M halts on $w \iff M^*$ halts on e.
M^* = on input "u"

Algorithm VI.15 M^*

run M on w

M^* halts on e
$\iff M^*$ halts on some input
$\iff M^*$ halts on every input
$\iff M$ halts on w.
$H \leq L_1$
2) $L_2 = \{$"M" : M is a TM that halts on some input$\}$ is not recursive.
$H \leq L_2$, same as L_1
3) $L_3 = \{$"M" : M is a TM that halts on every input$\}$ is not recursive.
$H \leq L_3$, same as L_1
4) $L_4 = \{$"M_1""M_2" : M_1 and M_2 are two TMS with $L(M_1) = L(M_2)\}$ is not recursive.
$L_3 \leq L_4$
"M" $\iff$ "M_1""M_2"
M halts on every input $\iff L(M_1) = L(M_2)$
$\iff L(M) = \Sigma^*$.
Let $M_1 = M$
M_2 = on input "x"

Algorithm VI.16 M_2

halt

$\Rightarrow L(M_2) = \Sigma^*$
M_{L_3} = on input "M"

Algorithm VI.17 M_{L_3}

construct a TM M^* that halt on every input
run M_{L_4} on "M""M^*"
return the result of M_{L_4}

reduction $f($"M"$)$ = "M""M^*" from L_3 to L_4.
5) $R_{TM} = \{$"M" : M is a Turing Machine with $L(M)$ being regular $\}$ is not recursively enumerable
target: $H \leq \overline{R_{TM}}$.
Assume $M_{R_{TM}}$ decides R_{TM}, we will ues it to construct M_H decides H.
M_H = on input "M""w"

Algorithm VI.18 M_H

construct a TM M' = on input x
run $M_{R_{TM}}$ on "M'"
return the result of $M_{R_{TM}}$

Algorithm VI.19 M'

run M on w
run U on x

$$L(M') = \begin{cases} \emptyset & \text{If } M \text{ doesn't halt on } w \\ L(U) = H & \text{If } M \text{ halt on } w \end{cases}$$

$\therefore\ H \leq \overline{R_{TM}}$, $\therefore\ \overline{H} \leq R_{TM}$

$\therefore\ R_{TM}$ is not respectively enumerable.

6) $CF_{TM} = \{$"M" : M is a TM with $L(M)$ being context-free $\}$

$H \leq \overline{CF_{TM}}$

7) $REC_{TM} = \{$"M" : M is a TM with $L(M)$ being recursive $\}$

$H \leq \overline{REC_{TM}}$

Theorem VI.15 (Rice's Theorem)**.** *Let $\mathcal{L}(P)$ be the set with property P. If $\mathcal{L}(P)$ is a non-empty proper subset of all recursive enumerable language,*

$$\{\text{“}M\text{”} : M \textit{ is a TM with } L(M) \in \mathcal{L}(P)\}$$

is not recursive.

Proof.

case 1 $\emptyset \notin \mathcal{L}(P)$, called $R(P)$

$H \leq R(P)$

$\exists L \in \mathcal{L}(P) \Rightarrow \exists M_A$ semidecides A.

$M_H =$ on input "M""w"

Algorithm VI.20 M_H

construct a TM $M^* =$ on input x
run M_R on "M^*"
return the result of M_R

Algorithm VI.21 M^*

run M on w
run M_L on x

$$L(M^*) = \begin{cases} L(M_A) = A & \text{if } M \text{ halts on } w \\ \emptyset & \text{if } M \text{ doesn't halt on } w \end{cases}$$

case 2 $\emptyset \in \mathcal{L}(P)$

$H \leq \overline{\mathcal{L}(P)}$

Q.E.D.

5. recursively enumerable

prove recursively enumerable

- by def
- $A \leq$ known recursively enumerable language

e.g. $\{A :$ "M" $: M$ is a TM that halts on some input$\}$ is recursively enumerable

Proof. $M_A =$ on input "M"

Algorithm VI.22 M_A

Require: "M"
  for $i = 1, 2, \ldots$ **do**
    for $s = s_1, \ldots, s_i$ **do**
      run M on s for i steps
      if M halts on s within i steps **then**
        halt
      end if
    end for
  end for

Q.E.D.

prove not recursively enumerable

- known non-recursively enumerable $\leq A$
- **Theorem** VI.6

H is recursively enumerable and is not recursive $\Rightarrow \overline{H}$ is not recursively enumerable.

6. Closure property

Table VI.1: Closure property

	recursive	recursively enumerable	CFL	regular
$\cup$	$\surd$	$\surd$	$\surd$	$\surd$
$\cap$	$\surd$	$\surd$	$\times$	$\surd$
$^-$	$\surd$	$\times$	$\times$	$\surd$
$\circ$	$\surd$	$\surd$	$\surd$	$\surd$
*	$\surd$	$\surd$	$\surd$	$\surd$

Proof.

- $M_{A \cup B} =$ on input x

Algorithm VI.23 $M_{A\cup B}$

run M_A on x
run M_B on x
if at least one of them accepts x **then** accept
else reject
end if

- $M_{A\circ B}$ = on input x

Algorithm VI.24 $M_{A\circ B}$

enumerate all possible u and v with $x = uv$
run M_A on u and run M_B on v
if both accept **then** accepts
else reject
end if

- $\bar{H}$ is not recursively enumerable.

Q.E.D.

7. Enumerator

Definition VI.16. *We say a TM M enumerate a language L. If for some state q,*

$$L = \{w : (s, \triangleright\sqcup) \vdash_M^* (q, \triangleright\sqcup w)\}$$

called Turing enumerable.

Theorem VI.17. *A language L is Turing enumerable iff it's recursively enumerable.*

Proof. If L is finite, trivially. Assume L is infinite.

$\Rightarrow$ $\exists M$ enumerate L,
goal: M' semidecides L
M' = on input x

Algorithm VI.25 M'

run M to enumerate L
every tiem M accepts a string w
if $w == x$ **then**
halt
end if

$\Leftarrow$ $\exists M$ semidecides L
goal: M' enumerate L
M' = on input x

Algorithm VI.26 M'

for $i = 1, \ldots, 2$ **do**
for $s = s_1, \ldots, s_i$ **do**
run M on s for i steps
if M halts on s within i steps **then**
accept s
end if
end for
end for

Q.E.D.

Definition VI.18. *Let M be a TM that enumerates L. We say M lexicographically enumerates L if whenever*

$$(q, \triangleright\sqcup w) \vdash_M^+ (q, \triangleright\sqcup w')$$

we have w' is after w in lexicographical order.

Theorem VI.19. *L is lexicographically enumerable iff it's recursive.*

Proof.

$\Leftarrow$ $\exists M$ decide L
M' = on input

Algorithm VI.27 M'

enumerate all string in lexicographical order
for each string s **do**
run M on s
if M accepts s **then**
accepts
end if
end for

$\Rightarrow$ $\exists M$ lexicographically enumerate L
goal: M' decides L
M' = on input x

Algorithm VI.28 M'

run M to enumerate all strings in L with lexicographical order $\leq x$
every time M accepts a string w
if $w == x$ **then**
accept
else
reject
end if

Q.E.D.

VII Numerical Functions

$$f : N^k \to N \ k \geq 0$$

Definition VII.1 (Numerical Functions)**.** *A TM M computes $f : N^k \to N$ if any $n_1, \ldots, n_k \in N$*

$$M(bin(n_1), \ldots, bin(n_k)) = bin(f(n_1, \ldots, n_k))$$

1. Basic Functions

1) zero function

$$zero(n_1, \ldots, n_k) = 0$$

for any $n_1, \ldots, n_k \in N$

2) identity function

$$id_{k,j}(n_1, \ldots, n_k) = n_j$$

3) successor function

$$succ(n) = n + 1$$

2. Operation

Definition VII.2 (composition)**.** *Let*

$$\begin{aligned} g &: N^k \to N \\ h_1, \ldots, h_k &: N^l \to N \end{aligned}$$

we have $f : N^l \to N$

$$f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l))$$

call f as the composition of g with $h_1, \ldots, h_k$.

Definition VII.3 (Recursive definition)**.** *Let*

$$\begin{aligned} g &: N^k \to N \\ h &: K^{k+2} \to N \end{aligned}$$

we have $f : N^{k+1} \to N$

$$\begin{cases} f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k) \\ \ldots \\ f(n_1, \ldots, n_k, t+1) = h(n_1, \ldots, n_k, t, f(n_1, \ldots, n_k, t)) \end{cases}$$

e.g. $n!$

$$\begin{cases} k = 0 \\ g = 1 \\ h(n, f(n)) = (n+1)f(n) \end{cases}$$

3. primitive recursive function

Definition VII.4 (primitive recursive function)**.**

$$basic\ functions + \begin{cases} composition \\ recursive\ definition \end{cases} = primitive\ recursive\ function$$

All primitive recursive functions is computable.

e.g.

1) $puls2(n) = n + 2$

$$succ(succ(n))$$

2) $puls(m, n) = m + n$

$$\begin{cases} plus(m, 0) & = id_{1,1}(m) \\ plus(m, n+1) & = plus(m, n) + 1 \\ & = succ(id_{3,3}(m, n, plus(m, n))) \end{cases}$$

平时写到就行 $succ(plus(m, n))$.

3) $mult(m, n) = m \cdot n$

$$\begin{cases} mult(m, 0) & = zero(m) \\ mult(m, n+1) & = mult(m, n) + m \\ & = plus(m, mult(m, n)) \end{cases}$$

4) $exp(m, n) = m^n$

$$\begin{cases} exp(m, 0) & = m \\ exp(m, n+1) & = exp(m, n) \cdot m \\ & = mult(m, exp(m, n)) \end{cases}$$

5) constant function: $f(n_1, \ldots, n_k) = c$

$$(succ \ldots (succ(zero(n_1, \ldots, n_k))))$$

c successor functions

6) sign function

$$sgn(n) = \begin{cases} 1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

$$\begin{cases} sgn(0) & = 0 \\ sgn(n+1) & = 1 \end{cases}$$

7) predecessor function

$$pred(n) = \begin{cases} n-1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

$$\begin{cases} pred(0) & = 0 \\ pred(n+1) & = id_{2,1}(n, pred(n)) \\ & = n \end{cases}$$

8) $m \sim n = \max(m - n, 0)$ non-negative substraction

$$\begin{cases} m \sim 0 & = 0 \\ m \sim (n+1) & = (m \sim n) - 1 \\ & = pred(m \sim n) \end{cases}$$

Lemma VII.5.

$$primitive\ recursive\ func + \begin{cases} composition \\ recursive\ def \end{cases} = primitive\ recursive\ func$$

Corollary VII.6. *if f, g is primitive recursive function, then $f + g, f \cdot g, f \sim g$ are all primitive recursive functions.*

Definition VII.7 (predicate function)**.** *The function's output is 1 or 0, called predicate function.*

Corollary VII.8. *If predicate p, q is primitive recursive, so is $\neg p, p \wedge q, p \vee q$*

1) $positive(n) = sgn(n)$
2) $iszero(n)$

$$iszero(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n > 0 \end{cases}$$
$$= 1 - positive(n)$$

3) $geq(m, n)$

$$geq(m, n) = \begin{cases} 1 & \text{if } m \geq n \\ 0 & \text{if } m < n \end{cases}$$
$$= iszero(m \sim n)$$

4) $eq(m, n)$

$$eq(m, n) = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}$$
$$= gep(m, n) \cdot gep(n, m)$$

Corollary VII.9. *Let*

$$f(n_1, \ldots, n_k) = \begin{cases} g(n_1, \ldots, n_k) & \text{if } p(n_1, \ldots, n_k) \\ h(n_1, \ldots, n_k) & \text{otherwise} \end{cases}$$

If g, h, p are primitive recursive, so is f.

Proof. $f = p \cdot g + (1 \sim p) \cdot h$ Q.E.D.

1) $rem(m, n) = m\%n$

$$rem(m, n) = \begin{cases} rem(0, n) & = 0 \\ rem(m+1, n) & = d(m+1, n) \end{cases}$$

$$d(m+1, n) = \begin{cases} 0 & \text{if } m+1 \text{ is divisible by } n \\ rem(m, n) + 1 & \text{otherwise} \end{cases}$$

$$m + 1 \text{ is divisible ny } n \iff eq(rem(m, n), pred(n))$$

2) $div(m,n) = \lfloor m/n \rfloor$, $n \neq 0$

$$div(m,n) = \begin{cases} div(0,n) & = 0 \\ div(m+1,n) & = d(m+1,n) \end{cases}$$

$$d(m+1,n) = \begin{cases} div(m,n)+1 & \text{if } m+1 \text{ is divisible by } n \\ div(m,n) & \text{otherwise} \end{cases}$$

3) $digit(m,n,p) = a_{m-1}$ with least digit

$$n = a_k p^k + \cdots + a_{m-1}p^{m-1} + \cdots + a_1 p_1 + a_0$$
$$digit(m,n,p) = div(rem(n,p^m), p^{m-1})$$

4) $sum_f(m,n) = \sum_{k=0}^{n} f(m,k)$, f is primitive recursive.

$$\begin{cases} sum_f(m,0) & = f(m,0) \\ sum_f(m,n+1) & = sum_f(m,n) + f(m, succ(n)) \end{cases}$$

5) $mult_f(m,n) = \prod_{k=0}^{n} f(m,k)$, f is primitive recursive.

6) Let p be a primitive recursive predicate from $N \to \{0,1\}$

$$\begin{aligned} g_p(n) &= \begin{cases} 1 & \text{if } \exists\, n' \leq n, p(n') = 1 \\ 0 & \text{otherwise} \end{cases} \\ &= p(0) \vee \cdots \vee p(n) \\ &= positive(\sum_{n'=0}^{n} p(n')) \\ &= positive(sum_p(n)) \end{aligned}$$

$$\begin{aligned} h_p(n) &= \begin{cases} 1 & \text{if } \forall\, n' \leq n, p(n') = 1 \\ 0 & \text{otherwise} \end{cases} \\ &= positive(\prod_{n'=0}^{n} p(n')) \\ &= positive(mult_p(n)) \end{aligned}$$

Definition VII.10.

$$PR = \{``f" : f \textit{ is a primitive recursive function}\}$$
$$C = \{``f" : f \textit{ is computable function}\}$$

$PR \subseteq C$, but $PR \neq C$

Lemma VII.11. *PR is decidable.*

Lemma VII.12. *C is undecidable.*

Proof. Assume that C is decidable. So $C' = \{$ unary computable numerical function $\}$ is decidable. So C' is lexicographically Turing enumerable by $g_1, g_2 \ldots$

Let $M =$ on input n

Algorithm VII.1 M

enumerable C' to get g_n
compute $g_n(n)$
return $g_n(n) + 1$

$g^*(n) = g_n(n) + 1$ is unary computable function, so $g^* \in C'$.

But $\because g^*(n) = g_n(n) + 1$, $\therefore \forall n, g^* \neq g_n$.
contradiction. Q.E.D.

4. Extend Basic Functions

Definition VII.13 (minimalization)**.** *Let $g : N^k \to N$, we have*

$$f(n_1, \ldots, n_k) = \begin{cases} m = n_{k+1} & \textit{if exists minimum } n_{k+1} \\ & \textit{such that } g(n_1, \ldots, n_{k+1}) = 1 \\ 0 & \textit{otherwise} \end{cases}$$

called a minimalization of g, $\mu m[g(n_1, \ldots, n_k, m) = 1]$

Theorem VII.14. *A function g is minimalizable if g is computable and $\forall n_1, \ldots, n_k, \exists m \geq 0, g(n_1, .., n_k, m) \geq 1$*

e.g. $\log(m,n) = \lceil \log_{m+2}(n+1) \rceil$ i.e. minimum $p \in N$, s.t. $(m+2)^p \geq n+1$

$$\log(m,n) = \mu p\,[gep((m+2)^p, n+1) = 1]$$

Theorem VII.15. *If $g(n_1, \ldots, n_{k+1})$ is minimalizable, the $\mu m[g(n_1, \ldots, n_k, m) = 1]$ is computable.*

Theorem VII.16. *$L = \{``g" : g$ is minimalizable $\}$ is undecidable*

Definition VII.17 (μ-recursive)**.**

$$\mu\textit{-recursive} = \textit{basic functions} + \begin{cases} \textit{composition} \\ \textit{recursive def} \\ \textit{minimalization of} \\ \quad \textit{minimalizable func} \end{cases}$$

Theorem VII.18. *A numerical function is μ-recursive iff it's computable.*

Proof. $\Rightarrow$ trivial

$\Leftarrow$, f, $\exists M$ accepts f. $n \to f(n)$ is

$$(s, \triangleright \underline{\sqcup} n) \vdash_M (q_1, \triangleright u_1 \underline{a_1} v_1) \vdash_M \cdots \vdash_M (h, \triangleright \underline{\sqcup} f(n))$$

用 state 标记读写头位置, i.e.

$$\triangleright \sqcup sn \quad \triangleright u_1 a_1 q_1 v_1 \quad \cdots \quad \triangleright h \sqcup h f(n)$$

Let $\Sigma \cup K \rightarrow \{0,1,2,\ldots,b-1\}$, the computation can be a base-b integer.

Therefore, we can

$$\begin{array}{cc} n & \\ \downarrow & 1) \\ \rhd \sqcup sn & \\ \downarrow & 2) \\ \rhd \sqcup sn \;\; \rhd u_1 a_1 q_1 v_1 \;\; \cdots \;\; \rhd h \sqcup hf(n) & \\ \downarrow & 3) \\ \rhd h \sqcup hf(n) & \\ \downarrow & 4) \\ f(n) & \end{array}$$

把计算过程分解为四个函数的叠加, 每个函数是 μ-recursive 的, 总体就是 μ-recursive.

1) $h_1(n) = \rhd \sqcup sb^{\lceil \log_b n+1 \rceil} + n$
   这里是为了留够 n 的空间.
2) $\mu m[iscomp(\rhd \sqcup sn, m) \wedge halted(m)]$. $iscomp$ and $halted$ are both primitive recursive
3) $\mu m[digit(m,n,b) == \rhd] = k$, and $rem(n, b^k)$
4) $\mu m[digit(m,n,b) == h] = k$, and $rem(n, b^k)$

Q.E.D.

VIII Grammar(unrestricted grammar)

grammar: $uAv \rightarrow w$

Definition VIII.1. *A grammar is a 4-tuple $G = (V, \Sigma, S, R)$*

- *V is a alphebte*
- *Σ is the set of terminals*
- *$S \in V - \Sigma$: start symbol*
- *R: a finite set of $(V^*(V_\Sigma)V^*) \times V^*$*

Definition VIII.2 (derive in one step and derive)**.** 类似之前的.

G generates a string $w \in \Sigma^*$ if $S \vdash_G^* w$

e.g. $\{a^n b^n c^n : n \geq 0\}$

- $S \rightarrow ABCS$
- $BA \rightarrow AB$
- $CA \rightarrow AC$
- $CB \rightarrow BC$
- $S \rightarrow T_c, CT_c \rightarrow T_c c, BT_c \rightarrow BT_b$
- $BT_b \rightarrow T_b b, AT_b \rightarrow AT_a$
- $AT_a \rightarrow T_a a, T_a \rightarrow e$

Theorem VIII.3. *A language is generateed by some grammar iff it's semidecided by some TM*

Proof.

$\Rightarrow$ Given G, construct M to semidecide $L(G)$, i.e. given w, is $S \Rightarrow_G^* w$?

进行枚举, 因为每一步最多只有 $|R|$ 种选择, 若有 w halt.

$\Leftarrow$ Given M, construct G to generate $L(M)$, i.e. $S \Rightarrow_G^* w$ iff $w \in L(M)$.

假设图灵机 halting state 唯一, 并且停机时纸带为空.

$$(S, \rhd \underline{\sqcup} w) \vdash_M (q_1, \rhd u_1 \underline{a_1} v_1) \vdash_M \cdots \vdash_M (h, \rhd \underline{\sqcup})$$

于是改造图灵机, 让其用 state 标记读写头, 并且停机时输出为空.

$$\rhd \sqcup sw \lhd \vdash_M \rhd u_1 a_1 q_1 v_1 \lhd \vdash_M \cdots \vdash_M \rhd \sqcup h \lhd$$

S 就进行 M 的回溯, 从最后的 configuration 推到最初的 configuration.

$$\begin{aligned} S \Rightarrow^* \rhd \sqcup h \lhd \Rightarrow^* \cdots &\Rightarrow^* \rhd u_1 a_1 q_1 v_1 \lhd \\ &\Rightarrow^* \rhd \sqcup sw \lhd \\ &\Rightarrow^* w \end{aligned}$$

于是增加如下规则:

– $S \to \triangleright \sqcup h \triangleleft$
– $\triangleright \sqcup s \to e$
– $\triangleleft \to e$
– 剩下的情况只有三种可能: 写或左右移

1) If M has $\delta(q,a) = (p,b)$ for some $a, b \in \Sigma$.

$$\because uaqv \vdash_M ubpv$$
$$\therefore bp \to aq$$

2) If M has $\delta(q,a) = (p,\to)$

$$\because uaqbv \vdash_M uabpv$$
$$\therefore abp \to aqb$$

for each $b \in \Sigma$.

但有特殊情况, 若 a 之后都是空格, i.e.

$$\because uaq\triangleleft \vdash_M ua \sqcup p\triangleleft$$
$$\therefore a \sqcup p\triangleleft \to aq\triangleleft$$

3) M has $\delta(q,a) = (p,\leftarrow)$ 类似

Q.E.D.

IX Computational Complexity

decidable:

- efficiently solvable (tractable)
- inefficiently solvable (intractable)

resource: time and space
e.g. :$A = \{0^k 1^k : k \geq 0\}$

Definition IX.1. *Let M be a deterministic TM that halts on every input.*

The running time f M is a function $f : N \to N$, where on any input of length n, M halts within $f(n)$ steps.

Definition IX.2. *We say a language L is in DTIME($t(n)$) if L is decided by some standard DTM with running time $O(t(n))$*

If Multiple tapes TM is $t(n)$, single tape TM is at most $O(t^2(n))$. 因为多带每走一步, 单带需要以一次扫描确定多带的每一个位置并修改, 扫一遍的时间是 $t(n)$.

Thesis IX.2.1 (The Cobham-Edmond Thesis)**.** *Any "reasonable" and "general" deterministic model of computational is polynomially related.*

Definition IX.3. *Let P (polynomially decidable) be the set of language that are decided by some DTM with poly(n) time. (robust)*

Theorem IX.4. *Every context-free language is in P*

Proof. If A is context-free, $\exists$ CFG $G = (V, \Sigma, S, R)$ in CNF generates A, given a string of length n, enumerate all derivations of length $\geq n-1$ (total $|R|^{n-1}$)

Dynamic Programing,

- $w = a_1 \dots a_n$, $S \Rightarrow^* w$?
- subproblem: for $1 \leq i \leq j \leq n$, define

$$T[i,j] = \{A \in V - \Sigma : A \Rightarrow^* a_i \dots a_j\}$$

- Goal: $T[1,n]$, $S \in T[1,n]$
- Base case: for $1 \leq i \leq n$,

$$T[i,i] = \{A \in V - \Sigma : A \to a_i\}$$

- Recurrence: for $1 \leq i \leq j \leq n$

$$T[i,j] = \bigcup_{k=i}^{j-1} \{A \to BC : B \in T[i,k] \wedge C \in T[k+1,j]\}$$

- n^2 subproblems, each subproblem costs: $n \cdot |R|^3$.

total: $\left(n^3 |R|^3\right)$, and R is constant Q.E.D.

1. SAT(satisfiability) Problem

Let $X = \{x_1, \dots, x_n\}$ be a set of boolean variable.

- $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$: literals
- a clause is like $x_1 \vee x_2 \vee \bar{x}_3$
- a boolean formular is like

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee x_3 \vee x_5)$$

- A truth assignment of X is $T : X \to \{0, 1\}$

SAT: given a formular F, is there any assignment T that makes F true/ satisfies F.

We don't know whether $SAT \in P$ or not.

2. NP

Definition IX.5. *Let M be a non-deterministic TM such that for any input every branch of M halts within k steps, where k depends only on the input.*

The running time of M is a function $f : N \to N$, for any input of length n, every branch of M halts within $f(n)$ steps.

SAT can be decided by NTM poly time.

$M =$ on input F

Algorithm IX.1 M

Require: F
  non-deterministically generate an assignment T
  if T satisfies F **then**
    accept
  else
    reject
  end if

Definition IX.6. *NP is the set of all languages that can be decided by some NTM in poly time.*

Definition IX.7. *We say that a language A is polynomially verifiable, if there is a polynomially DTM V such that for $x \in \Sigma^*$,*

1) if $x \in A$, $\exists$ y with $|y| \leq poly(|x|)$ s.t. V accepts "x""y" (completeness)

2) if $x \notin A$, $\forall$ y with $|y| \leq poly(|x|)$, V rejects "x""y" (soundness)

called y certificate

e.g.

- $A =$ SAT
- $x =$ boolean formular
- $y =$ an truth assignment that satisfies x
- $V =$ on input x and y

Algorithm IX.2 V

  evaluate x using y
  if x is satisfied by y **then**
    accept
  else
    reject
  end if

Theorem IX.8. *A language is polynomially verifiable iff it's NP*

Proof. $\Rightarrow$ $\exists$ polynomial-time verifier V

Algorithm IX.3 M (If $x \in A$?)

Require: x
  non-deterministically generate a certificate y with $|y| \leq poly(|x|)$
  run V on "x""y"
  if V accepts **then**
    accept
  else
    reject
  end if

$\Leftarrow$ $\exists$ NTM M decides A in $poly(|x|)$ time.

construct V, for $x \in A$, certificate $y =$ branch that accepts x, $|y| \leq poly(|x|)$

Algorithm IX.4 V

Require: x and y
  run M on x under the guidance of y
  if M accepts **then**
    accept
  else
    reject
  end if

Q.E.D.

$P \subseteq NP$

$P = NP$? i.e. 解决问题的难度与验证问题的难度是否相等.

widely believe $P \neq NP$

X NP-complete Problem

Definition X.1. *A NP-complete problem is in p iff $P = NP$.*

1. polynomial-time reductions

Definition X.2. *A reduction f from A to B ($A \le B$). And f can be computed by some DTM within poly(n) time. Called,*

$$A \le_p B$$

Theorem X.3. *If $A \le_p B$, and $B \in P$, then $A \in P$*

Proof.

$$x \to f(x) \to \text{ decide } f(x) \in B \to$$

Totsl time:

$$poly(|x|) + poly(|f(x)|) = poly(|x|)$$

where $|f(x)| \le poly(|x|)$ Q.E.D.

e.g. Obviously, $3SAT \le_p SAT$, $\because$ $f(x) = x$
But we also have $SAT \le_p 3SAT$

Proof. 对每一个 set, 多拆少补.

1) $(x_1 \vee x_2) \Rightarrow (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \bar{y})$
2)

$$\begin{aligned}
&(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \\
\Rightarrow&(x_1 \vee x_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_4 \vee x_5) \\
\Rightarrow&(x_1 \vee x_2 \vee y) \wedge (\bar{y} \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)
\end{aligned}$$

Q.E.D.

Definition X.4 (Clique)**.** *For $G = (V, E)$, a clique of G is a subgraph $G' = (V', E')$ with $V' \subseteq V$, $E' \subseteq E$ such that G' is a complete graph.*

$$Clique = \{\text{“}G\text{”“}k\text{”} : G \textit{ has a clique with at least } k \textit{ nodes}\}$$

$3SAT \le_p Clique$

Proof. m clause F and $3m$ nodes G. F is satisfiable $\iff$ G has a clique with at least m nodes. Q.E.D.

Definition X.5 (vertex cover)**.** *A vertex cover of G is a subset $V' \subseteq V$ s.t. for each $e \in E$, e has at least one endpoint in V'.*

$$VC = \{\text{“}G\text{”“}k\text{”} : G \textit{ has a vertex cover with at most } k \textit{ nodes}\}$$

$3SAT \le_p VC$

Proof. F is satisfiable $\iff$ G has a VC with at most $n+2m$ nodes. Q.E.D.

Definition X.6. *A language L is NP-complete if*

1) $L \in NP$
2) $\forall L' \in NP, L' \le_p L$

Theorem X.7. *If a NP-complete language $L \in P$, $NP = P$.*
If a NP-complete language $L \notin P$, $NP \ne P$

2. Cook's Theorem

Theorem X.8 (Cook's Theorem)**.** *SAT is NP-complete*

Proof. Let A be an arbitrary language in NP,

$$A \le_p SAT$$

i.e. $x \in A \iff F$ is satisfiable
$\exists NTM N$ decides A in n^k times.

$$a_1, \ldots, a_n \in A$$

$$\iff$$

Q.E.D.

3. space