作业题的扩展(历年卷×)

讨论 20% . 测验 20% . 期末 60% . 作业 0%

Lec 01

3. hardness complexity class → complexity theory 复杂性理论

为了研究复杂度

1. definitions of problem & computing model → automata theory

   - finite automata
   - pushdown automata
   - Turing machines

church-Turing Thesis: 图灵机是终级计算模型

2. computability theory 可计算理论

Optimization Problem

Given a graph $G=(V,E,w)$, what is the MST?

Search Problem

.... and an integer k, find a spanning tree with weight at most k.

Decision Problem

.................., is there a spanning tree with weight at most k

Counting Problem

Decision Problem.

抽象为形式化的问题

, is there a spanning tree with weight at most k

- yes-instance
- no-instance

这个问题由这个 set 决定

Given a string w, $w \in \{\text{encodings of yes-instances}\}$?

↓ a language （所有判定问题可以抽象为一个 yes-instance 的集合）

Σ

An alphabet is a finite set of symbols.

e.g. $\Sigma = \{0, 1\}$, $\{a, b, c \cdots z\}$

$= \{\triangle, \circ, \square, \times\}$

$= \{\ \} \rightarrow$ empty

A string over Σ is a finite sequence of symbols from Σ

e.g. $\Sigma = \{0, 1\}$ 0, 1, 00100 ……

Length $|w| =$ # symbols in w (# : number of) empty string : e with $|e| = 0$

Σ^i = the set of all strings of length i over Σ.

e.g. $\Sigma = \{0, 1\}$ 则 $\Sigma^0 = \{e\}$ $\Sigma^1 = \{0, 1\}$ $\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^* = \bigcup_{i=0} \Sigma^i$ $\Sigma^+ = \bigcup_{i \geq 1} \Sigma^i$

concatencation

e.g. $u = 123$, $v = 456$. $uv = 123456$

exponentation

$w^i = \underbrace{w \cdots w}_{i \text{ times}}$ e.g. $w = 01$ $w^0 = e$ $w^2 = 0101$

reversal

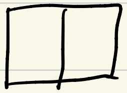
$w = a_1 \cdots a_i$ $w^R = a_i \cdots a_1$

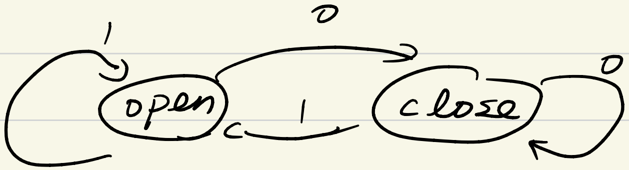
Any subset of Σ^* is a language over Σ.

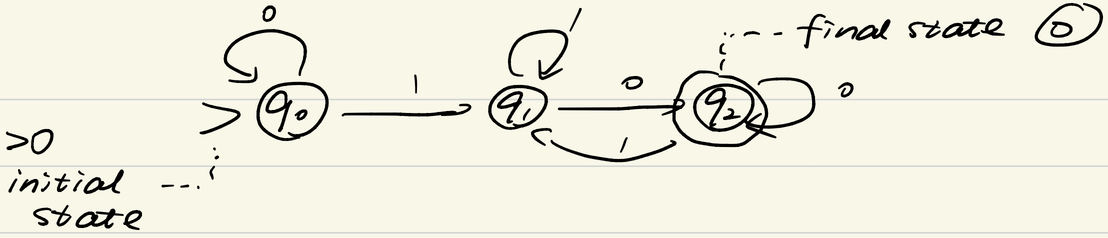
decision problems <=> languages

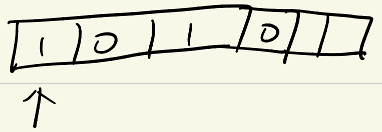
Given a string w, $w \in L$? <= L

finite automata









$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_1 \xrightarrow{0} q_2$$

A finite automata $M = (K, \Sigma, \delta, s, F)$

- Σ: input alphabet 纸带上的字符
- K: set of state
- $s \in K$: initial state 唯一
- $F \subseteq K$: the set of final states 可以为空,可以多个
- transition functions $\delta: \underset{\text{current state}}{K} \times \underset{\text{symbol}}{\Sigma} \longrightarrow \underset{\text{next state}}{K}$

e.g. $\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$ ......

$$(q_0, 1010) \overset{\text{一步}}{\vdash_M} (q_1, \cancel{1}010) \vdash_M \underbrace{(q_2, \cancel{1}010)}_{\text{configuration}} \vdash_M (q_1, \underset{\text{不重要}}{\Delta}10) \vdash_M (q_2, 1\cancel{01}0)$$

configuration

an element of $\underset{\text{current state}}{K} \times \underset{\text{unread input}}{\Sigma^*}$

yields in one step

$(q, w) \vdash_M (q', w')$ if $w = aw'$ for some $a \in \Sigma$ 走一步

$\delta(q, a) = q'$

$\vdash_M^*$ → yields

if $(q, w) = (q', w')$ or 走若干步

$(q, w) \vdash_M \cdots \vdash_M (q', w')$

M accepts $w \in \Sigma^*$ if $(s, w) \vdash_M^* (q, e)$ for some $q \in F$ 把input读完到final state

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

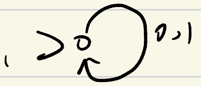
L language of M (由M唯一确定) M accepts $L(M)$

含义不同

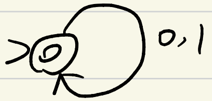
对象是language，M accepts $L \iff \begin{cases} \forall w \in L \cdot M \text{ accepts } w \\ \forall w \notin L \cdot M \text{ does not accepts } w. \end{cases}$

每一台M. accept的L有且仅有一个
(L(M)子集也不行)

Exercise



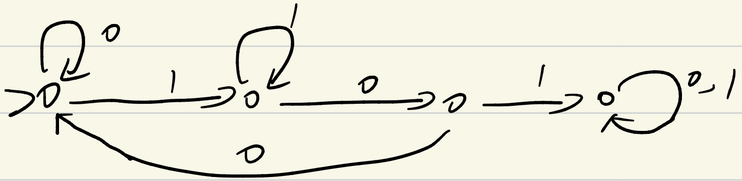
没有final state $\Rightarrow L(M) = \phi$ ($\{e\}$ ✗)



$L(M) =$
$\{0,1\}^*$

A language is regular if it is accepted by some FA.

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



Regular Operations

封闭的. 操作后也仍是regular

Union. $A \cup B = \{w : w \in A \text{ or } w \in B\}$

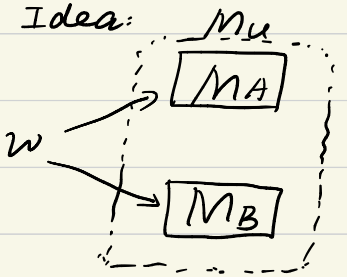
Concatenation $A \cdot B = \{ab : a \in A \text{ and } b \in B\}$ e.g. $A = \{good, bad\}$ $B = \{dog, cat\}$

$A \cdot B = \{gooddog, goodcat, baddog, badcat\}$

Star $A^* = \{w_1 w_2 \cdots w_k : w_i \in A \text{ and } k \geq 0\}$ e.g. $B^* = \{e, dog, cat, dogdog, catcat, dogcat, catdog \cdots\}$

Theorem: if A and B are regular, so is $A \cup B$.

Idea:



Proof: $\exists M_A = (K_A, \Sigma, \delta_A, s_A, F_A)$ accepts A

假设AB字符集同（否则union）

$\exists M_B = (K_B, \Sigma, \delta_B, s_B, F_B)$ ..... B.

$M_U = (K_U, \Sigma, \delta_U, s_U, F_U)$

$K_U = K_A \times K_B$ 并行,同时跑

$s_U = (s_A, s_B)$

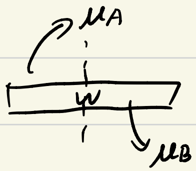
$F_U = \{(q_A, q_B) \in K_A \times K_B : q_A \in K_A \overset{\triangle}{\underline{\text{or}}} q_B \in K_B\}$

δ_U : for any $q_A \in K_A$, $q_B \in K_B$, any $a \in \Sigma$

$$\delta_U((q_A, q_B), a) = (\delta_A(q_A, a), \delta_B(q_B, a))$$

Theorem: if A and B are regular, so is $A \circ B$.

Idea:



切在哪? unknown!

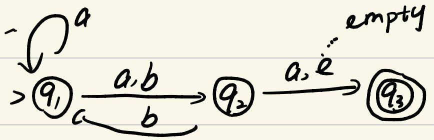
Non-determinism 非确定性

确定: $q \xrightarrow{a}$ "function" => deterministic f auto (DFA)

唯一确定 $(s,w) \vdash_m$ $\vdash_m (q,e)$ unique for each w

non - … … (NFA)

$q_1 \xrightarrow{a} q_1, q_2$



1. several choices for next state
2. e-transition 不读字符也能改变状态

$\Delta = \{(q_1, a, q_1), (q_1, a, q_2) \cdots, (q_2, e, q_3)\}$ a relation

三元tuple

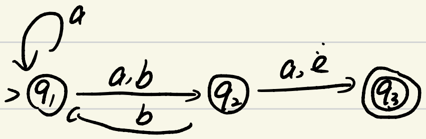
a NFA is a 5-tuple $(K, \Sigma, \Delta, s, F)$

transition relation $\Delta \subseteq K \times \Sigma \cup \{e\} \times K$

configuration $(q, w) \in K \times \Sigma^*$

$\vdash_M$ $\vdash_M^*$

Example



input: abb

$\vdash_m (q_3, b)$

$(q_1, abb) \vdash_m (q_1, bb) \vdash_m (q_2, b) \vdash_m (q_1, e)$

$\vdash_m (q_2, bb) \vdash_m (q_1, b) \vdash_m (q_2, e)$

$\top_M$ (q_3, bb)

$\top_M$ (q_3, e)

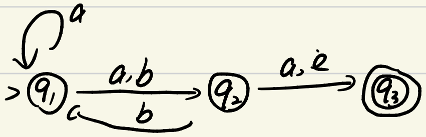
M accepts w if $(s, w) \vdash_m^* (q, e)$ for some $q \in F$

存在一条路即可

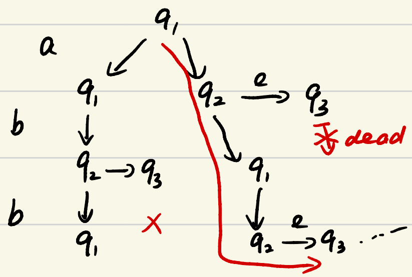
$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$. M accepts $L(M)$

"理解角度"（并非真实运作逻辑）

Parallel



input: abb



活下来的进程，有final state则接受

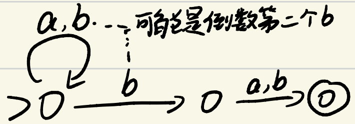
Magic

always make the right guess.

Example:

倒数第二个

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

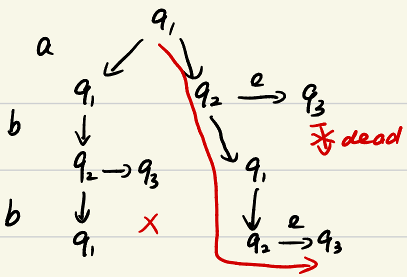


Theorem: ① $\forall$ DFA $M \rightarrow \exists$ NFA M' s.t. $L(M') = L(M)$

DFA也是一种特殊的NFA

② $\forall$ NFA $M \rightarrow \exists$ DFA M' s.t. $L(M) = L(M')$

Idea: DFA M' simulate "tree-like" computation of M



$\{q_1\}$
$\downarrow$
$\{q_1, q_2, q_3\}$
$\downarrow$
$\{q_1, q_2, q_3\}$
$\downarrow$
$\{q_1, q_2, q_3\}$

把一层看作一个点，模拟每一层的变化

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

DFA $M'=(K', \Sigma, \delta, s', F')$

$K' = 2^K = \{Q : Q \subseteq K\}$

$F' = \{Q \subseteq K : Q \cap F \neq \phi\}$

$s' = \cancel{\{s\}}\ E(s)$ （可能有: $s \xrightarrow{e} q_1 \xrightarrow{e} q_2$ ）

从q出发,不读symbol情况下能到的点

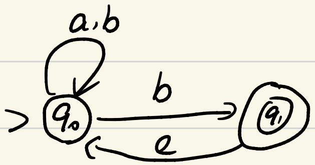
$\forall q \in K.\ E(q) = \{p \in K : (q, e) \vdash_M^* (p, e)\}$

δ: for $\forall Q \subseteq K$, $\forall a \in \Sigma$. $\delta(Q, a) = \bigcup_{q \in Q} \bigcup_{p:(q,a,p) \in \Delta} \cancel{\{p\}}\ E(p)$

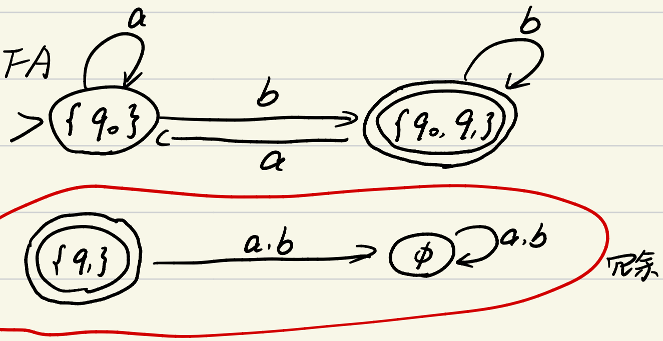
NFA: Q $\downarrow$ Q' $\quad q_1, \ldots\ldots\ q_k$ $\downarrow\downarrow$

Example

NFA:



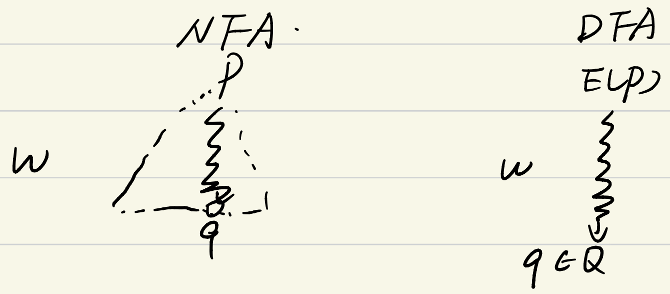
=> DFA



证明: NFA M accepts w $\Leftrightarrow$ DFA M' accepts w.

Claim. for $p, q \in K$ and $w \in \Sigma^*$.

$(p, w) \vdash_M^* (q, e)$ iff $(E(p), w) \vdash_{M'}^* (Q, e)$ for $q \in Q$.



by induction on $\overset{|w|}{\text{length}}$ of w

→ 套定义

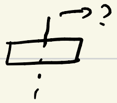
假设 claim 成立 : M. accepts w $\Leftrightarrow (s, w) \vdash_M^* (q, e)$ with $q \in Q$

$\Leftrightarrow (E(s), w) \vdash_{M'}^* (Q, e)$ with $Q \ni q$ ($\begin{matrix} Q \cap F \neq \phi \\ Q \in F' \end{matrix}$)

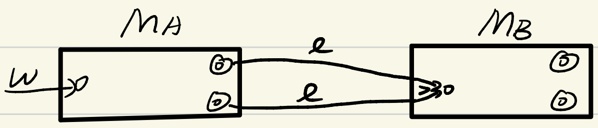
$\Leftrightarrow$ M' accepts w.

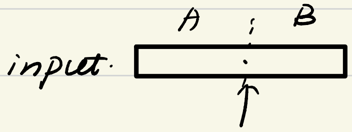
Corollary : regular $\Leftrightarrow$ $\exists$ NFA

Theorem : if A and B are regular, so is $A \circ B$.



Proof: $\exists$ NFA M_A, M_B accepts A and B.





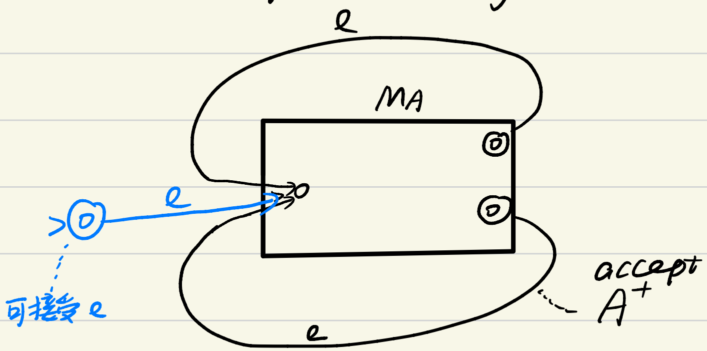
$M_A = (K_A, \Sigma, \Delta_A, S_A, F_A)$

$M_B = (K_B, \Sigma, \Delta_B, S_B, F_B)$ $\Rightarrow$ $M^{\circ} = (K^{\circ}, \Sigma, \Delta^{\circ}, s^{\circ}, F^{\circ})$

则 $K^{\circ} = K_A \cup K_B$ $\quad s^{\circ} = S_A$ $\quad F^{\circ} = F_B$

$\Delta = \Delta_A \cup \Delta_B \cup \{(q, e, s_B) : q \in F_A\}$

Theorem : if A is regular, so is A^*.



$e \in A^*$

$M_A = (K_A, \Sigma, \Delta_A, S_A, F_A)$ $\quad M^* = (K^*, \Sigma, \Delta^*, S^*, F^*)$

$K^* = K_A \cup \{s^*\}$

$F^* = F_A \cup \{s^*\}$

$\Delta^* = \Delta_A \cup \{(q, e, S_A) : q \in F_A\} \cup \{(s^*, e, S_A)\}$ ✓ 构造写出即可

$$\begin{tikzcd} DFA \arrow[r, bend left, leftrightarrow, "equiv"] & NFA \end{tikzcd}$$

$\Downarrow$

regular languages

$\Downarrow$

closure property

$\cup . \circ . * . \cap . -$

Regular Expression (REX)

表达式 ---- $R = (a \cup b)^* a$

$L(R) = (\{a\} \cup \{b\})^* \circ \{a\}$ 以a结尾的ab串

Atomic

symbol

ϕ. $L(\phi) = \phi$

$a \in \Sigma$ $L(a) = \{a\}$

composite $\cup . \circ . *$

$R_1 \cup R_2$ $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$

$R_1 . R_2$ $L(R_1 R_2) = L(R_1) \circ L(R_2)$

R^* $L(R^*) = (L(R))^*$

Precedence : $* > \circ > \cup$

$ab^* \cup b^* a = ((a^*)b) \cup ((b^*)a)$

Example.

language	表达式
$\{e\}$	ϕ^*
$\{w \in (a,b)^* :$ w starts with a and ends with b$\}$	$a(a \cup b)^* b$
$\{ \cdot - \cdots \cdots :$ w contains at least 2 a's$\}$	$(a \cup b)^* a (a \cup b)^* a (a \cup b)^*$

Theorem: A language A is regular $\Leftrightarrow$ there is some REX R with $L(R) = A$

(只需证. NFA $\rightleftarrows$ REX)

$$\begin{tikzcd} \underset{L(M)=L(R)}{M} \arrow[r, shift right] & R \arrow[l, shift right] \end{tikzcd}$$

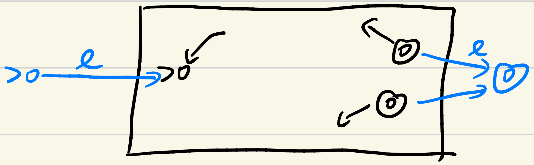
R → NFA M

NFA M ——→ REX R s.t. L(R) = L(M)

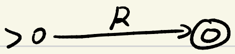
(1) simplify M

a) no arc enters the initial state

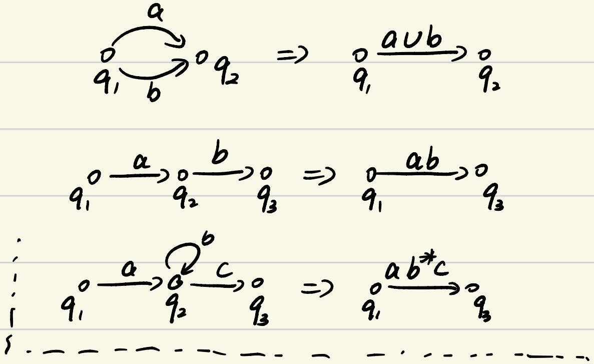
b) only one final state with no arc leaving it.



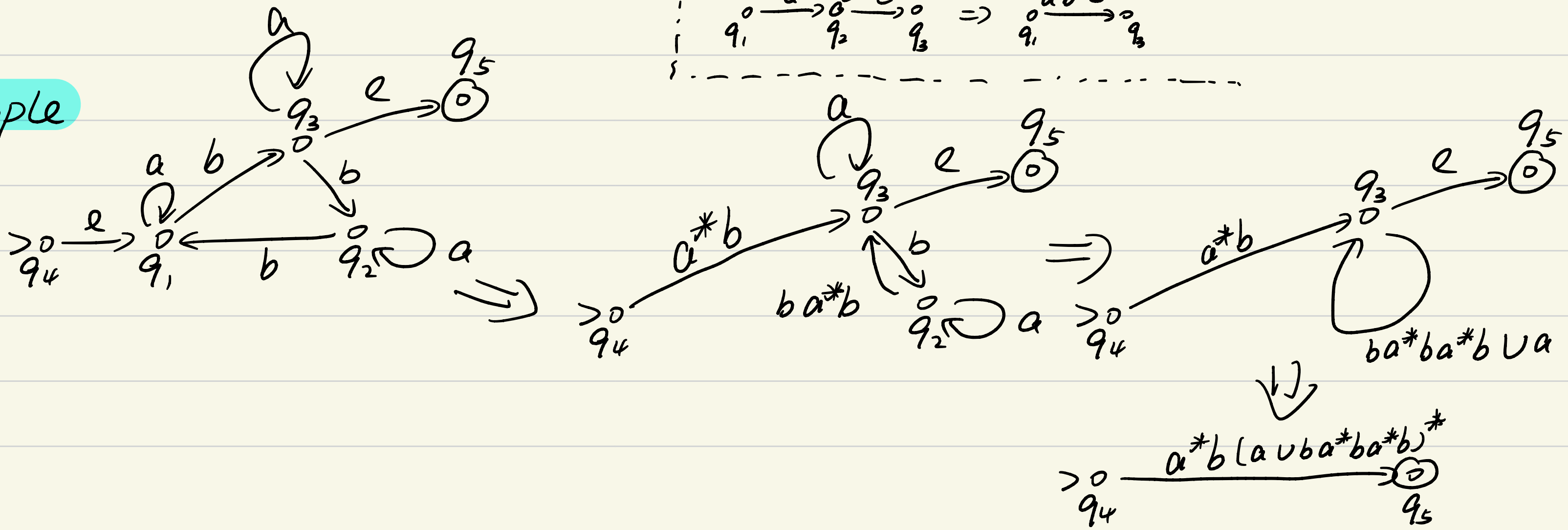
(2) eliminate states



e.g.



Example



Let $M = (K, \Sigma, \Delta, s, F)$ be a NFA

(1) $K = \{q_1, \cdots q_n\}$, $s = q_{n-1}$. $F = \{q_n\}$

(2) $(p, a, q_{n-1}) \notin \Delta$ for any $p \in K$ and $a \in \Sigma$

(3) $(q_n, a, p) \notin \Delta$

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

本质 (DP)

subproblems: for $i,j \in [1,n]$. for $k \in [0,n]$. define.

(不含 q_i, q_j)

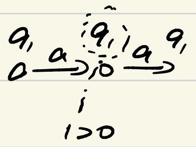
$L_{ij}^{k} = \{w \in \Sigma^* :$ w drive M from q_i to q_j with no intermediate state having index$>k$.

$\downarrow$

R_{ij}^{k}

在上图中

e.g. $L_{11}^{0} = \{a, e\}$ ⚠ aa is wrong since



$R_{11}^{0} = \phi^* \cup a$

$L_{13}^{0} = \{b\}$

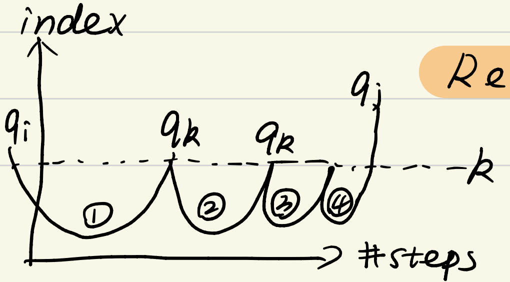
$L_{41}^{1} = \{e, a, aa \cdots\} =$

ans : $R_{(n-1)n}^{\underline{n-2}}$

Base case: $k=0$

if $i==j$ $L_{ii}^{0} = \{e\} \cup \{a : (q_i, a, q_i) \in \Delta\}$, R_{ii}^{0}

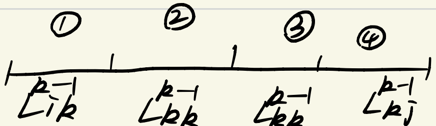
$i \neq j$ $L_{ij}^{0} = \{a : (q_i, a, q_j) \in \Delta\}$, R_{ij}^{0}



Recurrence

$$L_{ij}^{k} = L_{ij}^{k-1} \cup L_{ik}^{k-1} \circ (L_{kk}^{k-1})^* \circ L_{kj}^{k-1}$$

$$R_{ij}^{k} = R_{ij}^{k-1} \cup R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$$ (删状态 => ↑k)



Pumping theorem

Let L be regular language, There exists an integer $p \geq 1$ ↗ pumping length such that for $w \in L$ with $|w| \geq p$, we can divide into 3 pieces $w = xyz$ satisfying

(1) for any $k \geq 0$. $xy^k z \in L$

(2) $|y| \geq 1$

(3) $|xy| \leq p$

$\exists p \geq 1$

for any $w \in L$ with $|w| \geq p$ 只要串足够长，一定能抽出 y.（DFA 状态有限，总会经历重复状态.）

Proof:

If L is finite, Let $p = \max\limits_{w \in L} |w| + 1$. $\neg \exists$ string

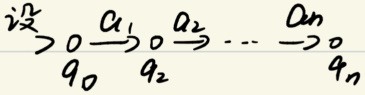
Assume L is infinite.

L is regular $\Rightarrow$ $\exists$ DFA M. accepting L.

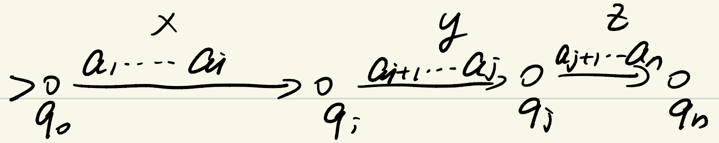
Let p = # states of M.

Take any $w \in L$ with $|w| \geq p$

$w = a_1 \cdots a_n$



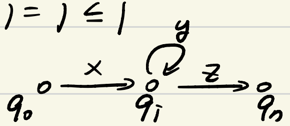
$\exists\ 0 \leq i < j \leq p$. $q_i = q_j$, 以 q_i . q_j 为界.三分



(2) $|y| = j - i \geq 1$ (3) $|xy| = j \leq 1$

0^r

(1) $xy^kz \in L$ for any $k \geq 0$



Example. prove $\{\underline{0^n1^n} : n \geq 0\}$ is not regular

反证: Assume L is regular, Let p be the pumping length given by the pumping theorem.

By pumping theorem. $0^p1^p \in L$ can be written as

1) for any $k \geq 0$, $xy^kz \in L$

2) $|y| \geq 1$

3) $|xy| \leq p$

$\overset{2)}{3)} \Rightarrow$ $y = 0^t$ for some $t \geq 1$ $\Rightarrow xy^0z = 0^{p-t}1^p \notin L$

contradicting (1)

Example. $\{w \in \{0,1\}^* : w$ contains equal numbers of 0's and 1's$\}$ is not regular.

可用pumping thm, 也可 Assume L is regular

$\Downarrow$

$L \cap a^*b^*$ is regular

$\Downarrow$

$\{0^n1^n : n \geq 0\}$

Regular Languages

$$\begin{tikzcd}
\text{DFA} \arrow[d, shift right] \\
\text{NFA} \arrow[u, shift right] \arrow[d, shift right] \\
\text{REX} \arrow[u, shift right]
\end{tikzcd}$$

closure. $\cup$. $\cap$. $\circ$. $*$. $\bar{\ }$.

pumping theorem (regular 的必要条件)

Context-free language

Context-free grammar (CFG)

左边可以替换为右边

$S \rightarrow aSb$ S: start symbol

$S \rightarrow A$ S, A: non-terminal

$A \rightarrow c$ a. b. c: terminal

$A \rightarrow e$

$S \overset{1}{\Rightarrow} aSb \overset{1}{\Rightarrow} aaSSbb \overset{2}{\Rightarrow} aaAbb \overset{4}{\Rightarrow} aabb$ 不断替换直到全是 terminals

A CFG $G = (V, \Sigma, S, R)$

- V: a "finite" set of symbols
- $\Sigma \subseteq V$: the set of terminals

  $V - \Sigma$: the set of non-terminals

- $S \subseteq V - \Sigma$: start symbol
- $R \subseteq (V - \Sigma) \times V^*$ e.g. $S \rightarrow aSb$

  non-terminal $\rightarrow w \in V^*$

for any $x.y.u \in V^*$, for any $A \in V$.

$$xAy \underset{\text{derive in one step.}}{\Rightarrow_G} xuy \text{ if } (A,u) \in R$$

for any $w.u \in V^*$

$$w \Rightarrow_G^* u \text{ if } w=u \text{ or } w \underbrace{\Rightarrow_G \cdots\cdots \Rightarrow_G}_{\text{derive from w to u of length n.}} u$$

G generates a string $w \in \overset{\text{terminals}}{\Sigma^*}$ if $S \Rightarrow_G^* w$

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

G generates $L(G)$

Definition: A language is context-free if some CFG generates it.

Example: $\{w \in \{a,b\}^* : w = w^R\}$ is context-free

$S \longrightarrow e \mid a \mid b \mid aSa \mid bSb$

Definition:

A CFG is an Chomsky norm form (CNF) if every of its rule is one of the following form:

1. $S \longrightarrow e$
2. $A \rightarrow BC$ for some $B.C \in V-\Sigma-\{S\}$
3. $A \rightarrow a$ for some $a \in \Sigma$

CNF 生成一个长为n的串,需要2n-1次

Theorem:

$\forall$ CFG $\longrightarrow$ CFG in CNF

proof sketch

1. S appears RHS $\Rightarrow$ new start symbol S_0, $S_0 \to S$

2. $A \to e$ for some $A \neq S$

e.g. $B \to ACA \to \underline{CA} \mid \underline{AC} \mid \underline{C}$. 删去后补上 $B \to CA$ 向前补偿

$B \to AC$

$B \to C$

3. $A \to B$ for some $B \in V - \Sigma$

e.g. $A \cancel{\to} B \to CDE$ 删去后补上 $A \to CDE$ 向后补偿

e.g. $C \cancel{\to} ADA$ 不能补 $C \to BDA$

ADB

BDB

$\because$ 若 $S = A$. 则 S 不能出现在 RHS

4、若 $A \to u_1 u_2 \cdots u_k$ $\overset{\text{RHS}}{k \geq 3}$

$\Rightarrow A \to u_1 V_2$ 右边长度为 2

$V_2 \to u_2 V_3$

$\vdots$

$V_{k-1} \to u_{k-1} u_k$

5. $A \to u_1 u_2$ at least one $\overset{\text{terminal}}{u_i \in \Sigma}$

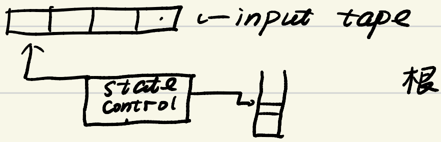
$A \to a_1 B \Rightarrow A \to A_1 B$

$A_1 \to a_1$

Pushdown Automata (PDA)

PDA ⇄ CFG

PDA = NFA + stack



根据 input 和 stack 里元素决定下一步

Definition:

A PDA is a 6-tuple $P = (\underset{\text{state}}{K}, \underset{\text{stack alphabet}}{\Gamma}, \overset{\text{input symbol}}{\Sigma}, \Delta, \overset{\text{start}}{s}, \overset{\text{final}}{F})$

Δ : transition relation   a finite "subset" of $(K \times (\Sigma \cup \{e\}) \boxed{\times \Gamma^*}) \times (K \times \boxed{\Gamma^*})$

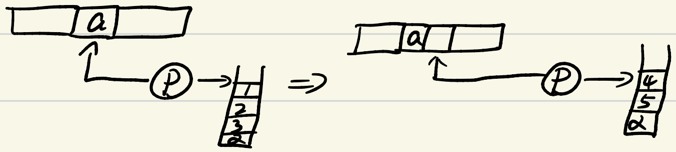
a string at the top of stack → pop

push onto stack

跳次态，删匹配串，push新串

Example

$((p, a, 123), (q, 45))$

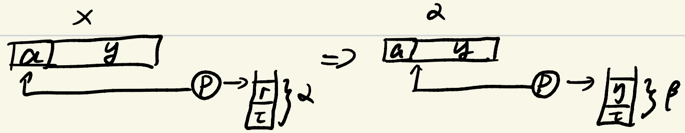


$((p, a, e), (q, \beta))$

└ 无论栈内有什么，均可匹配

A configuration of P is a member of $K \times \Sigma^* \times \underset{\text{stack element}}{\Gamma^*}$

$(p, x, \alpha) \vdash_P (q, y, \beta)$ if $\exists ((p, a, \alpha), (q, \eta)) \in \Delta$ s.t. $x = ay$, $\alpha = \gamma\tau$ and $\beta = \eta\tau$ for some $\tau \in \Gamma^*$



$(p,x,\alpha) \vdash_P^* (q,y,\beta)$ if $(p,x,\alpha)=(q,y,\beta)$ or $(p,x,a) \vdash_P \cdots \vdash_P (q,y,\beta)$

P accepts $w \in \Sigma^*$ if

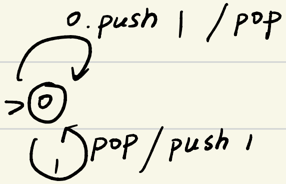
①到final
②input string 空
③stack空

$(s,w,e) \vdash_P^* (q,e,e)$ for some $q \in F$

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

P accepts $L(P)$

Example. $\{w \in \{0,1\}^* : \#\, 0\text{'s} = \#\, 1\text{'s}\}$



$\{((q,0,1),(q,e))$

$((q,0,e),(q,1))$

$((q,1,1),(q,e))$

$((q,1,e),(q,1))\}$

NFA! guess!

1. CFG $\rightleftarrows$ PDA

2. CFL properties: closure properties. pumping theorem

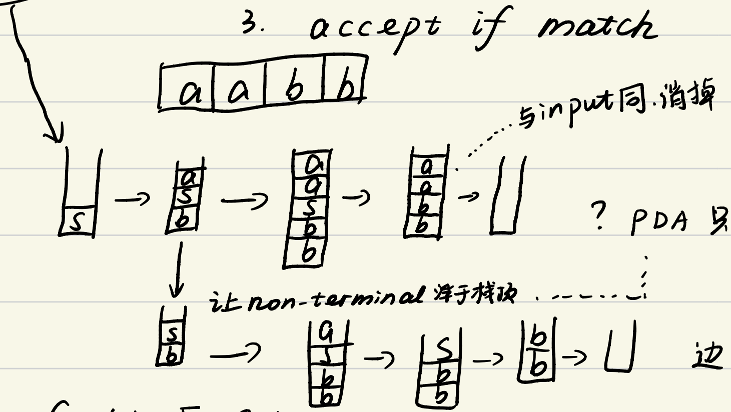
CFG $G \longrightarrow$ PDA M s.t. $L(M)=L(G)$

Idea:

1. in stack, non-determinitically generate a string from S.
2. compare it to the input
3. accept if match

$S \to aSb$

$S \to e$



? PDA 只能对栈顶有限长的串push/pop (若S藏在下面,难以操作)

边替换,边匹配

Given $G=(V, \Sigma, S, R)$

$\Rightarrow P=(K, \Sigma, T, \Delta, s, F)$

$K=\{s,f\}$ $F=\{f\}$

$T=V$

$\Delta = \{((s,e,e),(f,S))$ ①推S入栈

$((f,a,a),(f,e))$ for each $a \in \Sigma$ ②栈顶消掉terminal

$((f,e,A),(f,u))$ for each $(A,u) \in R\}$ ③栈顶non-terminal → 非确定换掉

$$\begin{tikzcd}
\text{PDA} \arrow[r] \arrow[d] & \text{CFG} \\
\text{Simple PDA} \arrow[ur] &
\end{tikzcd}$$

If $|F|=0$, trivial
Assume $|F|\geq 1$

Def: A PDA $M=(K,\Sigma,T,\Delta,s,F)$ is **simple** if

(1) $|F|=1$ and

(2) for each transition $((p,a,\alpha),(q,\beta))\in\Delta$

either $\alpha=e$ and $|\beta|=1$ 要么只push一个，要么只pop一个

or $|\alpha|=1$ and $\beta=e$

PDA → simple PDA

1. $|F|\neq 1$ add a new state F'

for each $q\in F$. add a new transition $((q,e,e),(f',e))$

$F:=\{f'\}$

2. 2.1 $|\alpha|\geq 1$ and $|\beta|\geq 1$ 同时push.pop

2.2 $|\alpha|>1$ and $\beta=e$ push >1 元素

2.3 $\alpha=e$ and $|\beta|>1$ pop>1

2.4 $\alpha=\beta=e$ nop

2.1 $((p,a,\alpha),(q,\beta))$ with $|\alpha|\geq 1$ and $|\beta|\geq 1$ 先做pop 再做push

└ add new state r

replace it with $((p,a,\alpha),(r,e))$ pop α

$((r,e,e),(q,\beta))$ push β

2.2 $((p,a,\alpha),(q,\beta))$ with $\beta=e$, $\alpha=c_1c_2\cdots c_k$. $k\geq 2$

add $k-1$ new states $r_1\cdots r_{k-1}$

$((p,a,c_1),(q,e))$ 拆成 k步

$((r_1,e,c_2),(r_2,e))$

$\vdots$

$((r_{k-1},e,c_k),(q,e))$

2.3 同理

2.4 $((p,a,e),(q,e))$

add a new state r

pick $b\in T$

$((p,a,e),(r,b))$ 先push再pop出来(同一个元素)

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

Simple PDA → CFG

Given a simple PDA $M=(K,\Sigma,T,\Delta,\ s,\{f\})\Rightarrow G=(V,\Sigma,S,R)$

↗subproblem

Nonterminal : $\{A_{pq}$: for any $(p,q)\in K\times K\}$

Goal: $A_{pq}\Rightarrow^* w\in\Sigma^*$ if and only if $(p,w,e)\vdash_M^*(q,e,e)$ 希望制定R使这项成立

$\therefore S=A_{sf}$ ($\because$

我们希望 $L(G)=L(M)$

$S\Rightarrow^* \underline{w}$ iff $w\in \underline{L(M)}$

$w\in L(G)$ $(s,w,e)\vdash_M^*(f,e)$

R : (recurrence)

① $\forall p \in K$

$A_{pp} \to e$

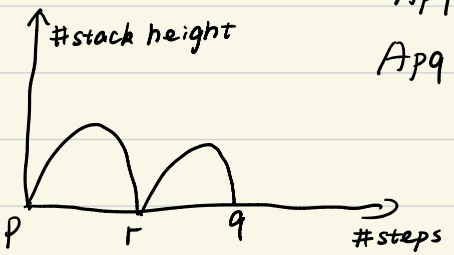
② $\forall p, q \in K$

(i)

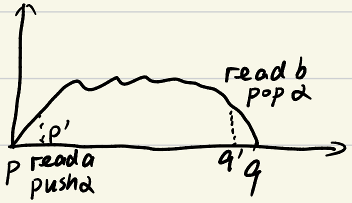
$A_{pq} \to A_{pr} A_{rq}$ $\underline{\forall r \in K}$ → 枚举.

$A_{pq} \to a A_{p'q'} b$ $\forall ((p, a, e), (p', \alpha)) \in \Delta$ for some $\alpha \in T$

$((q', b, \alpha), (q, e))$



(ii)



(∵ simple)

Prove that $A_{pq} \Rightarrow^* w \in \Sigma^*$ iff $(p, w, e) \vdash_M^* (q, e, e)$

⇒ by induction on length of derivation from A_{pq} to w

⇐ by induction on #steps of computation

PDA $\xrightarrow{\text{defines}}$ CFL

Theorem.

Every regular language is context-free. (∵ NFA → PDA → CFL)

CFL closure properties $\cup$. $\circ$. $*$ ✓

$\cap$. $\bar{A}$ ✗

A and B are context-free, so are $A \cup B$. $A \circ B$. A^*.

$$G_A = (V_A, \Sigma, S_A, R_A)$$

$$G_B = (V_B, \Sigma, S_B, R_B)$$

$G_{A \cup B}$: $S \rightarrow S_A \mid S_B$

$G_{A \circ B}$: $S \rightarrow S_A S_B$

G_{A^*} : $S \rightarrow e \mid S_A S$

$A = \{a^i b^j c^k : i = j\}$ context-free

$B = \{a^i b^j c^k : j = k\}$

$$A \cap B = \underline{\{a^n b^n c^n : n \geq 0\}}$$

not context-free (by pumping theorem)

$A \cap B = \overline{\overline{A} \cup \overline{B}}$ ∴ 补集也不封闭（否则 $A \cap B$ 也封闭）

Pumping theorem for CFL:

Let L be a context-free language. There exists an integer $p > 0$ such that any $w \in L$ with $|w| \geq p$ can be divided 5 pieces $w = uvxyz$ satisfying

(1) $uv^i x y^i z \in L$ for any $i \geq 0$

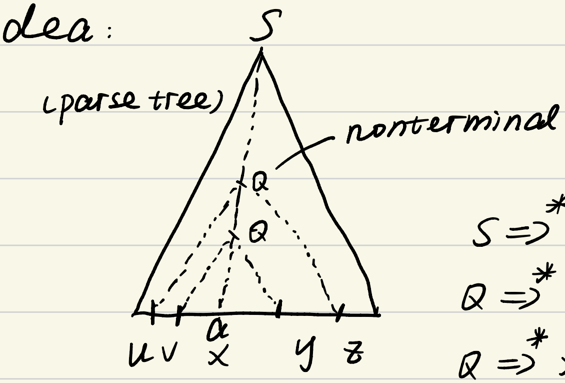
(2) $|v| + |y| > 0$ 不能同时为空

(3) $|vxy| \leq p$

$\{a^n b^n : n \geq 0\}$ $p=2$

$ab = \overset{}{\underset{u}{e}} \cdot \underset{v}{a} \cdot \underset{x}{e} \cdot \underset{y}{b} \cdot \underset{z}{e}$

Idea:



$S \overset{*}{\Rightarrow} uQz$

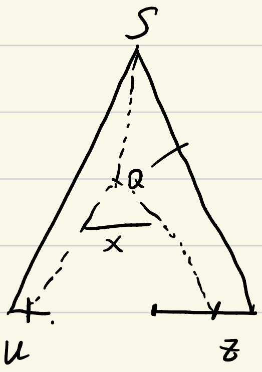
$Q \overset{*}{\Rightarrow} vQy$

$Q \overset{*}{\Rightarrow} x$

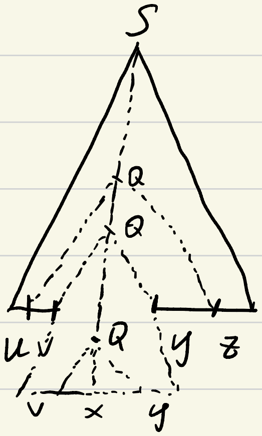
一条 path. 除了leave 均是 non-terminal

path足够长
必有重复的nonterminal

这里选最长的 S Q Q a



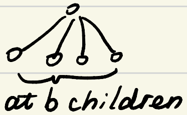
$S \overset{*}{\Rightarrow} uQz \overset{*}{\Rightarrow} uxz$



$S \overset{*}{\Rightarrow} uv^2xy^2z$

L is context-free $\Rightarrow \exists\, G=(V, \Sigma, S, R)$ generates L

Let $b = \max\{|u| : (A, u) \in R\}$ 最多儿子数 (规则右边的最长长度)
CFG



fanout $\leq b$

Fact: if a tree with fanout $\leq b$. has n leaves. then its height $\geq \log_b n$

(# edges of the longest descending path) define $p = b^{|V-\Sigma|+1}$, pick $w \in L$ with $|w| \geq p$

Let T be $\overset{a}{\wedge}$ parse tree that yields w (with smallest number of nodes)

height of $T \geq \log_b p = |V-\Sigma|+1$

$\#\text{edges} \geq |V-\Sigma|+1$

$\#\text{nodes} \geq |V-\Sigma|+2$

$\#\text{non-terminals} \geq |V-\Sigma|+1$

$\Downarrow$

some non-terminal Q appears at least twice

$\Downarrow$

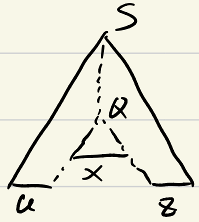
choose the lowest pair

(1) $uv^ixy^iz \in L$ for any $i \geq 0$ ✓

(2) $|v|+|y|>0$

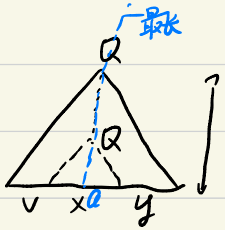
反证 if $v=y=e$.

$w = uxz$



is smaller than T. contradiction

(3) $|vxy| \leq p$



只用证

height $\leq |V-\Sigma|+1$?

$\Rightarrow |vxy| \leq \#\text{Leaves} \leq b^{|V-\Sigma|+1} = p$

height: Length of QQa. (∵ SQQa最长)

if every non-terminal appears at most once in the path (excluding endpoints) 就成立了

$\Downarrow$

选Q时，选最低的一对 pair

□

$\{a^n b^n c^n : n \geqslant 0\}$

assume it is context-free. Let p be the pumping length.

pick $a^p b^p c^p \in L$

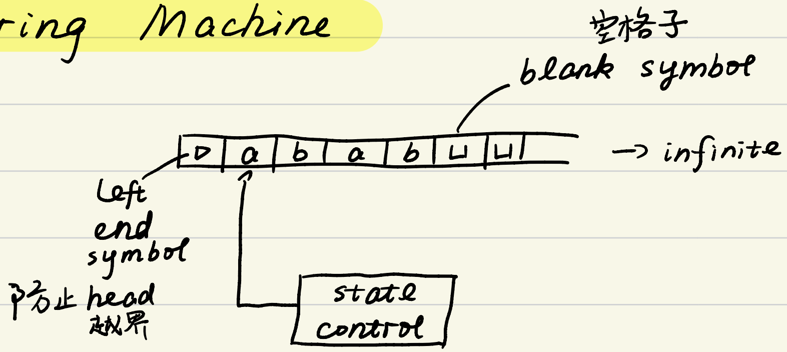
By pumping theorem. $a^p b^p c^p = uvxyz$

(3) $|vxy| \leq p \Rightarrow$ at least one of a and c does not appear in v or y.

$$\underbrace{a \cdots a}_{p}\underbrace{b \cdots b}_{p}\underbrace{c \cdots c}_{p}$$

$uv^0xy^0z \in L$ contradiction

Turing Machine



1. ← . →

2. read & write

Definition:

A Turing machine is 5-tuple $M=(K,\Sigma,\delta,s,H)$

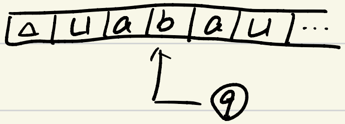
- K : a finite set of states
- Σ : tape alphabet (containg $\triangleright$ and $\sqcup$)
- $s\in K$: initial state
- $H\subseteq K$: a set of halting states
- δ : transition function

$$\overset{\text{不能是 halt}}{(K-H)}\times\underset{\text{当前格里的元素}}{\Sigma}\longrightarrow K\times(\overset{\text{移动}}{\{\leftarrow,\rightarrow\}}\underset{\text{head action}}{\cup}\overset{\text{写symbol}}{(\Sigma-\{\triangleright\})})$$

satisfy for any $q\in K$

在最左端
只能 →, 不能 ←. 也不能 overwrite

$\delta(q,\triangleright)=(p,\rightarrow)$ for some p



$(q, \triangleright \sqcup a\underline{b}a) \Leftrightarrow (q, \triangleright \sqcup ab, a)$

特殊

$(q, \triangleright \sqcup ab\underline{a}) \Leftrightarrow (q, \triangleright \sqcup aba, e)$

以某symbol结尾

A configuration a member of $K \times \triangleright(\Sigma - \{\triangle\})^* \times (\{e\} \cup (\Sigma - \{\triangle\})^* (\Sigma - \{\triangle, \sqcup\}))$

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

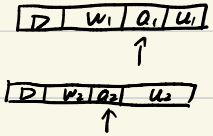
1) // writing

$\delta(q_1, a_1) = (q_2, a_2)$ $w_2 = w_1$, $u_2 = u_1$

2) // moving left

$\delta(q_1, a_1) = (q_2, \leftarrow)$ $w_1 = w_2 a_2$. $u_2 = a_1 u_1$

$($if $a_1 = \sqcup$. $u_1 = e$ then $u_2 = e)$



③ // moving right

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

① "   " = "   " or

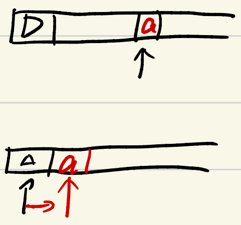
② " $\vdash_M \cdots \vdash_M \cdots \vdash_M$ " $n \geq 1$ steps

$(q, \triangleright w\underline{a}u)$ is a halting config if $q \in H$

那么 initial config $(s, ?)$

固定
Fix Σ.

(1) symbol writing machine M_a $(a \in \Sigma - \{\triangleright\})$ 作用：$\left\{ \begin{array}{l} \text{初始指向 symbol 为}\triangleright. \rightarrow.\text{then write } A \\ \cdots\cdots\cdots\cdots\text{不为}\triangleright.\text{ write } A. \end{array} \right\}$ then halt



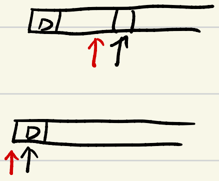
$M_a = (\{s, h\}, \Sigma, \delta, s, \{h\})$ for each $b \in \Sigma - \{\triangleright\}$,

$\delta(s, b) = (h, a)$

$\delta(s, \triangleright) = (s, \rightarrow)$

(2) head moving machine $M_\leftarrow$ $M_\rightarrow$. 左/右移读写头

若在▷. 则不动



basic machines : M_a. M_L. M_R

a L R

$a \sqcup b \sqcup c$

Left-shifting machine $S_\leftarrow$

for any $\nu \in (\Sigma - \{\triangleright . \sqcup\})^*$

$\triangleright \sqcup \sqcup w \underline{\sqcup} \longrightarrow \triangleright \sqcup w \underline{\sqcup}$ 整体左移一格

Example

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

1. run M_1, until it halts 停机时看此时读写头
2. if the current symbol is 0 , run M_2
3. . . . . 1 , run M_3
4. else halt.

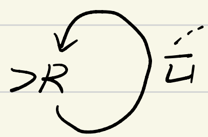
$>R \xrightarrow{\Sigma} R$

⇕

$>RR \Leftrightarrow >R^2$

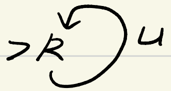
$>R \xrightarrow{a \neq \sqcup} Ra$ $(Ra : R \xrightarrow{\Sigma} a)$

$R_{\sqcup}$:



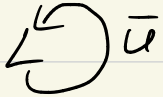
：找到当前读写头右第一个空格

$R_{\overline{\sqcup}}$

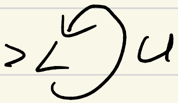


：……………… 非空格 （可能不会halt，即全是 空格）

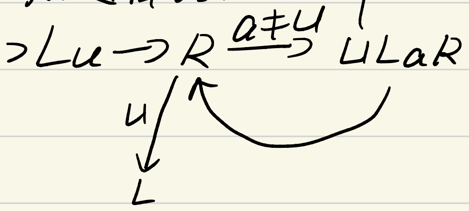
$L_{\sqcup}$



$L_{\overline{\sqcup}}$



∴ $S_{\leftarrow}$ 可以这样表示



当前写空格 pos

先移到左边第一个空格

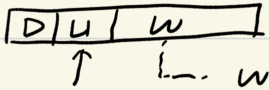
Recognize language

$M=(K,\Sigma,\delta,s,H)$

通过是否停机辨别语言

input alphabet $\Sigma_0 \subseteq \Sigma-\{\triangleright,\sqcup\}$

initial config : $(s,\triangleright\sqcup w)$



w是输入

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

M **semidecides** L(M) （需要时间可能很长，无法确定下一秒是否halt）

recursively enumerable (recognizable) if some TM semidecide it.

Let $M = (K, \Sigma_0, \Sigma, \delta, s, \{y, n\})$ be a TM.

We say M **decides** a language $L \subseteq \Sigma_0^*$ if

(1) for every $w \in L$,

$(s, \triangleright\underline{\sqcup}w) \vdash_M^* (y, \cdots)$ we say M accepts w.

(2) for every $w \in \Sigma_0^* - L$

$(s, \triangleright\underline{\sqcup}w) \vdash_M^* (n, \cdots)$ . . . . rejects .

A language is <u>recursive (deciable)</u> if some TM <u>decides</u> it.

Theorem:

If L is recursive, it must recursively enumerable. 判定更强

Compute functions 图灵机还可用来计算函数

for $w \in \Sigma_0^*$, if ① $(s, \triangleright\sqcup w) \vdash_M^* (h, \triangleright\sqcup y)$ for $h \in H$. ② $y \in \Sigma_0^*$

input output

$\triangleright$	$\sqcup$	y

↑

$y = M(w)$

称为 recursive / computable

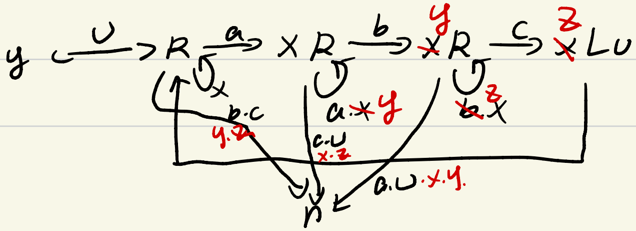
for any $f : \Sigma_0^* \longrightarrow \Sigma_0^*$, we say M **computes** f if for any $w \in \Sigma_0^*$, $M(w) = f(w)$

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

$\triangleright$	$\sqcup$	~~a~~	a	~~b~~	b	~~c~~	c	$\sqcup$	$\sqcup$

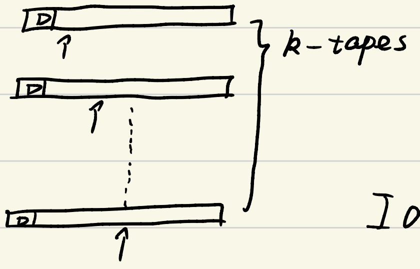
↑ ↑

每次删一个 a.b.c



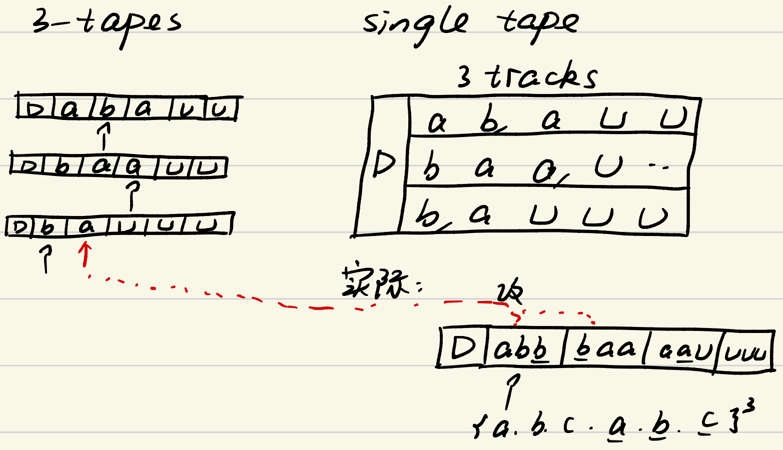
abc. abc ?
把 x 变为 x.y.z

1. multiple tapes

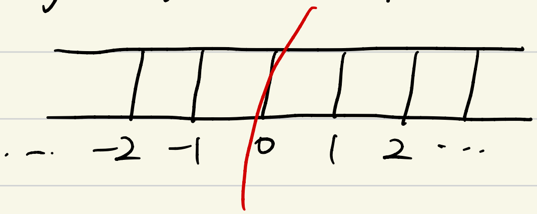


$\delta: (K-H)\times\Sigma^{k} \longrightarrow K\times(\{\Sigma-\Delta\}\cup\{\leftarrow \rightarrow\})^{k}$

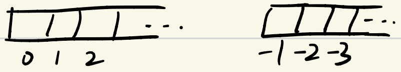
Idea:



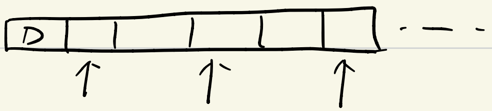
2. Two-way infinite tape



用 2-tape 模拟

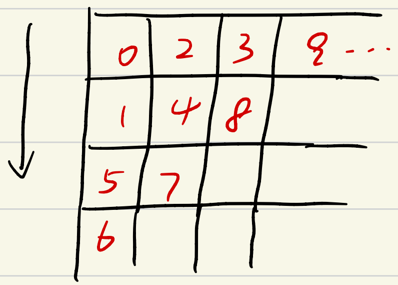


3. multiple head



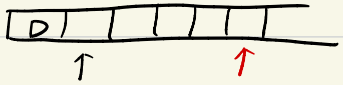
每次扫一遍确定头的位置. 再操作

4. Two-dimensional tape



0	2	3	9 ...
1	4	8	
5	7		
6			

5. random access



head 可以跳 ⇒ 拆成若干步

☆

6. non-deterministic TM (NTM)

非确定性图灵机

a NTM is a 5-tuple $(K, \Sigma, \Delta, s, H)$

Δ: relation (not function)

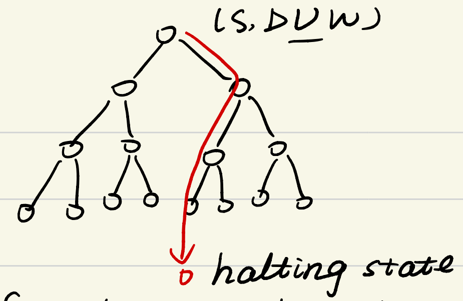
a finite subset of $((K-H)\times\Sigma)\times(K\times((\Sigma-\{\triangleright\})\cup\{\leftarrow\rightarrow\})$

configuration $(q, \triangleright ab\underline{a}bb)$

$\vdash_M$ $\vdash_M^*$ $\vdash_M^N$: yields in N steps

A NTM $M=(K,\Sigma,\Delta,s,H)$ with input alphabet Σ_0 semidecides $L\subseteq\Sigma_0^*$ if for any $w\in\Sigma_0^*$. $w\in L$ if and only if $(s, \triangleright\underline{\ }w)\vdash_M^*(h,\cdots)$ for some $h\in H$.

存在一条路



if $w \in L$.. some branches halt

$\notin$. no ....

Let $M = (K, \Sigma, \Delta, s, \{y, n\})$ with input alphabet Σ_0

M decides a language $L \subseteq \Sigma_0^*$ if

每一个分支均在 N步内停止

(1) for any $w \in \Sigma_0^*$. $\exists$ a natural number N. s.t. no configuration C satisfying

$(s, \triangleright\sqcup w) \vdash_M^N C$

即 $\forall$ 输入, $\overset{\exists N}{\wedge}$ 对应树 height $< N$

(2) $w \in L \iff \underline{(s, \triangleright\underline{\sqcup}w) \vdash_M^* (y, \cdots)}$ (若 $w \notin L$. 则所有分支都会停在n上)

some branch halts with y.

Example.

Let $C = \{$ binary encodings of $\overset{合数}{\text{composite}}$ numbers $\}$

猜是否是两个数的乘积

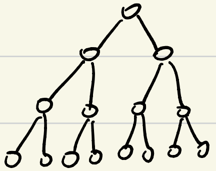
				猜		猜
$\triangleright$	$\sqcup$	w	$\sqcup$	p	$\sqcup$	q

(1) 有限步停机

(2) ✓

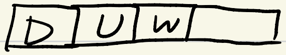
Theorem: Every NTM can be simulated by DTM.

proof (sketch): NTM N semidecides $L \to$ DTM M semidecides L

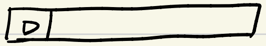


DTM searching for a halting state BFS (no DFS)

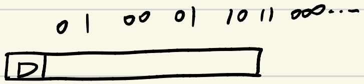
3-tape DTM to simulate N.



store the input



simulate N 在树上.向下走



enumerate hint 记录纸带往哪走 实际中多有限个分叉 (不一定 binary)

Church-Turing Thesis

算法本质就是 TM!

Intuition of algorithms equals (deterministic) Turing machines that halts on every input.

solves decides

(decision)

problem equals languages

编码

Fact: Any finite set can be encoded. $\{a_1 \cdots a_n\} \leftarrow \{a, 0, 1\}$

A finite collection of finite sets can be encoded.

$\{A. B. C. D\} \longleftarrow (a.b.c.d.0.1)$

$(a,0,1)$ $(b,0,1)$ $(c,0,1)$ $(d,0,1)$

$\Downarrow$

FA. PDA. TM. CFG. REX

Object D ⟶ "D" 表示它的编码

decide problem (recursive languages)

Problem

R1 $A_{DFM} = \{ \text{"D""w"} : D \text{ is a DFA that accepts } w \}$

```
M_R1 = on input "D" "w"
by default { 0.1 if input is illegal. reject
           { 0.2 decode "D""w" to obtain D and w
  1. run D on w
  2. if D ends with final / D accepts w
  3.         accept "D""w"
  4. else
  5.         reject "D""w"
```

R2. $A_{NFA} = \{ \text{"N""w"} : N \text{ is a NFA that accepts } w \}$

```
M_R2 = on input "N""w"
  1. N ⟶ an equivalent DFA D
  2. run M_R1 on "D""w"
  3. return the result of M_R1
```

R2 ⟶ R1

f: "N""w" ⟶ "D""w"　　对输入映射,且答案一样

"N""w" $\in A_{NFA} \iff$ "D""w" $\in A_{DFA}$

A reduction from A_{NFA} to A_{DFA}

归约

R3 $A_{REX} = \{$ "R""w" : R is a regular expression that generates w $\}$

```
$M_{R3}$ = on input "R""w"
    1. R ⟶ an equivalent NFA N
    2. run $M_{R2}$ on "N""w"
    3. return the result of $M_{R2}$
```

$f: A \to B$ and B is recursive ⇒ A is also recursive.

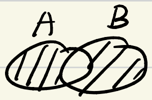
R4. $E_{DFA} = \{$ "D" : D is a DFA with $L(D) = \phi \}$

```
$M_{R4}$ = on input "D"
    1. if D has no final state
    2.         accept.
    3. else
    4.         "conceptually" do BFS on the diagram
    5.         if there is a path from s to a final.
                   reject
               else
                   accept
```

R5. $EQ_{DFA} = \{$ "D_1""D_2" ; D_1 and D_2 are two DFAs with $L(D_1) = L(D_2) \}$

Hint: ① 利用 R4

② symmetric $A \oplus B = \{ x \in A \cup B \wedge x \notin A \cap B \}$



③ $A = B \iff A \oplus B = \phi$

$$A \oplus B = A \cup B - A \cap B = (A \cup B) \cap (\overline{A \cap B})$$

$$= (A \cup B) \cap (\bar{A} \cup \bar{B})$$

$$\underset{\underset{D_1}{\uparrow}}{L(D_1)} = \underset{\underset{D_2}{\uparrow}}{L(D_2)} \iff \underset{\underset{D_3}{\overset{?}{\|}}}{(L(D_1) \cup L(D_2)) \cap (\overline{L(D_1)} \cup \overline{L(D_2)})} = \phi$$

M_{R5} = on input "D_1" "D_2"

1. construct D_3 with $L(D_3) = L(D_1) \oplus L(D_2)$ ⟶ 能 construct 出吗?
2. run M_{R4} on "D_3"
3. return the result of M_{R4}.

A. B. languages over same alphabet Σ

☆ A reduction from A to B is a recursive (computable) function

$f: \Sigma^* \longrightarrow \Sigma^*$ such that for $\forall x \in \Sigma^*$, $x \in A \iff f(x) \in B$

reduction from EQ_{DFA} to E_{DFA}

$f($"D_1" "D_2"$)$ = "D_3" (with $L(D_3) = L(D_1) \oplus L(D_2)$) recursive

f(illegal input) = illegal input (被 reject. by default)

"D_1" "D_2" $\in EQ_{DFA} \iff$ "D_3" $\in E_{DFA}$

Theorem:

If B is recursive, and $\exists$ a reduction f from A to B. then A is recursive.

即 $A \leq B$ (判定的难度)

Proof: $\exists M_B$ decides B.

```
$M_A$ = on input $x$.  $\xrightarrow[\text{可计算}]{f}$
    1. compute $f(x)$
    2. run $M_B$ on "$f(x)$"
    3. return the result of $M_B$.
```

Example

C1 = { "G""w" : G is a CFG that generates w }

```
$M_{C1}$ = on input "G""w"
    1. $G \to G'$ in CNF
    2. enumerate all derivations of length $2|w|-1$
    3. if any of them generates w.
    4.         accept "G""w"
    5. else
    6.         reject "G""w"
```

C2 A_{PDA} = { "P""w" : P is a PDA that accepts w }

C2 $A_{PDA} \longrightarrow A_{CFG}$

"P""w" → "G""w"

C3. $E_{CFG} = \{$ "G" : G is a CFG with $L(G) = \phi\}$

$S \to Aa$ 从 terminal 和e 开始. 若→右 symbol 被标记, →左边 symbol 也标记.

$A \to Bb$ 看 start symbol 是否被标记

$B \to AC$

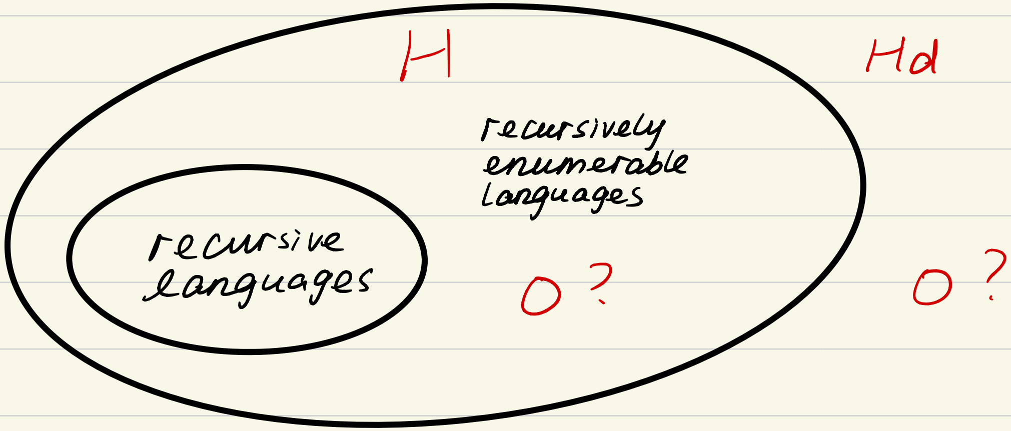
$C \to e$

$C \to a$

$B \to b$

C4. $E_{PDA} = \{$ "P" : P is a PDA with $L(P) = \phi\}$

$C_4 \le C_3$



A set S is countable if it is finite or ∃ bijective $f: S \to N$. uncountable otherwise.

Lemma. A set S is countable ⟺ ∃ injection $f: S \to N$

proof: ⇒ ✓

⟸ ∃ injection $f: S \to N$ (assume S is infinite) otherwise finite → trivial

⇓

label element of S as $S_1, S_2, S_3 \cdots$

so that $f(s_1) < f(s_2) < f(s_3) < \cdots$

$g(S_i) = i$

Corollary: Any subset of a countable set is countable.

Proof: A countable $\qquad$ A' countable

↓ $\qquad\qquad$ ⇑

∃ injection $f: A \to N$ ⇒ ∃ injection $f': A' \to N$

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

proof: $\overset{e.g.}{\Sigma} = \{0,1\}$

$$\begin{array}{ccccl} e, & 0, & 1, & 00, & 01, 10, 11 \cdots \\ \downarrow & \downarrow & \downarrow & \downarrow & \\ 0 & 1 & 2 & 3 & \cdots \end{array}$$

要讲 $\forall s \in \Sigma^*. \exists f(s)$ # strings with $\le |s|$: $2^{|s|}$

Corollary: $\{M : M \text{ is a TM}\}$ is countable. 图灵机可以进行编码

Lemma: Let Σ be some alphabet

Let $\mathcal{L}$ be the set of all the languages over Σ.

$\mathcal{L}$ is uncountable.

且每台TM仅能半判定一个问题

$\Rightarrow$ $\exists$ language is not recursively enumerable.

(TM 可数, 问题不可数)

Proof: suppose $\mathcal{L}$ is countable

$\Downarrow$

$L_1, L_2, L_3, \cdots$

since Σ^* is countable. 所有串

$s_1, s_2, s_3, \cdots\cdots$

构造: $D = \{s_i : s_i \notin L_i\} \in \mathcal{L}$ $\rightarrow$ contradiction

$\forall i,\ s_i \in D \text{ iff } s_i \notin L_i$

$\therefore D \neq L_i$ 即 D与$\overset{\mathcal{L}}{\wedge}$列出的每一个元素都不同

$$\begin{array}{c|cccc} & s_1 & s_2 & s_3 & \cdots \\ L_1 & \overset{0}{1} & 0 & 0 & \\ L_2 & 0 & \overset{0}{1} & 0 & \cdots \\ L_3 & 1 & 0 & \overset{1}{0} & \cdots \\ L_4 & & & & \\ \vdots & & & & \end{array}$$

取反

D: 与Li每一行都不同

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

Theorem: H is recursively enumerable.

universal TM ← U = on input "M" "w"

$L(U) = H$

```
1. run M on w
```

U halt on "M""w" $\Leftrightarrow$ M halts on w

("M""w" $\in H$)

Theorem. H is not recursive. 不可判定

Proof: $H_d = \{``M" : M$ is a TM that does not halt on "M"$\}$

①

If H is recursive, so is H_d

②

H_d is not recursively enumerable.

① If H is recursive $\Rightarrow M_H$ decides H

则 M_d = on input "M"

```
1. run M_H on "M""w" where w = "M"
2. If M_H accepts "M""w"
       reject "M"
3. else
       accept "M"
```

反证

② Assume .... $\Rightarrow \exists D$ semidecides H_d

D on input M $\begin{cases} \text{halt. if "M"} \in H_d \text{ (M does not halt on "M")} \\ \text{not halt. if "M"} \notin H_d \text{ (M halts on "M")} \end{cases}$

Let $M = D$?

D halts on "D" $\Leftrightarrow$ D does not halt on "D"

$\emptyset$

halt

$$\begin{array}{c|ccc} & ``M_1" & ``M_2" & ``M_3" \\ \hline M_1 & 1 & 0 & 0 \\ M_2 & 0 & 1 & 0 \\ M_3 & 1 & 0 & 0 \\ D & 0 & 0 & 1 \end{array}$$

取反

刻画problem难度关系

If $A \leq B$ and A is not recursive, then B is not recursive.

① $A_1 = \{ ``M" : M$ is a TM that halts on $e \}$

$H \leq A_1$

$``M" ``w" \longrightarrow ``M^*"$

保证映射前后答案一样

M halts on w $\Leftrightarrow$ M^* halts on e.

我们要构造M^*使其满足 令 $M^* =$ on input u

1. run M on w

M^* halts on e $\Leftrightarrow$ M^* halts on $\overset{\text{any}}{\text{some}}$ input $\Leftrightarrow$ M halts on w

$f(``M" ``w") = ``M^*"$

② $A_2 := \{ ``M" : M$ is a TM that halts on some inputs.$\}$

$H \leq A_2$

③ $A_3 = \{ ``M" : M$ is a TM that halts on every input$\}$

$H \leq A_3$

同①

④ $A_4 = \{$"M_1" "M_2" : M_1 and M_2 are two TMs with $L(M_1) = L(M_2)\}$

Hint: 从 A_3 归约 ∵ "M" $\longrightarrow$ "M_1" "M_2"

M halts on every input $\Longleftrightarrow$ $L(M_1) = L(M_2)$

M_2 = on input x

1. halt

Let $M_1 = M$, then M halts on every input $\Leftrightarrow \underset{L(M_1)}{L(M)} = \underset{L(M_2)}{\Sigma^*}$

⑤ $R_{TM} = \{$"M" : M is a TM with $L(M)$ is regular$\}$.

要证 $\overline{R_{TM}} = \{$"M" : M is a TM with $L(M)$ is not regular$\}$ 不可判定

$$\begin{tikzcd} H & \overline{R_{TM}} \\ \text{"}M\text{" "}w\text{"} \arrow[r] & M^* \end{tikzcd}$$

M halts on w $\Longleftrightarrow$ $L(M^*)$ is not regular.

M^* = on input x

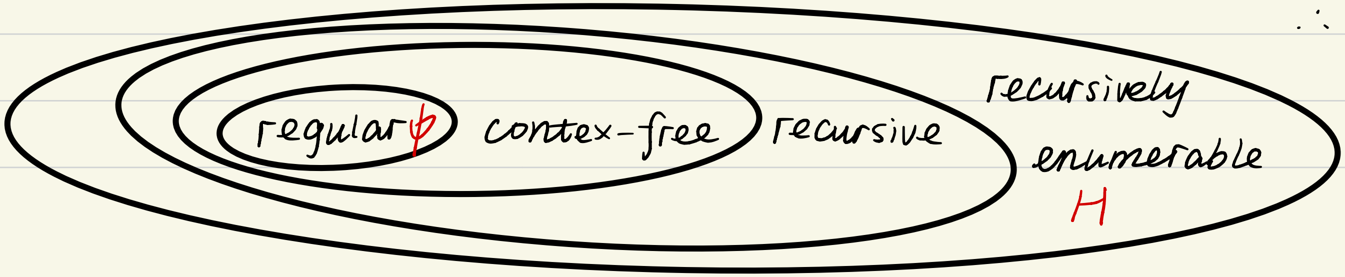
1. run M on w
2. run U on x

$$L(M^*) = \begin{cases} L(U) = H & \text{if } M \text{ halts on } w \\ \phi & \text{if } M \text{ does not halt on } w. \end{cases}$$

∴ $L(M^*)$ is not regular

$\Updownarrow$

M halts on w



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

$H \le \overline{CF_T}$

$L(M^*)$ is not context-free $\Leftrightarrow$ M halts on w.

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

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

$A = \{ \text{"}M\text{"} : M$ is a TM that halts on every input$\}$

半判定

$B = \{ \text{"}M_1\text{"}\text{"}M_2\text{"} : M_1$ and M_2 are two TMs with $L(M_1) = L(M_2)\}$

$A \le B$: 用 B 来解决 A : 若有 M_B. 用其构造 M_A

reduction from A to B.

M_A = on input "M"

on input x
1. halt

1. consider a TM M^* that halts on every input.

2. run M_B on "M""M^*"

3. return the result of M_B.

半判定

$\{ \text{"}M\text{"} \mid M$ is a TM with $L(M)$ having property $P\}$

regular / context-free / recursive / $e \in L(M)$ / $L(M) = \Sigma^*$

$\mathcal{L}(P)$ = the set of recursively enumerable languages satisfying P

$R(P) = \{ \text{"}M\text{"} \mid M$ is a TM with $L(M) \in \mathcal{L}(P)\}$ 不可判定?

if $\mathcal{L}(P) = \phi$ or the set of all recursively enumerable, $R(P)$ is recursive.

Rice's Theorem: If $\mathcal{L}(P)$ is a non-empty proper subset of all recursively enumerable languages,

then $R(P)$ is not recursive.

Proof:

case 1. $\phi \notin \mathcal{L}(P)$ 则 $\exists A \in \mathcal{L}(P)$ 且 $A \neq \phi$.

$\Downarrow$

$\boxed{\exists M_A}$ semidecides A

要证: $\underset{\overset{\uparrow}{M_H}}{H} \leq \underset{\overset{\uparrow}{M_R}}{R(P)}$

M_H = on input "M""w"

1. construct a TM $\underline{M^*}$ = on input x

   (1) run M on w

   (2) run M_A on x

2. run M_R on "M^*"

3. return the result of M^*

则 $L(M^*) = \begin{cases} \overset{\in \mathcal{L}(P)}{L(M_A) = A}, & \text{if } M \text{ halts on } w. \\ \underset{\notin \mathcal{L}(P)}{\phi} & \text{if} \cdots \text{not} \cdots \end{cases}$

我们只要知道 $L(M^*) = ? \Rightarrow M$是否halt on w.

case 2: $\phi \in \mathcal{L}(P)$. 则 $\phi \in \overline{\mathcal{L}(P)}$

Summary

proving recursive
- by def (construct TM)
- $A \leq$ a known recursive language

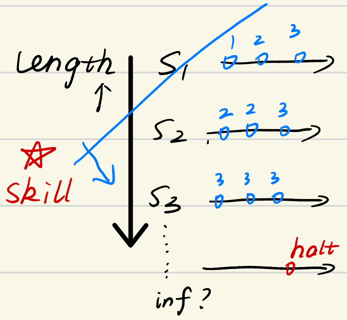
H is not recursive. (Diagonalization)

proving non-recursive : A known non-recursive language $\leq A$.

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

Example

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



M halts on S_j at the k-th step $\Rightarrow \max(k, j)$

只要有halt. 一定能在有限步内找到

```
MA = on input "M"
  for i = 1. 2. 3 ...
    for s = s_1 .... s_i
      run M on s for i steps
      if M halts on s within i steps:
        halt
```

proving not recursively enumerable — A known non-recursively enumerable lang. $\le A$

— theorem.

Theorem: If A and $\bar{A}$ are recursively enum. then A is recursive.

M_1 + M_2 $\Rightarrow$ M_3 (decides A?)

```
M_3 = on input x
  1. run M_1 and M_2 on x in parallel
  2. if M_1 halts
  3.     accept x
  4. if M_2 halts
  5.     reject x
```

原因: $x \in L(A)$ or $x \in L(\bar{A})$

Example.

H is recursively enum. $\Rightarrow$ $\bar{H}$ is not recursively enum.

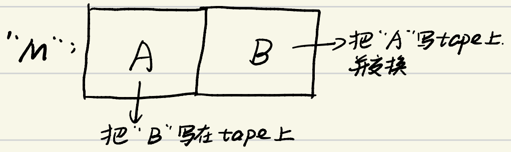
H is not recursive

Closure property

	recursive	recursively enum.
$\cup$	✓	✓
$\cap$	✓	✓
$-$	✓	X
$\circ$	✓	✓
$*$	✓	✓

Example. write a program that print itself.

M write "M" on its tape.



A: write "B" on the tape.

B: write "A" on the tape, and swap it with "B"

要让 B 的定义不依赖于 A

循环定义？

function $q(w) =$ "M_w" where M_w is a TM that prints w on its tape

q is computable (∵ Given w, M_w = on input x.

1. write w on the tape

2. halt.)

→ 定义不依赖 A，但根据 A的输出作为输入（推出"A"）

B := on input w.

1. compute $q(w)$ A的输出 $q(\text{"B"}) = \text{"A"}$

2. write $\underbrace{q(w)}_{\text{"A"}} \cdot \underbrace{w}_{\text{"B"}}$ on its tape

先运行 A $\boxed{\text{"B"}\quad\quad}$ 此时输入是"B"，再运行 B（写"A" "B"）

Recursion Theorem

for any TM T, there is a TM R such that for any string w, the compution of R on w is equivalent to that of T on "R" w.

↑ R 拿到了自己encoding

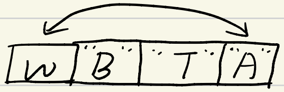
作用: M = on input x

1. obtain "M" ← legal（可以在TM里有这样操作）

proof sketch.

"R": $\boxed{A}\boxed{B}\boxed{T}$

A: print "B" "T"



B: print "A" and reorder "A""B""T" : "R"

Example

可用来证 H non-recursive.   Assume H is recursive, ∃ M_H decides H.

```
R = on input w
    1. obtain "R"
    2. run M_H on "R""w"
    3. if M_H accepts "R"w
    4.     looping
    5. else M_H rejects "R"w
    6.     halt  → contradiction
```

拿到自己code后 → 1.
先用H做判定
再反过来halt

Enumerator:

从空输入开始. 到 q 时记下 w

We say a TM enumerate a language L, if for some state q,

?

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

output state　　output w

↓ Turing enumerable

Theorem. A language L is Turing enumerable <=> it is recursively enum.

proof:　finite => trivial

Assume L is infinite

=> ∃M enumerate L　　goal: M' semidecides L

M' = on input x.

```
1. run M to enumerate L
2. every time M outputs a string w
3.     if w == x:
4.         halt
```

<= ∃M. semidecides L. goal: M' enumerate L

∵ 只能半判定 => 可能 loop forever

$$\begin{array}{llll} s_1 & 0^1 & 0^2 & 0^3 \\ s_2 & 0^2 & 0^2 & 0^3 \\ s_3 & 0^3 & 0^3 & 0^3 \\ \vdots & & & \end{array}$$

output s_i if M halts

同样串可能重复. 乱序输出 ✓

按字典序枚举

Let M be a TM that decides L, we say M **lexicographically enumerates** L if whenever $(q, \triangleright\sqcup w_1) \vdash_M^* (q, \triangleright\sqcup w_2)$, we have w_2 is after w_1 in lexicograhical order.

Theorem. L is lexicographically enumerable ⟺ it is recursive.

证明类似：

$\Rightarrow$ $\exists M$ enumerate L goal: M' decides L

Lexicographically

M' = on input x.

```
1. run M to enumerate L   (Lexicograhpically)
2. every time M outputs a string w  → only order ≤ x
                                      (字典序 > x ⟹ reject)
3. if w == x:
4.       accept.
```

$\Leftarrow$ $\exists M$. decides L.

S_1 —————— 一行行枚举即可 (decide, 必会停机)

S_2 ——————

S_3 ——————

⋮

numerical function

$f: N^k \to N \quad (k \geq 0)$

computable

A TM M compute $f: N^k \to N$ if for any $n_1 \cdots n_k \in N$, $M(bin(n_1), bin(n_2) \cdots bin(n_k))$
$= bin(f(n_1, n_2 \cdots n_k))$

basic functions

(1) **zero function**

$zero(n_1, n_2 \cdots n_k) = 0$ for any $n_1, \cdots n_k$

(2) **identity**

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

(3) **successor function**

$succ(n) = n+1$

} computable

两种操作：

(1) **composition**: $g: N \to N.$ $h: N \to N \Rightarrow f(x) = g(h(x))$

多元： $g: N^k \to N.$ $h_1 \cdots h_k: N^l \to N \Rightarrow \underset{\text{composition of } g \text{ and } \cdots}{\underset{\downarrow}{f}}(n_1, \cdots n_l) = g(h_1(n_1 \cdots n_l), h_2(n_1 \cdots n_l), \cdots h_k(n_1 \cdots n_l))$

(2) **recursive definition**

$$f(n) = n! \overset{\text{可用}}{\Rightarrow} \begin{cases} f(0) = 1 & \text{定义} \\ f(n+1) = f(n) \cdot (n+1) = h(f(n), n) \end{cases}$$

多元： $g: N^k \to N$, $h: N^{k+2} \to N \Rightarrow f: N^{k+1} \to N.$

$$\begin{cases} f(n_1, \cdots n_k, 0) = g(n_1, \cdots n_k) \\ f(n_1, \cdots n_k, m+1) = h(n_1, \cdots n_k, m, \overset{\text{前一次的值}}{\overset{\uparrow}{f(n_1, \cdots n_k, m)}}) \end{cases}$$

Def: basic functions + $\begin{cases} \text{composition} \\ \text{recursive def} \end{cases}$ $\longrightarrow$ **primitive recursive function**

Corollary: primitive recursive function + $\begin{cases} \text{composition} \\ \text{recursive def} \end{cases}$ = primitive recursive functions

Example.

(1) $plus2(n) = n+2$

$$succ(succ(n))$$

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

$$\begin{cases} plus(m,0) = m \\ plus(m,n+1) = \underline{succs(plus(m,n))} \end{cases}$$

严格应为关于 $m.n.plus(m,n)$ 函数

$$succ(id_{3,3}(m,n,plus(m,n)))$$

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

PR => PR

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

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

(5) $f(n_1, \cdots n_k) = c$

$$\underset{\text{做 c 次}}{succ}(zero(n_1, \cdots n_k))$$

(6) sgn function

$$\begin{cases} sgn(0) = 0 \\ sgn(n+1) = 1 \quad // \ h(n, sgn(n)) = 1 \end{cases}$$

(7) predecessor function

$$pred(n) = \begin{cases} n-1 & \text{if } n>0 \\ 0 & \text{if } n=0 \end{cases} \quad \Rightarrow \quad \begin{cases} pred(0) = 0 \\ pred(n+1) = n = id_{2,1}(n, pred(n)) \end{cases}$$

(8) $m \sim n = \max\{m-n, 0\}$

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

$+ \quad - \quad \times \quad \Rightarrow$ primitive recursive

$\Downarrow$

if $f.\ g$ are p.r. so are $f+g,\ f-g,\ f\cdot g$

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

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

$\rightarrow$ predicates

If two predicates p and q are p.r., so are $\neg p.\ p\wedge q.\ p\vee q.$

$\because \neg p = 1-p,\ p\wedge q = p\cdot q,\ p\vee q = \text{positive}(p+q)$

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

(12) $\text{eq}(m,n) = \begin{cases} 1 & \text{if } m=n \\ 0 & \text{if } m\neq n \end{cases}$ $\qquad \text{geq}(m,n) \wedge \text{geq}(n,m)$

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

If $g.\ h.\ p$ are p.r., so is $f.$ ($f = p\cdot g + (1 \sim p)\cdot h$

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

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

(14) $div(m,n) = \lfloor m/n \rfloor$ // $\overset{\text{假定}}{n \neq 0}$

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

(15) $digit(m,n,p) = a_{m-1}$

$n = a_k p^k + \cdots + a_{m-1}p^{m-1}, a_1 p^1 + a_0$ 将n用p进制表示,返回第m位

$div(rem(n,p^m), p^{m-1})$

(16) p: primitive recursive predicates

bounded disjunction. $g_p(n) = \begin{cases} 1 & \text{if } \exists 1 \le i \le n, p(i)=1 \\ 0 & \end{cases}$

1~n中是否有i使p为真

g_p也是p.r.

$g_p(n) = p(0) \cup \cdots p(n)$

$= positive(sum_p(n))$

bounded conjunction $h_p(n) = \begin{cases} 1 & \text{if } \forall 1 \le i \le n, p(i)=1 \\ 0 & \text{otherwise} \end{cases}$

1~n中是否∀i使p为真

(17) $sum_f(m,n) = \sum_{k=0}^{n} f(m,k)$ 可证: if f is p.r., so is sum_f.

$sum_f(m,n) = f(m,0)+\cdots f(m,n)$ // sum of $\underset{\substack{\downarrow \\ n\text{不是一个常数!}}}{n+1}$ p.r. func.? ✗

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

$mult_p(m,n) = \prod_{k=0}^{n} f(m,k) \Rightarrow h_p(n)$ is also p.r.

Lemma. All p.r. func. are computable.

proof: basic functions are computable, $\left\{\begin{array}{l}\text{composition}\\ \text{recursive def}\end{array}\right.$ preserve computability.

反之, All computable func. are primitive recursive? ✗

all p.r. func $\Rightarrow$ expression 组合,递归类似正则表达式 ? computable

$\Downarrow$

enumerate all the expression

$\Downarrow$

enumerate all unary p.r. func. $g_1, g_2 \cdots g_n$

M = on input n

1. enumerate $g_1, g_2 \cdots$ to get g_n
2. compute $g_n(n)$
3. return $g_n(n)+1$

g^* is not p.r. $\longleftarrow$ compute g^*, 但 $g^* \neq g_n$ $\forall n$

$(\because g^*(n) = g_n(n)+1 \neq g_n(n))$

basic functions + $\left\{\begin{array}{l}\text{composition}\\ \text{recursive def}\\ \text{minimalization of minimalizable functions}\end{array}\right.$

μ-recursive <=> computable

min 操作

$g(n_1, \cdots n_k, n_{k+1})$

$$f(n_1, \cdots n_k) = \begin{cases} \text{minimum } m \text{ with } g(n_1, \cdots n_k, m) = 1 & \text{if exists} \\ 0. & \text{otherwise} \end{cases}$$

最小 m. 使得 $g=1$

f is a minimalization of g, 记作 $\mu m[g(n_1, \cdots n_k, m) = 1]$

Example :

$Log(m,n) = \lceil \log_{m+2}(n+1) \rceil$ // $\min\{p : (m+2)^p \geq n+1\}$

$\quad\quad$ " $\mu p[geq((m+2)^p, n+1) = 1]$

A function $\overset{g}{\wedge}$ is **minimalizable** if

(1) g is computable

(2) for $\forall n_1 \cdots n_k, \exists m \geq 0$ s.t. $g(n_1, n_2 \cdots n_k, m) = 1$

一个个试总会停机.

minimalization of g is computable if g is minimalizable.

Given a computable function g, is g mininalizable ? → undecidable

μ- recursive = basic functions + $\begin{cases} \text{composition} \\ \text{recursively def.} \\ \text{minimalization of minimalizable functions} \end{cases}$

Theorem: A numerical function f is μ-recursive $\Leftrightarrow$ it is computable.

Proof: $\Rightarrow$ trivial

$\Leftarrow$ f. $\exists M$ computes f. $\quad (s, D\underline{U}n) \vdash_M (q_1, DU_1a_1v_1) \cdots\cdots \vdash_M (h, D\underline{U}f(n))$

可写作:

$DUSnDU_1a_1q_1v_1D \cdots\cdots DUhf(n)$ 串 可以表示为一个数 (base-b integer)

$\Sigma \cup K \to \{0, \cdots b-1\}$ ($b = |\Sigma \cup K|$)

每一层可看作数值函数

$$\begin{tikzcd}
n \arrow[d, "h_1"] \\
DUSn \arrow[d, "h_2"] \\
DUSnDU_1a_1q_1v_1D \cdots\cdots DUhf(n) \arrow[d, "h_3"] \\
DUhf(n) \arrow[r, "h_4"] & f(n)
\end{tikzcd}$$

h_2: $\mu m[\overset{p.r.}{iscomp(DUSn, m)} \wedge \overset{p.r.}{ishalted(m)}]$

从初始状态到 m

h_4 …… 类似 h_3

$h_1(n) = DUS \cdot b^{\log_b n} + n$

找到 D 并取出

h_3 : $\mu_k[digit(k, n, b) == D]$ 得 k^*

n 用 b 进制下的第 k 位

$rem(n, b^{k^*+1})$

Grammar (Conrestricted grammar)

CFG : $A \to u$, $B \to v$, ... 可以与上下文有关 e.g. $uAv \to w$

Def. A grammar is a 4-tuple $G = (V, \Sigma, S, R)$

- V is an alphabet
- $\Sigma \subseteq V$ is the set of terminals
- $S \in V - \Sigma$: start symbol
- R : a finite set of $(\underbrace{V^*}_{} \underbrace{(V-\Sigma)}_{\text{nonterminal}} \underbrace{V^*}_{}) \times V^*$ context

$\Rightarrow_G$, $\Rightarrow_G^*$ G generates a string $w \in \Sigma^*$ if $S \Rightarrow_G^* w$. $L(G) = \{w \in \Sigma^* : G \text{ generates } w\}$

Example.

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

$S \to ABCS$ ABCABC....

$BA \to AB$, $CA \to AC$, $CB \to BC$ A...AB...BC...CS

$S \to T_c$ $CT_c \to T_c C$ $BT_c \to BT_b$ 从右往左扫

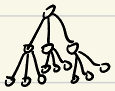
$BT_b \to T_b b$ $AT_b \to AT_a$

$AT_a \to T_a a$ $T_a \to e$

Theorem. A language is generated by some grammar $\Leftrightarrow$ it is semidecides by some TM.

Proof. $G \Rightarrow$ TM M to semidecides $L(G)$.

given $w \in \Sigma^*$, is $S \Rightarrow_G^* w$? (第 i 步有 $|R|^i$ 种 . 若找到 $\Rightarrow$ halt)



$\Leftarrow$ Given M. construct G to generate $L(M)$ ($S \Rightarrow_G^* w \Leftrightarrow w \in L(M)$)

对于 $w \in L(M)$: $(s, \rhd \sqcup w) \vdash_M (q_1, \rhd \sqcup u_1 \underline{a_1} v_1) \cdots\cdots \vdash_M (h, \rhd \sqcup \underline{\sqcup})$

$\rhd \sqcup S w \overset{\text{末尾}}{\lhd} \vdash_M \rhd \sqcup u_1 a_1 q_1 v_1 \lhd \cdots\cdots \vdash_M \rhd \sqcup h \lhd$

用 state 标记下划线位置

$\underline{S \Rightarrow \triangleright \sqcup h \triangleleft} \Rightarrow \underline{\cdots\cdots \Rightarrow \triangleright u_1 a_1 q_1 v_1} \triangleleft \Rightarrow \underline{\triangleright \sqcup S w \triangleleft \Rightarrow w}$

① $S \rightarrow \triangleright \sqcup h \triangleleft$? ③ ② $\triangleright \sqcup S \rightarrow e$

$\triangleleft \rightarrow e$

③

写: if $\delta(q,a)=(p,b)$ for some $a,b \in \Sigma$

$uaqv\triangleleft \quad \vdash_M \quad ubpv\triangleleft \qquad bp \rightarrow aq$

右移, if $\delta(q,a)=(p,\rightarrow)$

$uaqbv\triangleleft \quad \vdash_M \quad uabpv\triangleleft \qquad abp \rightarrow aqb$ for $b \in \Sigma$

若 b.v 为空格 if $b=\sqcup$. $v=e$ $\quad uaq\triangleleft \Rightarrow ua\sqcup p\triangleleft \qquad a\sqcup p \rightarrow aq$

左移 ...

$\therefore L(G) = L(M)$

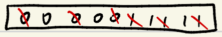
复杂度.

decidable vs. undecidable

resource: time, space

$A=\{0^k 1^k : k \geq 0\}$ Given w, $w \in A$? 要走多少步 $n=|w|$

用单带TM情况下 扫$\frac{n}{2}$次, 每次走$O(n)$步 $\Rightarrow O(n^2)$ 每次扫消一个0+1



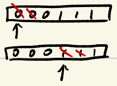
$\log_2 n \cdot O(n) = O(n\log n)$ 每次扫消掉一半0和1 (隔一格消)

Def: Let M be a deterministic TM that halts on every input. The running time of M is a function $f: \underset{\text{input length}}{N} \rightarrow \underset{\#\text{step}}{N}$ where for any input of length n. M halts within $f(n)$ steps 最坏情况. n

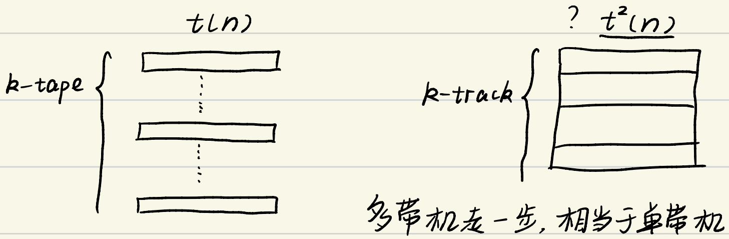
$DTIME(\overset{?}{t(n)})=\{A: A$ is decided by some standard TM within $O(t(n))$ running time.$\}$

依赖于特定TM (单带)

$A=\{0^k 1^k : k \geq 0\}$ 2-tape: $O(N)$



Multi-tape TM   standard TM



改变每个 track (先扫一遍确定读写头)

多带机走一步，相当于单带机扫两遍 (写 t(n) 格 ⇒ O(t(n)))

└ 纸带最多 t(n) 格

DTM 的变种 deterministic variant   standard

t(n)   poly(t(n))

Cobham - Edmonds Thesis:

Any "reasonable" and "general" deterministic model of computation is polynomially related.

复杂类 **P** is the set of languages that are decided by some deterministic TM whose running time is poly(n).

$$P = \bigcup_{k \geq 0} DTIME(n^k)$$

Theorem: Every context-free language is in P.

$$\begin{cases} S \Rightarrow e \\ A \Rightarrow BC \\ A \Rightarrow a \end{cases}$$

proof: for any context-free language A, ∃ CFG $G = (V, \Sigma, S, R)$ in CNF generates A.

Given w, enumerate all derivations of length $2|w|-1$. $R^{2|w|-1}$ 可判定，但无法证明多项式时间

可以用 DP: Dynamical Programming $w = a_1 \cdots a_n$, $S \Rightarrow^* w$?

可以生成子串的 nonterminals 的集合.

subproblem: for $1 \leq i \leq j \leq n$, define $T[i, j] = \{A \in V - \Sigma : A \Rightarrow^* a_i a_{i+1} \cdots a_j\}$

Goal: $S \in T[1, n]$?

base case: for $1 \leq i \leq n$: $T[i, i] = \{A \in V - \Sigma : A \to a_i\}$

recurrence: $1 \leq i < j \leq n$: $T[i, j] = \bigcup_{k=i}^{j-1} \{A \to BC : \underbrace{B \Rightarrow^* a_i \cdots a_k}_{B \in T[i,k]} \wedge \underbrace{C \Rightarrow^* a_{k+1} \cdots a_j}_{C \in T[k+1, j]}\}$

$\underbrace{a_i a_{i+1} \cdots a_k}_{\Uparrow^* B}$ $\underbrace{a_{k+1} \cdots a_j}_{\Uparrow^* C}$ 且 $A \to BC$

subproblems: $\frac{n^2}{2}$ cost per subproblem : $n \cdot |R|$ total: $O(n^3 \underline{|R|})$ 与输入无关，是规则数

$f(s) = |s|$

M = on input $\underline{F}$ (boolen formular)

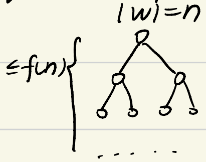
1. non-deterministically generate an assignment of boolen variable.
2. If F is satisfied, accept.
3. otherwise reject.

SAT ∈ P ? unknown

satisfiability $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4 \vee x_5)$ 用NTM可以在多项式时间内✓

Def. Let M be a non-deterministic TM that for any input every branch of M halts with k steps where k depends only on the input. 每个分支都停机

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



NP is the set of all languages that can be decided by some NTMs in polynomial time.
(non-deterministically polynomial)

(1) for any F that is satisfiable, ∃ certificate y, 在y的帮助下验证F

evaluate F under y if F is satisfied. accept.

Def: A language A is poly veriable if there is a polynomial-time DTM V (verifier) such that for any $x \in \Sigma^*$,

(1) if $x \in A$, $\exists y$ with $|y| \le poly(|x|)$, V accept "x""y"

(2) if $x \notin A$, $\forall y$ ·················· rejects ····

Example A= SAT, x=boolen formular, y= a truth assignment that satisfies x.

V= on input "x""y"

1. evaluate x under y
2. if x is satisfied by y
   accepts "x""y"
3. else
   rejects "x""y"

Theorem: A language A is polynomially verifiable <=> it is in NP.

Proof : => ∃ polynomial-time verifier V

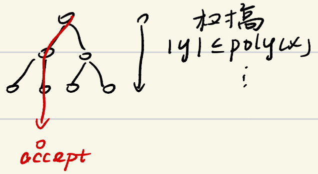
to construct a NTM M decides A in polynomial time.

```
M = on input x
  1. non-deterministically generate a certificate y with $y \le poly(|x|)$
  2. run V on "x" "y"
  3. if V accepts "x" "y"
        accept
  4. else
        reject
```

<= ∃ NTM M decides A in poly time.

to construct poly-time verifier V for A.

certificate y = the branch that accepts x 每个分支如何选择



```
V = on input "x" "y"
  1. run M on x deterministically under guidance of y
  2. if M accepts x
        accept "x" "y"
  3. else
        reject "x" "y"
```

P. vs NP. ($P \subseteq NP$) — a NTM is a DTM

— $A \in P$: DTM D, 这样不需要 certificate

```
V= on input "x" "y"
  1. run D on "x"
```

$P = NP$? unknown

直觉上: $P \subsetneq NP$

Cook & Levin :

an NP-complete problem is in P <=> P = NP

NP - Complete : hardest in NP

a reduction f from A to B (记作 $A \le B$)

$+$

f can be computed by some DTM in poly(n) time.

$\Downarrow$

$A \le_p B$ "A在多项式时间内可被归约到B". 用来判断难度

Theorem. If $A \le_p B$, $B \in P$ then $A \in P$

$x \longrightarrow f(x) \longrightarrow$ decide $f(x) \in B$?

$\text{poly}(|x|) + \text{poly}(|f(x)|)$ and $|f(x)| \le \text{poly}(|x|)$

└ 写输出的长度

Example

SAT $(x_1 \vee x_2 \vee x_3 \cdots\cdots) \wedge (x_1 \vee x_2) \wedge \cdots -$

3SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge \cdots\cdots$

$3SAT \le_p SAT$ $f(x) = x$

$SAT \le_p 3SAT$

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

至少一项为真

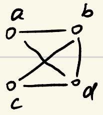
$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Rightarrow (\underline{x_1 \vee x_2} \vee y) \wedge (\underline{x_3 \vee x_4 \vee x_5} \vee \bar{y}) \Rightarrow \cdots\cdots$

$3 \cdot (k-2) = O(k)$

└ 括号内长度

Clique 团问题

$G = (V, E)$



A clique of G is a subset $V' \subseteq V$ such that for any $u, v \in V'$ and $u \ne v$, $(u, v) \in E$

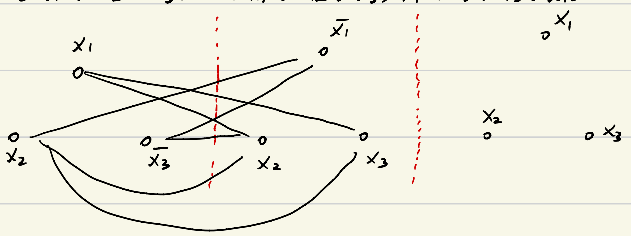
e.g. $\{a, b\}$ ✓ $\{a, b, d\}$ ✓ $\{a, b, c, d\}$ ✗

CLIQUE = { "G" "k" : G has a clique of at least k }

要证: $3\text{-}SAT \le_p CLIQUE$

$F \Rightarrow G \quad k$

$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_1)$ m clauses



then $k = m$

要证：F is satisfiable <=> G has a clique of size at least m.

=> ✓

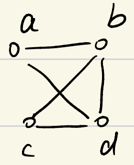
<= 团至多包含一个组中的一个点 => k个点的团中k个组均有点选中

G: # nodes: 3m, # edges: $\le 9m^2$

Vertex Cover

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

e.g.



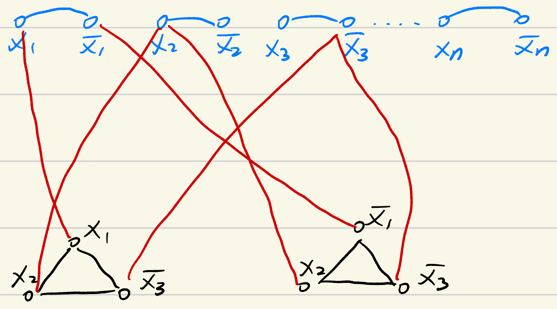
{a, b, d} ✓ {a, d} ✗

$VC = \{ "G" "k" : G$ has a vertex cover of size at most $k\}$

要证：3-SAT $\le_p$ VC

F (n variables, m clauses) => "G" "k"

2n nodes



n 条蓝边

3m 条红边

3m nodes

3m 条黑边

$(x_1 \vee x_2 \vee \bar{x}_3)$ $(\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

F is satisfiable $\Leftrightarrow$ G has a vertex cover of size $n+2m$

$\Rightarrow$?

$\Leftarrow$

G: # nodes : $2n+3m$

edges : $n+3m+\underset{\le 6m\cdot n}{\text{红}}$

Def. A language L is NP-complete if

(1) $L \in NP$

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

The Cook-Levin Theorem: SAT is NP-complete.

Proof: Let A be an arbitrary language in NP.

$$A \le_p SAT$$

$$x \longrightarrow F$$

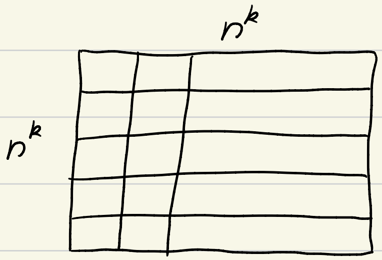
$$x \in A \Leftrightarrow F \text{ is satisfiable}$$

$\exists$ NTM N decides A in n^k time $a_1 \cdots a_n \in A$.

$\Leftrightarrow \exists (s, D\underline{u} a_1 \cdots a_n) \vdash_M (q_1, Du_1 \underline{a} v_1) \vdash_M \cdots \vdash_M (y, Du\underline{a}v)$

$\Leftrightarrow \exists \underbrace{Dusa_1 \cdots a_n \vdash_M Du_1 a q_1 v_1 \vdash_M \cdots\cdots \vdash_M Duayv}_{\le n^k \text{ configurations of length } n^k.}$

?



for $1 \le i \le n^k,\ 1 \le j \le n^k,\ c \in K \cup \Sigma$.

x_{ijc} for each i and j $\sum_{c \in K \cup \Sigma} x_{ijc} \ge 1 \Leftrightarrow \bigvee_{c \in K \cup \Sigma} x_{ijc}$

for each i and j. $\bigwedge_{c \ne c'} \overline{x_{ijc} \wedge x_{ijc'}} = \bigwedge_{c \ne c'} (\overline{x_{ijc}} \vee \overline{x_{ijc'}})$ 一格只能至多放一个symbol

让初始行统一 $x_{11D} = 1 \wedge x_{12u} = 1 \wedge x_{13s} = 1 \wedge \cdots$

要保证行按顺序(第2行接着第1行操作)

c_1	c_2	c_3
c_1	c_2	c_3

c_1	c_2	c_3
q	c_2	c_3

c_1	q	c_2
c_1'	p	c_2'

....

#legal 2×3 rectangle $\le |K\cup\Sigma|^6$

Theorem: If A is NP-complete, and

(1) $B \in NP$

(2) $A \le_p B$

then B is NP-complete.

Proof: $\forall L' \in NP.\ L \le_p A$ 又∵ $A \le_p B \Rightarrow L \le_p B$ 归约传递性.

空间: Let M be a DTM. We say that M runs in $f(n)$ space if for input of length n, M uses at most $f(n)$ tape cells.

假定 $f(n) \geq n$

one-tape DTM (实际上变种TM只相差一个常数)

Let N be a NTM. We say that N runs in $f(n)$ space if for any input length n, every branch uses at most $f(n)$ tape cells.

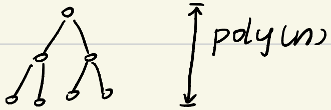
PSPACE = {A | A can be decided by some DTM in poly(n) space}

NPSPACE = {A | .... - - ----.... NTM ....-- .------- }

P $\subseteq$ PSPACE   If a DTM runs in $f(n)$ time ($f(n) \geq n$)

then it runs in $f(n)$ space.   “走这么多格” ?

NP $\subseteq$ NPSPACE



空间可以复用, 只用考虑某一分支的空间 $\Rightarrow$ poly(n)

还要记分支的选择情况(每个结点一个) $\Rightarrow$ poly(n) configuration 不会重复 (否则有loop)

↓

最多步数取决于 configuration 个数

状态 $|K|$

位置 $f(n)$

纸带上写的 $|\Sigma|^{f(n)}$

If a DTM runs in $f(n)$ space and it halts on all inputs, then it runs in $\underline{|K| \cdot f(n) \cdot |\Sigma|^{f(n)}}$ time.

↓

$c^{f(n)}$

$\Rightarrow$ PSPACE $\subseteq$ EXP $= \{A \mid A$ can be decided by some DTM in $2^{poly(n)}$ time$\}$

$P \subseteq NP \subseteq PSPACE \subseteq EXP$ NPSPACE?

$\subsetneq$? $\subsetneq$? $\subsetneq$? unknown $\Rightarrow$ 但可以证明 $P \subsetneq EXP$. 则必有一个真包含.

Theorem: NPSPACE = PSPACE

Savitch's theorem: If A is decided by some NTM in $f(n)$ space where $f(n) \geq n$, then it is decided by some DTM in $O(f^2(n))$ space.

错误证明: 要记每一步的选择 $f(n) + c^{f(n)}$ total space

递归

proof: $C_{init} \rightsquigarrow C_{accept}$ // all configurations use $f(n)$ space.

within $2^{f(n)}$ steps

$\Updownarrow$

$\exists C'$, $C_{init} \rightsquigarrow C'$ within $2^{f(n)-1}$ steps

未知, 但 $2^{f(n)}$ choices

↓

枚举 (对空间✓)

$C' \rightsquigarrow C_{accept}$ within $2^{f(n)-1}$ steps.

```
Y = on input C1. C2. t
1.  if t==1
2.      if C1==C2 or C1 ⊢M C2
3.          accept
4.      else
5.          reject
```

只用存 C_1. C_2

$S(1) = O(f(n))$

↱ Line 7.8 可以复用

$s(t) = O(f(n)) + s(\frac{t}{2})$

$\Rightarrow s(t) = O(f(n)\cdot \log t)$

```
6.  for all configurations $c'$ using $\le f(n)$ space
7.      run $Y$ on $c_1, c', \frac{t}{2}$
8.      run $Y$ on $c', c_2, \frac{t}{2}$
9.      If both accept,
10.             accept,
11.  reject
```

run Y on $C_{init}, C_{accept}, 2^{f(n)}$

$O(f(n)\cdot \log_2 2^{f(n)}) = O(f^2(n))$

Hierarchy Theorem

space : for any $f: N \longrightarrow N$ (satisfying technical conditions)

there is a language A such that

(1) A can be decided by some DTM in $O(f(n))$ space

(2) A cannot . . . . $\overset{\text{严格低阶}}{o(f(n))}$ space

即空间↑, 就可以判定^其他^语言

Proof: construct a DTM D

(1) D decides some languages A in $O(f(n))$ space

(2) for any DTM M that runs in $o(f(n))$ space,

D and M differs on at least one input.

$o(f(n))$ space TM :

	"M_1"	"M_2"	"M_3"
M_1	1		
M_2		-1	
M_3			0
⋮			
D	-1	+1	-1

会停机, 且 space < $f(n)$

可以保证性质 2.

```
D = on input "M"
    1. let $n = |"M"|$                 technical conditions:
                                           $f(n)$ can be computed in $f(n)$ space.
$o(f(n))$    2. compute $f(n)$
    3. run M on "M"  for $c^{f(n)}$ steps → 不保证停机
        3.1 if M does not halts in $c^{f(n)}$ steps, reject
$f(n)$ ?     3.2 if M ever uses more than $f(n)$ space, reject
                                               ↓
    4. if M accept "M"                     不在 $o(f(n))$表中
    5.        reject
    6. if M reject "M"
    7.        accept
```

TIME.

for any $f: N \to N$ satisfying some technical conditions,

there is a language A such that

(1) A can be decided by some DTM in $o(f(n))$ time.

(2) cannot - - - - - - - - - - - - - - - $o(\frac{f(n)}{\log f(n)})$ time

较多时间要提高 $\log f(n)$ 才能有用

proof: D (1) decides some language A in $o(f(n))$ time

(2) for any DTM M that runs in $o(\frac{f(n)}{\log f(n)})$ time

D and M differs on at least one step ("M")

```
D = on input "M"
    1. let $n = |"M"|$            technical condition
                                     $f(n)$ can be computed in $o(f(n))$ time.
    2. compute $f(n)$
    3. run M on "M" for $\frac{f(n)}{\log(f(n))}$ steps
                          └→ 要维护一个 counter : 最后有 $\log_2 f(n)$ 长.
                                                每次 +1. 需要 $\log f(n)$ steps
                                                # steps : $\log f(n) \cdot \frac{f(n)}{\log f(n)} = f(n)$
    4. if ⋯
       ⋮
```

$\Rightarrow P \subsetneq EXP$ （由上定理可知）