Turing Machine

§ 3-18th

No.

Date

Push-down Automaton: Recap:

* Read from Tape
* Move to right
* write the stack
* Entering final state
  not imply end
* non-deterministic

Turing Machine:

* Read from Tape
* Move to right or left
* write to tape
* Entering halting state $\begin{cases} \text{yes acc} \\ \text{no non-acc.} \end{cases}$
  imply end

* deterministic

conventions

\rhd (triangle)

Tap: * ▷ at left end ▷ not at any other states

* ⊔ blank ($\neq e$)

R/W head:

* initial position: usually ▷σσσ......

* Never moves off the left end of tape

Operation: At each step, read current symbol

* move to the next state
* the head: Write Symbol at current squre
  or move left or right

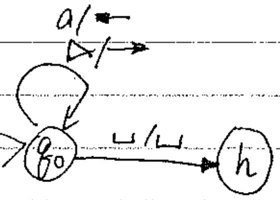
Configuration:

$(q_0, \triangleright \underline{a}\, a) \vdash (q_1, \triangleright \underline{\sqcup}, a) \vdash (q_0, \triangleright \sqcup \underline{a}) \vdash (q_1, \triangleright \sqcup \underline{\sqcup})$

$\vdash (q_0, \triangleright \sqcup \sqcup \underline{\sqcup}) \vdash (h, \triangleright \sqcup \sqcup \underline{\sqcup})$

$(q_0 \triangleright \sqcup aa)$ $\cancel{(q_0, \triangleright aa \sqcup \sqcup)}$
$(q_0, \triangleright a \sqcup a)$

Eg2:



State of ^{a} machines

$(q_0, \triangleright \underline{\sqcup}\, aa) \vdash (h, \triangleright \underline{\sqcup}\, aa)$

$(q_0, \triangleright \underline{a} a) \vdash (q_0, \underline{\triangleright} aa) \vdash (q_0, \triangleright \underline{a} a) \vdash \ldots\ldots$

State Diagram
Rule Table

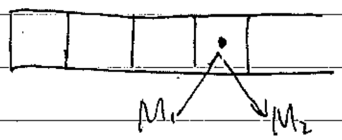
§ Graphical Notation for Turing Machine.

Basic Machines.

$\sigma \in \Sigma - \{\triangleright\}$

M_σ Write σ } short hand σ

$M_\rightarrow$: Move to the right R
$M_\leftarrow$: Move to the left L



Math Def:

K: set of states

Σ: $\{\rhd, \sqcup\} \in \Sigma$

$\{\rightarrow, \leftarrow\} \notin \Sigma$

$s \in K$: initial state

H: set of Haulting states,

Transition

$$\delta(q, a) = (p, b)$$

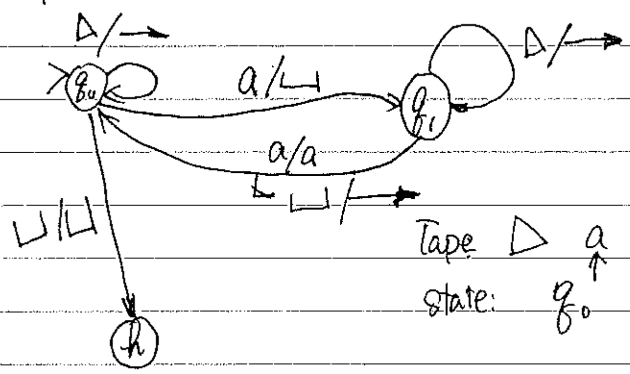
q: current state; a: current symbol; p: next state; b: action $\begin{cases} \text{write } b: & b \in \Sigma \\ \text{move} & b = \leftarrow, \rightarrow \end{cases}$

$$\delta : K \times \Sigma \rightarrow K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$$

$*$ $\delta(q, \rhd) = (p, \rightarrow)$ must be in every TM.

$*$ $\delta(q, a) = (p, b)$ $\quad b \neq \rhd$

Example: $M = \{K, \Sigma, \delta, s, \{h\}\}$



Tape $\rhd\ \underset{\uparrow}{a}\ a$ $\qquad \rhd \underset{\uparrow}{\sqcup} a$ $\qquad \rhd \sqcup \underset{\uparrow}{a}$

state: q_0 $\qquad q_1$ $\qquad q_0$

Tape: $\rhd \sqcup \underset{\uparrow}{\sqcup}$ $\qquad \rhd \sqcup \sqcup \underset{\uparrow}{\sqcup}$ $\qquad \rhd \sqcup \sqcup \underset{\uparrow}{\sqcup}$

State: q_1 $\qquad q_0$ $\qquad h$

short hand

$>R \rightarrow a$ $\qquad$ R_a $\quad$ move right first & write a

$>R \rightarrow R$ $\qquad$ R^2 (L^2) move two steps to the righ

$>R \circlearrowleft^{\sqcup}$ $\qquad$ $R_{\overline{\sqcup}}$; $(L_{\overline{\sqcup}})$

↳ move right untill non-blank $\sqcup$

$>R \circlearrowleft^{\sigma \neq \sqcup}$ $\qquad$ $R_{\sqcup}$ $(L_{\sqcup})$

move right untill a blank

E.g:

M:

$> R_{\overline{\sqcup}} \longrightarrow L\sigma$

$\triangleright \underline{\sqcup} \sqcup \sqcup a$

M:

$R_{\overline{\sqcup}}$: $\triangleright \sqcup \sqcup \sqcup \underline{a}$ $\quad$ $\sigma = a$

$L\sigma$: $\triangleright \sqcup \sqcup \underline{a} a$.

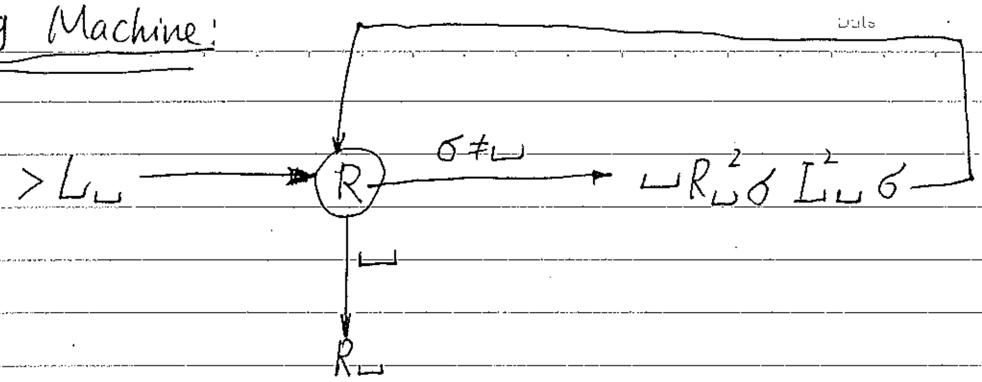
$\triangleright \underline{a} \sqcup a$ $\qquad$ $\underline{\triangleright} a \sqcup a$

M:

$R_{\overline{\sqcup}}$: $\triangleright a \sqcup \underline{a}$ $\qquad$ $R_{\overline{\sqcup}}$ $\triangleright \underline{a} \sqcup a$

$L\sigma$: $\triangleright a \underline{a} a$ $\qquad$ $L\sigma$ $\triangleright \underline{a}$

Copying Machine:



$\triangleright \sqcup a b \underline{\sqcup}$

$L_\sqcup$ $\triangleright \underline{\sqcup} a b \sqcup$

R $\triangleright \sqcup a b \sqcup$ $\sigma = a$

*TM are algorithms

$\sqcup$ $\triangleright \sqcup \underline{\sqcup} b \sqcup$

*TM can be given verbally.

$R^2_\sqcup$ $\triangleright \sqcup \sqcup b \sqcup \underline{\sqcup}$

σ $\triangleright \sqcup \sqcup b \sqcup \underline{a}$

$L^2_\sqcup$ $\triangleright \sqcup \underline{\sqcup} b \sqcup a$

σ $\triangleright \sqcup \underline{a} b \sqcup a$ Goto R

R: $\triangleright \sqcup a \underline{b} \sqcup a$ $\sigma = b$

$\sqcup$: $\triangleright \sqcup a \underline{\sqcup} \sqcup a$

$R^2_\sqcup$ $\triangleright \sqcup a \sqcup \sqcup a \underline{\sqcup}$

σ $\triangleright \sqcup a \sqcup \sqcup a . \underline{b}$

$L^2_\sqcup$ $\triangleright \sqcup a \underline{\sqcup} \sqcup a b$

σ $\triangleright \sqcup a \underline{b} \sqcup a b$ Go to R!

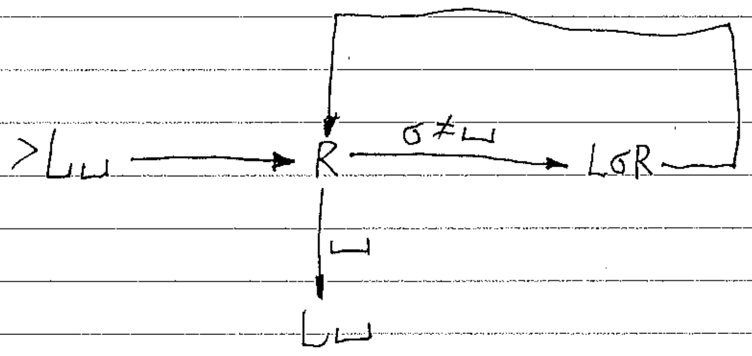
R: $\triangleright \sqcup a b \underline{\sqcup} a b \sqcup \sqcup$

$R_\sqcup$: $\triangleright \sqcup a b \sqcup a b \underline{\sqcup}$

8 Lecture 3/20th

Examples :-

Left shifting machine $S_{\leftarrow}$



$\triangleright \sqcup a b b \underline{\sqcup}$

$L_{\sqcup}$: $\triangleright \underline{\sqcup} a b b \sqcup$

- - - - - - - - . .

R: $\triangleright \sqcup \underline{a} b b \sqcup$ $\quad \sigma = a$

$L\sigma R$: $\triangleright a \underline{a} b b \sqcup$

- - - - - .

R: $\triangleright a a \underline{b} b \sqcup$ $\quad \sigma = b$

L: $\triangleright a \underline{a} b b \sqcup$

σR: $\triangleright a b \underline{b} b \sqcup$

- - - - - - -

R: $\triangleright a b b \underline{b} \sqcup$ $\quad \sigma = b$

$L\sigma R$: $\triangleright a b b \underline{b} \sqcup$

- - - - - - - . .

R: $\triangleright a b b b \underline{\sqcup}$ $\quad \sigma = \sqcup$

$L_{\sqcup}$ $\triangleright a b b \underline{\sqcup} \sqcup$

§ Lecture 15:

Language Recognizer

DFA. M: * (s, w) always halts (the length of $|w|$)

* $(s, w) \vdash_M^* (f, e)$: M accepts w

* M accepts a Language L iff
(1) $w \in L$ M accepts w
(2) $w \notin L$, M rejects w.

* L is a regular language

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

Idea: $\triangleright \sqcup aabbcc$ * Replace first symbol in each block by d

$\triangleright \sqcup dadbdc$ * Repeat

$\triangleright \sqcup dd\,dddd$ * If run out of a's & b's & c's at the same time (pass) accepts
if not, reject

Detail:

$\triangleright \sqcup aabbcc$

R: $\triangleright \sqcup aabbcc$

dR: $\triangleright \sqcup dabbcc$

R: $\triangleright \sqcup dabbcc$

dR: $\triangleright \sqcup dadbcc$

R: $\triangleright \sqcup dadbcc$

dR: $\triangleright \sqcup dadbdc$ dR a,d (in the graph)

R: $\triangleright \sqcup dadbdc\sqcup$

$L_\sqcup$: $\triangleright \sqcup dadbdc\sqcup$

R: $\triangleright \sqcup dadbdc\sqcup$

R: $\triangleright \sqcup dadbdc\sqcup$

dR: $\triangleright \sqcup dd\,dbdc\sqcup$

R: ▷⊔ddd<u>b</u>dc⊔
dR: ▷⊔dddd<u>d</u>c⊔
R: ▷⊔ddddd<u>c</u>⊔
dR: ▷⊔dddddd<u>⊔</u>
~~d~~
L⊔ ▷<u>⊔</u>dddddd⊔
R:

R ▷⊔dddddd<u>⊔</u>
Y acc.
↳ (goto yes state)

Eg2: ▷<u>⊔</u>a b⊔

R: ▷⊔ <u>a</u>b⊔
dR: ▷⊔ d<u>b</u>⊔
dR: ▷⊔ dd<u>⊔</u>
n:
[Eg:]
$a^i b^j c^k \quad i \times j = k$

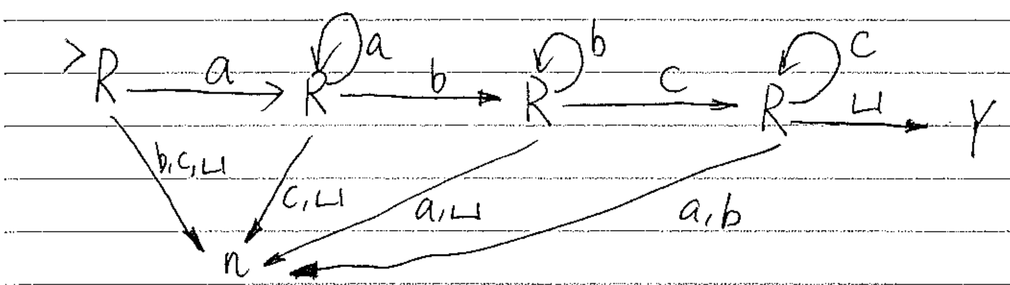
▷<u>⊔</u>aabb cccc⊔

▷⊔ ~~a~~a <u>~~bb~~</u> ~~cc~~cc⊔ one a run use b to cross c

▷⊔ ~~a~~a bb ~~cc~~cc⊔

▷⊔ ~~aa~~ ~~bb~~ ~~cccc~~⊔

T.M. for step 1. (graphical representation)



<u>Bouns Question</u> T.M. for step 3.

<u>Compute functions.</u>

$(S, \triangleright \underline{\sqcup} w) \vdash^* (h, \triangleright \sqcup \underline{y})$

Input output

$y = f(w)$

f: Recursive function

Successor function

$succ(n) = n+1$

n	10	11	15
binary	1010	1011	1111
$n+1$	1011	1100	10000

No.

Date

$$
\begin{tikzcd}
> R_\sqcup L \arrow[r, "1"] \arrow[dr, "0"] \arrow[d, "\sqcup"] & 0 \arrow[l, bend right] \\
1 S_R & 1 L_\sqcup
\end{tikzcd}
$$

(S_R: right shifting machine)

$\triangleright \sqcup 1010 \sqcup$	$\triangleright \sqcup 1011 \sqcup$	$\triangleright \sqcup 1111 \sqcup$
$R_\sqcup$ $\triangleright \sqcup 1010 \sqcup$	$R_\sqcup L$ $\triangleright \sqcup 1011 \sqcup$	$R_\sqcup L$ $\triangleright \sqcup 1111 \sqcup$
L $\triangleright \sqcup 1010 \sqcup$	0 $\triangleright \sqcup 1010 \sqcup$	$0L$: $\triangleright \sqcup 1110 \sqcup$
$1L$ $\triangleright \sqcup 1011 \sqcup$	L: $\triangleright \sqcup 1010 \sqcup$	$0L$: $\triangleright \sqcup 1100 \sqcup$
	$0L$: $\triangleright \sqcup 1000 \sqcup$	$0L$: $\triangleright \sqcup 1000 \sqcup$
	$1L_\sqcup$: $\triangleright \sqcup 1100 \sqcup$	$0L$: $\triangleright \sqcup 0000 \sqcup$
		1: $\triangleright 1\ 0000 \sqcup$
		S_R: $\triangleright \sqcup 10000 \sqcup$

Lecture 3/25

TM: M decides $L \subset \Sigma_0^*$

* $\forall w \in \Sigma_0^*$, M halts on w

* if $w \in L$, M accepts w (y state)

* if $w \notin L$, M rejects w (N state)

L: recursive/Turing-decidable : $a^n b^n c^n$; $a^i b^j c^k$ $i+j=k$

TM computes function
* $(S, \triangleright \sqcup w) \vdash (h, \triangleright \sqcup y)$

$y = f(w)$ $\quad$ $Succ(n) = n+1$

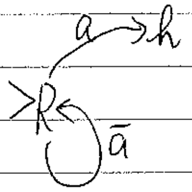
M ~~sem~~ semi decider L iff:
* if $w \in L$, M halts on w
* if $w \notin L$, M does not halt on w

L. Recursively enumerable

Note: can never decided $w \notin L$

$L = \{w \in \{a, b\}^*, w$ contains at least one $a\}$

M_1:



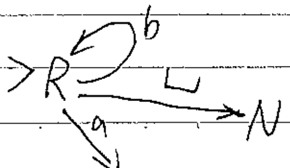
$w \in L$ $\quad$ $w \ldots\ldots a \ldots\ldots$

$w \notin L$ $\quad$ $b \sqcup b \ldots\ldots\ldots$

M_1 semi-decides L.

L also recursive.

M_2: decides L



$\triangleright \sqcup bbbab \ldots$

$\triangleright \sqcup bbbb \sqcup$

Theorem 1:

Proof: change the N machine.

non-recursively enumerable?

Recursively enumerable?

Recursive languages $\Rightarrow a^n b^n c^n$

Context-free languages
$S \to aSb,\ S \to e$

regular language $\to L(a^* b c^*)$

CFG $\subset$ R

Show that recursively enumerable = Recursive lang ~~e~~

Prepartion: All context free grammars can be written into Chomsky Normal Form

Type 1: $X \to yz$

Type 2: $X \to \sigma$ $(\sigma \neq e)$
$S \to e$ allowed only if $e \in L$

Eg. $S \to aSb \quad S \to e$

$S \to AX$
$X \to SB$
$A \to b$
$B \to b$

$S \to A\cancel{X} \to ASB \to \cancel{SAXB}$

$\cancel{S \to A} \to AAXB \to AASBB$

Derivation with Normal form:

* Apply Type 1 rules, k times
  $k+1$ non-terminals
* Apply Type 2 rules $k+1$ times
  length of word $= k+1$

$\left.\right\}$ $2k+1$ steps: $\left.\right\}$ $\Rightarrow 2n-1$

$n = k+1$

* w, $|w| = n$
  Derived in $2n-1$ Steps

CFG $\subseteq$ recursive

Proof:

* L generated by CFG G
* Assume G is in the normal Form.
* w, $|w|=n$, $w \in L$?
  * List all words generated in $2n-1$ steps.
  * If w is one of them, yes
    If not no

Mechanic can be done using TM.

§ Mile Stone: (notes from 4.16)

* Course objective:
  Show some problems cannot be solved using algos.
* what are algos? procedure, Recipes
* Intiuitive notion: Set of instructions for carrying out some task:
  (~~informal~~ cannot be used in Proofs:)

No.

* mathematical models: DFA, NFA, PA Limited in Power.

* Turing Machines:

* Is TM General Enough?

* Attempt to come up even more general models?

* Extend TM:

* other math models: unrestricted grammar,
λ calculus
post-production system, ...

by Turing himself

* All of those are actually equivalent to Turing Machine

Church-Turing Thesis:
TM = ALGORITHMS

$ax^2+bx+c=0$

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

~~Hilbert's 21 Q's~~

Problem:

$$a_0x^n+a_1x^{n-1}+\cdots+a_n=0$$

Does the equation have closed-form solution!
algebraic

Second Further reading

1. Multi Tape TM:

M1: 2-tape copying machine

objective

T1: ▷⊔ab T1: ▷⊔ab⊔ab.

T2: ▷⊔ T2: ▷⊔

Multitape ⇔ TM

T1 ▷⊔ab

T2 ▷⊔

Proof:

standard TM, M_2 multiple steps

▷#⊔ab#⊔

head position of M_2

virtual heads of M_1

1.1 T1: ▷⊔ab

T2: ▷⊔a

1.1. T1 ▷#

1.1. T ▷#⊔ab#⊔a

↳ R⊔⊔R(σ=a)σ

R⊔⊔R ~~RR~~ σ σ L#

1.2 T1: ▷⊔ab

T2: ▷⊔ab

1.2. ▷#⊔ab#⊔ab

Shift it to ~~left~~ right

1.3

T1: ▷⊔ab⊔

T2: ▷⊔ab⊔

1.3. ▷#⊔ab⊔#ab⊔

4.

1 T1: ▷⊔ab⊔

T2: ▷⊔ab⊔

2 T1: ▷⊔ab⊔a

T2: ▷⊔ab⊔

3 T1: ▷⊔ab⊔ab

T2: ▷⊔ab⊔

4 T1: ▷⊔ab⊔ab⊔

T2: ▷⊔ab⊔

8 Lecture 1st Apr.

No.

Date

Nondeterministic TM

* TM is deterministic

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

$$\underset{\text{state}}{(p},\ \underset{\text{char}}{a)} \vdash (q,b)$$

Computation is sequential

$$(s,\triangleright \cdots) \vdash (p_1,\triangleright,\cdots) \vdash (p_2,\triangleright \cdots)\cdots\cdots$$

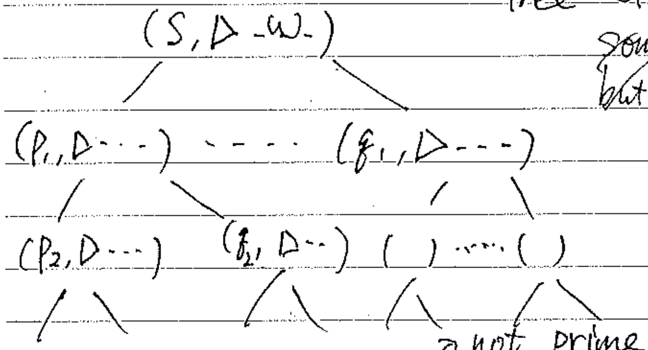
* Nondeterministic TM, (NTM)

$$\delta \qquad \Delta \subseteq ((K-H)\times\Sigma)\times(K\times(\Sigma\cup\{\leftarrow,\rightarrow\}))$$

$$(p,a)\rightarrow(q_1,b_1)$$
$$(p,a)\rightarrow(q_2,b_2)$$

computation with NTM

Tree structure.



~~some of the paths, never stops, but one of them it haults~~

decides

semidecides — { yes hault / no never stop }

not prime

$L = \{I^m : m \text{ is a composite number}\}$

NTM decides L

1* $\triangleright \underbrace{I\cdots\cdots I}_{m}\#$

2* Nondeterministically choose $1<p,q<m$

$$\triangleright\underbrace{I\cdots\cdots I}_{m}\#\underbrace{I\cdots\cdots I}_{p}\#\underbrace{I\cdots\cdots I}_{q}$$

2.2 check if $p\times q = m$

2.2 check if $p \times q = m$
if yes halt on y
if no halt on n

※ <u>Theorem 4</u>: If NTM M decides L, then exists DTM M' that decides L.

function — relation

<u>Proof</u>: Sketch (Different with Nodes) δ, Δ

* Rules of M $r_1, r_2, \cdots r_k$ $(p,a), (q,b)$
* $x \in \{1,2,\cdots,k\}^+$ sequence of rule applications $r_2((p_2,a_2),(q_2,b_2))$

$x=231$: step 1: apply rule 2
(2) r_3
(3): r_1:

* might not be feasible

* Simulate M Using 3-tape turing Machine:
  * Tape 1: Keep the input w
  * Tape 3: Enumerate $\{1,2,\cdots,k\}^+$ (the Path) in dictionary order
    First all length 1 string 1; 2; ... 9;
    Then all length 2 string 10; 11, 12, ...
    ⋮
  * Tape 2: simulate computation by M, according to x (go down the path on two) on tape 3;
    * Infeasible: next x
    * Doesn't halt: next x
    * halts on no: next x
    * halts on y: M' haults on yes.

/* similar to BFS (Breadth-first-search) */

If all x's of some length considered,
All feasible computations by M haults on n.
M' hault's on n;

Example: DTM:

1. ~~$\triangleright I \cdots I$~~

1. $\triangleright \underbrace{I \cdots I}_{m}$

2. For each ~~$1 \le$~~ $1 < p, q < m$

   2.1 $\triangleright \underbrace{I \cdots I}_{m} \# \underbrace{I \cdots I}_{p} \# \underbrace{I \cdots I}_{q} \#$

   2.2 If $p \times q = m$, haults on y

3. Haults on N

* depth first search

* one step for each branch in turn.

[Lecture 17]

C.F.L. close under

* union concatenation Kleene star

not closed under

* complementation, inter section

$L = \{a^n b^n c^n\}$
Not C.F.

$\overline{L} = L(\overline{a^* b^* c^*})$
$\cup \{a^n b^m c^k, n \ne m\}$
$\cup \{a^n b^m c^k, m \ne k\}$
$\overline{L}$ C.F.
but L is not

For recursive Langs. R.L.

1. L is r.L.
   TM M decides L
   $w \in L$, M halts on y
   $w \notin L$, M halts on n

M': same as M, except swap y, n;
M' decides the $\overline{L}$
$w \in \overline{L}$, $w \notin L$, M' halts on y
$w \notin \overline{L}$, $w \in L$, M' halts on n.

Proof of 2:

$\triangleright \sqcup w \sqcup \ldots$

$\triangleright \sqcup \sqcup \ldots$

* L_1, L_2 are r.L.
* M_i decides L_i $(i=1,2)$
* M: 2-tape TM.

1. copy input w to Tape2
2. simulate M_1 on Tape1

$$M_1 \text{ halts on } \begin{cases} y, & M \text{ halts on } y \\ n, & \text{goto } 3 \end{cases}$$

3. Simulate M_2 on tape 2:

$$M_2 \text{ ~~halt~~ halt on } \begin{cases} y, & M \text{ halts on } y \\ n, & M \text{ halts on } n \end{cases}$$

M decides $\overset{L_1}{\cancel{L_1}} \cup L_2$

Tape 1 : $\triangleright\triangleright \sqcup w \sqcup$ M_1

Tape2 $\triangleright \sqcup w \sqcup$ M_2

$$w \in L_1 \cup L_2 \Rightarrow \begin{cases} w \in L_1 & \begin{array}{l} M_1 \text{ halts on } y \text{ at } 2 \\ M \text{ halts on } y \end{array} \\ \begin{array}{l} w \in L_2 \\ (w \notin L_1) \end{array} & \begin{array}{l} M_2 \text{ halts on } y \text{ at } 3 \\ M \text{ halts on } y \end{array} \end{cases}$$

$w \notin L_1 \cup L_2$

$w \notin L_1$, M_1 haults on n

$w \notin L_2$, M_2 halts on n

M haults on n

Proof: $L_1 \cap L_2$ is r.L.

§ Lecture 4/10

No.

Date

TM is deteministic

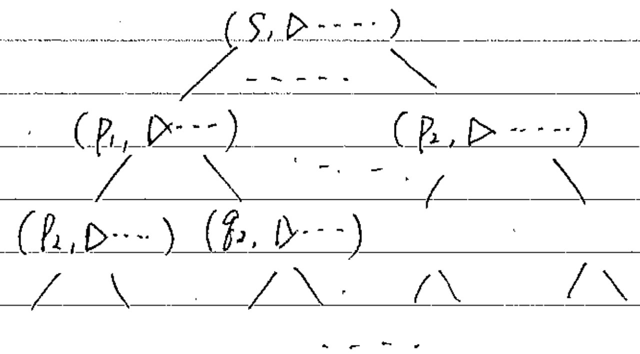
* Derivation is sequence

$(s, \triangleright \underline{a} b \ldots) \vdash (p_1, \triangleright \cdots) \vdash (p_2, \triangleright \cdots) \cdots$

NTM

* Derivation is a tree

<u>Math. Def.</u>



Depth of tree : 1000

NTM solves problem in 1000 steps

TM :(sequential machine) 2^{1000} steps

$\underline{NP} \overset{?}{=} P$

Lecture 17. Closure Properties

L_1, L_2, recursive

$i=1,2,$ M_i

$w \in L_i$ M_i halts on y

$w \notin L_i$ M_i halts on n

$L_1 \cap L_2$, $L_1 \cup L_2$ recursive

M:

Tape1: $\triangleright \underline{\sqcup} x_1$ simulate M_1

Tape2: $\triangleright \underline{\sqcup} x_1$ simulate M_2

M halts on y

iff both M_1 & M_2 halts on y

<u>M decides ~~M_1~~ $L_1 \cap L_2$</u>

M on y

iff either M_1 or M_2 h on y

M decides $L_1 \cup L_2$

No.

Date

L_1, L_2 recursive

M : * Nondeterministically split $w = x, y$

move y to Tape 2

Tape1: $\rhd \sqcup x$ simulate M_1, if halts y

Tape2: $\rhd \sqcup y$ simulate M_2, if halts y

M halts on y

M decides $L_1 L_2$

Proved.

L_1^* is recursive.

M: * Nondeteministially run subroutine 1 or subroutine 2

* subroutine 1:

Simulate M_1 on tape 1

* subroutine 2:

N-split $w = xy$, move y to tape 2

Tape 1: $\rhd \sqcup x$ simulate M_1

Tape 2: ~~$\rhd$~~ $\rhd \sqcup y$ Move y to Tape 1,

goto ①

Note: recursive langages closed under complementation

* L is recursive

* M decides L

construct M' same as M except swap the Y & N

Why this idea doesn't work for recursive enumerable language.

* L is r. e. L.

* M semi-decides L

$w \in L$ M halts

$w \notin L$ M does not halt

Why wrong?

$w \in L$, M halts, M' does not halt
$w \notin L$, M does not halt, M' halts

M' decides $\overline{L}$

*M' can't know if M doesnot hault.

To construct M', we need to have a way to decide the following whether M halts on w?

Haulting Problem. unsolvable

§ Lecture 18

ATM: an algorithm for solveing particular problem (program you write)

Universal TM: "computer" than can run various algorithms.
* Input: TM M, data w
* Do : Simulate M on w

prolog; C

Questions:
* How to encode TM & data in uniform way
* How to simulate

Example:

$(s, \triangleright \underline{a}a \sqcup) \vdash (q, \triangleright \underline{\sqcup} a \sqcup) \vdash (s, \triangleright \sqcup \underline{a} \sqcup) \vdash (q, \triangleright \sqcup \underline{\sqcup} \sqcup)$
$\vdash (h, \triangleright \sqcup \sqcup \underline{\sqcup})$

10 states (Suppose) encode the states; the symbols

$s - q0000$
$p_1 - q0001$
$p_2 - q0010$
$p_8 - q1000$ y } halting state
$p_9 - q1001$ n

the input; encode aa⊔

Tape1: ▷⊔ a100 a100 a000

Tape2: ▷⊔ $\underbrace{\text{q00 a100 q01 a000}}_{r1}$ $\underbrace{\text{q00 a000 q10 a00}}_{r2}$

.... $\underbrace{\text{q01, a001, q01 a011}}_{r5}$ (rules)

Tape3: ▷⊔ q00
(initial state)

scan Tape 2

step1:

Tape1: ▷⊔ a000 a100 a000

Tape3: ~~q01~~ ▷⊔ q01

step2:

Tape1: ▷⊔ a000 a100 a000

Tape3: ▷⊔ q00

Step3:

Tape1: ▷⊔ a000 a000 a000

Tape3: ▷⊔ q01

Step4:

Tape1: ▷⊔ a000 a000 a000

Tape3: ▷⊔ q00

Step5:

Tape1: ▷⊔ a00 0a000 a000

Tape3: ▷⊔ q10 → halt (don't have any outgoing transactions.)

No.

Date

* TM M operates on String

* Encoding of M, 'M' is a string

* M can operates on 'M'

* L = { 'M' : M is TM with certain property }

Language { Problems: }

§ 4/15
19 - The haulting Problem

L recursive : ∃ TM, M decides L
$w \in L$, M hault on w on y
$w \notin L$, M hault on w on n

closed under complementation, $\bar{L}$

L recursively enumerable, M semi-decides L, ~~w∉L~~
$w \in L$, M hault on w on y
$w \notin L$, M $\nearrow$
not closed under complementation:

Universal TM. * "M" "W"
* simulate M on W
* TM M operates on string W
* M can operate "M"
* L = { "M", M with property }
* Deciding "M" ∈ L ⟺
Deciding ~~M~~ if M has the property

$\overline{H_0} = \{ "M" : M \text{ does not halt on } "M" \}$ inverse of the diagnol element

Lemma1 : $\overline{H_0}$ is not r.e. (recursively enumerable)

Proof: Suppose $\overline{H_0}$ is r.e. ("M^*" is the encoding)

* Exist M^* s.t. $\begin{cases} w \in \overline{H_0}, M^* \text{ halts on } w & ① \\ w \notin \overline{H_0}, M^* \text{ does not halt} & ② \end{cases}$

* "M^*" $\in \overline{H_0}$
  * by definition of $\overline{H_0}$; M^* doesn't halt on "M^*",
  * M^* does not halt on a word of $\overline{H_0}$, contradicts with ①

* "M^*" $\notin \overline{H_0}$
  * coz of ②, M^* doesn't halt on "M^*"
  * by definition of $\overline{H_0}$ "M^*" $\in \overline{H_0}$

contradiction:

Diagnolization principle

$S = \{ P : p$ doesn't cut own hair $\}$

Exist a person (p^*) who only cuts the hair of anyone who does not cut own hair.] rule

$P^* \in S$: By def of S, p^* doesn't cut ~~his~~ own hair
by rule, P^* cut his own hair

$P^* \notin S$ By def of s, p^* cut his own hair.
By rule, he shouldn't cut his own hair.

$\overline{H_1} = \overline{H_0} \cup \{w: w$ is not encoding of TM$\}$

Theorem 1: $\overline{H_1}$ is not r.e. (recursively enumerable)

Complement of $\overline{H_1}$?

$\{w: w$ is encoding of TM. M; M halts on "M"$\}$

$H_1 = \{$"M"; M haults on the "M"$\}$

Lemma2:
~~Theorem2~~: H_1 is not recursive

proof: suppose H_1 is recursive, then $\overline{H_1}$ is recursive
then $\overline{H_1}$ is r.e.
contradicts with theorem 1

$H = \{$"M" "w": M halts on $w\}$

Halting Problem
Does M halts on w?
$\Longleftrightarrow$ "M""w" $\in H$?

Theorom 2: H is not recursive. The halting Problem is undecidabl

H is recursive
Exist M^*:
"M""w" $\in H$, M^* halts on y
"M" "w" $\notin H$, M^* halts on n

/*
* Decidable worst
* NP-complete bad
* P good
* n
* log n
*/

Proof: suppose H is recursive,
TM M_H decides H
M_H: method to (B) decide if a turing machine M halts on w

* show: can design a method to (A) decide
if M halts on "M" Contradiction with lemma 2

M_{H_1} (Method)
* Input X
* Run M_H on X"X"

M_H halts on y
* X is encoding of TM, M
* M halts on "M"

M_H halts on n
* X is encoding of TM, M
* but M not halt on itself "M"

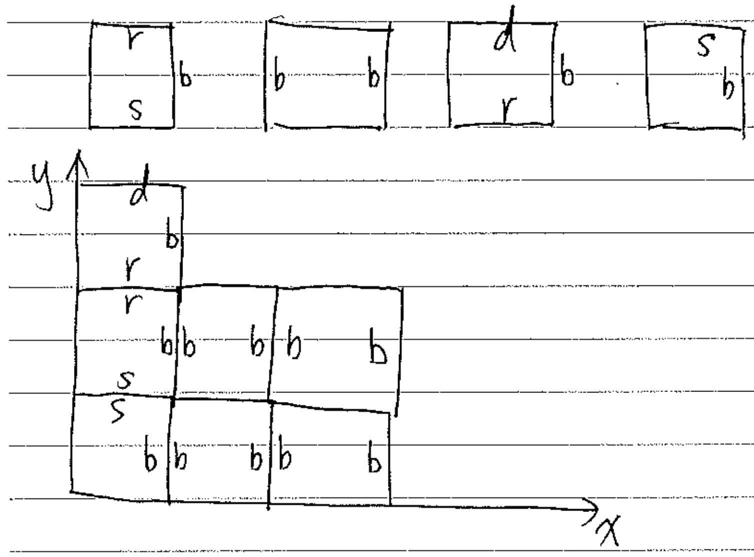
OR X is not an encoding of TM M

M_{H_1} decides H_1 contradiction with Lemma 2.

$A \Rightarrow B$ * soln for B can be used for A
* if A not solvable conclude B not solvable

To prove B is unsolvable
* Find A
① A is known to be unsolvable
② A can be reduced to B
$A \Rightarrow B$

TILING Problem:



recap:

8 Lecture 4/17

$\overline{H_0} = \{\text{"}M\text{"}, \underline{M \text{ not halt on "}M\text{"}}\}$ ☆

undecidable

is not r.e.

$\overline{H_1} = \overline{H_0} \cup \{w, w \text{ is not encoding of TM}\}$

is not r.e.

$H_1 = \{\text{"}M\text{"}; M \text{ halts on "}M\text{"}\}$

is not recursive

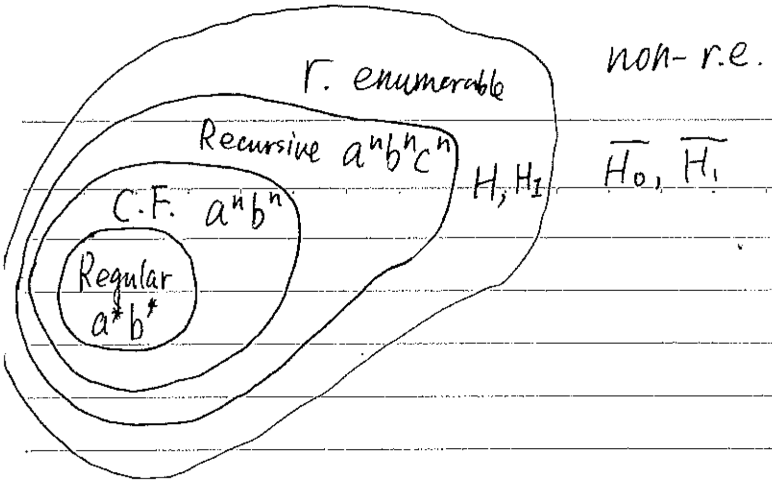
$H = \{\text{"}M\text{""}w\text{"}, M \text{ halts on } w\}$

is not recursive undecidable ☆

halting problem is undecidable

Regular

a^*b^*



H is r.e.
UTM semi-decides H
$X = "M""w"$ H X
Run UTM on X, Simulate M on X halt
$X \notin H$ $\begin{cases} X \text{ is not encoding of TM} \\ X = "M", \text{ M does not halt on } w \end{cases}$ } not halt

$\overline{H}$.
Assume $\overline{H_0}$ is r.e.
Exist M^* $\begin{cases} w \in \overline{H_0} & M^* \text{ halts on } w \quad ① \\ w \notin \overline{H_0} & M^* \text{ not halt } w \quad ② \end{cases}$
the encoding of this lang.
$"M^*" \in \overline{H_0}$ M^* not halt on "M^*" (by def. of $\overline{H_0}$) contradicts ①
$"M^*" \notin \overline{H_0}$ By ② M^* not halt on "M^*" contradicts def. of $\overline{H_0}$

§ Lecture 20
Theorem 2:
$K_1 = \{"M"$: Turing machine M halts on an empty tape$\}$ is not recursive.
Proof: * suppose K_1 is recursive
* Exist TM M_{K_1}:
$w \in K_1$ M_{K_1} halts at y
$w \notin K_1$ M_{K_1} halts on n

No.

* M_{K_1}: A method to decide if a TM halts on empty tape at y

(B) a TM halts on empty tape at y

* show: can design method to deciede if

(A) a TM M halts on w

Contradiction: $\Rightarrow K_1$ not recursive.

Q: $B \Rightarrow A$?

Need TM that starts from empty tape & related to A

Design M_w: Simulate M on w
from empty tape

Method for A (M_H)

* Input: M, w
* Build M_w
* Run M_{K_1} on "M_w"

$\begin{cases} y & M_w \text{ halts on empty tape} \\ & M \text{ halts on } w \\ n & M_w \text{ not halts on empty tape} \\ & M \text{ not halts on } w \end{cases}$

M_H decides H,

$\begin{cases} w \in H & M_H \text{ halts at } y \\ w \notin H & M_H \text{ halts at } n \end{cases}$

Theorem 3: $K_2 = \{"M" : \text{Turing Machine } M \text{ halts on every input string}\}$ is not recursive

Proof: * suppose K_2 recursive
* exist M_{K_2}

$$\begin{cases} w \in K_2 & M_{K_2} \text{ halts at } y \\ w \notin K_2 & M_{K_2} \text{ halts at } n \end{cases}$$

(B) Decides if a TM halts on every input

* show: Can design method (M_{K_2}) for solve

(A) if a TM M halts on empty Tape, contradiction:

(B) $\Rightarrow$ (A) Define TM
M^*: * Erase the input
* simulate M on empty Tape

Method:
M_{K_2} (Method for A)
* Input: "M"
① Build M^*

② Run M_{K_2} on "M^*"

$$\begin{cases} y & M^* \text{ halt on every input} \\ & \Leftrightarrow M \text{ halts on empty tape} \\ n & \text{not halt on some input} \\ & \Leftrightarrow M \text{ not halt on empty } \cancel{\text{em}} \text{ tape} \end{cases}$$

No.

Date

Theorem 4:

$K_4' = \overline{K_3}$ closure property

Proof: suppose K_4 is recursive

Exist a TM M_{K_4}

$w \in K_4$, M_{K_4} halts at y

$w \notin K_4$ M_{K_4} halts at no

(B) decide if two turing machines halt same inputs

Design M^*: halts on every input

Method for M_{K_2}:

input: "M"

① Run M_{K_4} on "M" "M^*"

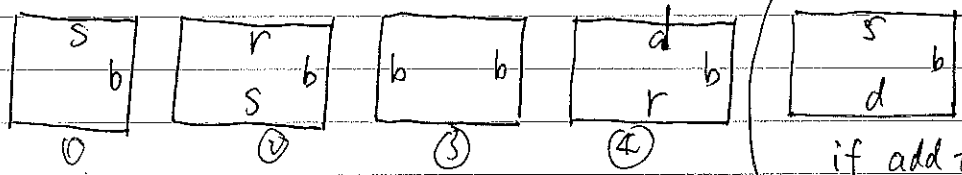
$\begin{cases} y & \text{halts on every input} \\ n & \end{cases}$

Ms can actually decides if M halts on e.

Pilling Problem:

Bounded Tiling is First NPC Problem

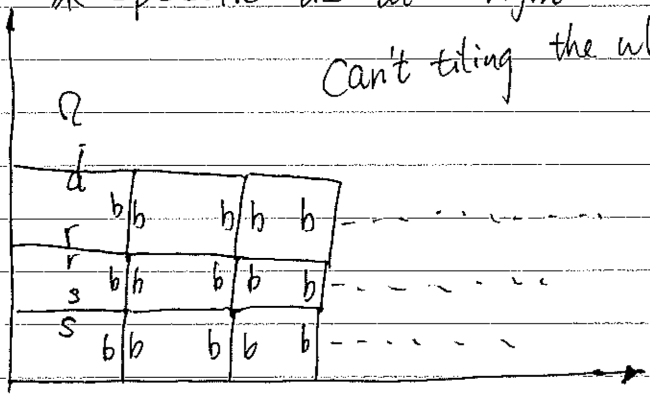
* D: Collection of tiles with marking on some edges

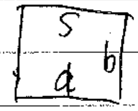


(if add this one, it works)

* possible to tile 1st quadrant
  * neighors share same marking
  * specific tile at origin

Can't tiling the whole quadrent





H = {(1, 3), (2, 3)
(~~4~~3, 2), (3, 3) ....

~~V = {(1, 2), (2, 4)~~ ....

Formal def.

* D, tiles $d_0 \in D$

* $H, V \subseteq D \times D$ Horizontal / vertical Constraints

* Tiling: $f: N \times N \rightarrow D$

  $f(0,0) = d_0$

  $(f(m,n), f(m+1,n)) \in H$

  $(f(m,n), f(m,n+1)) \in V$

8 lecture Apr. 22

Reduce problem A to problem B
* sln for B can be used to solve A
* If A is undecidable, B is undecidable

M halts "M" ? (Diagonolization)

↓

H: M halts on w ? H_n : M fails to halt on w → Tiling

↓

M halts on e ? → L(M) regular ?

↓

M halts on all (some) input ?

$L(M) = \{w : M \text{ halts on } w\}$

Proof of ⑤ L(M) is regular

* Suppose it is decidable

* Exist TM M_5 (method) to (B)
  decide if language semidecide by a TM is regular ?

* Show: Exist method to (A) decide
  if a TM M halts on e

UTM:
* "M" "w"
* Simulate M on w

Define M^*
* keep input X
* Run M on empty tape
* Run UTM on X

$L(M^*) = \begin{cases} \phi & M \text{ fails to halt} \rightarrow \text{regular} \\ \{\text{"M"}, \text{"w"}, & M \text{ hats on } e \quad \text{not regular} \end{cases}$

Tiling Problem:
* Given D, d_0, H, V
* Exist tiling?

Theorem: Tiling problem is undecidable

Proof: Reduce H_n to T

* H_n: if M fails to halt on e

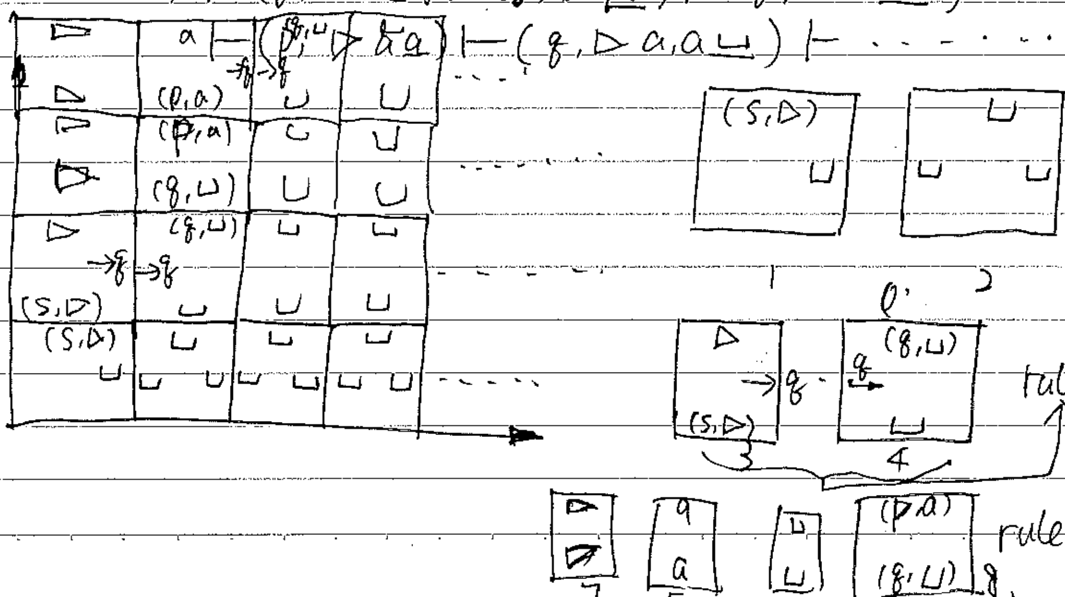
* create tiling system to simulate computation by M
  * 1st row: Initial Conf.
  * 2nd row: conf. after 1st derivation

M fails to halt $\equiv$ Exist Tiling

example

$M: \delta(s, \triangleright) = (q, \rightarrow)$ ①
$\delta(q, \sqcup) = (p, a)$ ②
$\delta(p, a) = (q, \rightarrow)$ ③

$(s, \underline{\triangleright}) \vdash (q, \triangleright\underline{\sqcup}) \vdash (p, \triangleright\underline{a}) \vdash (q, \triangleright a\underline{\sqcup})$



rule ①

rule ②

▷
▷

Rules 3:

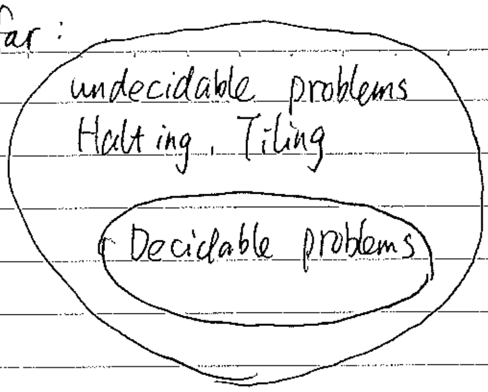
9	10
a	$(q, ⊔)$
$\to q$	$\to q$
(p, a)	⊔

▷	a	(p, a)	⊔
▷	a	$(q, ⊔)$	⊔
▷	a	$(q, ⊔)$	⊔
▷	(p, a)	⊔	⊔

polynominal Time reduction

8 lecture Apr. 24

So far:



$P \subseteq NP$

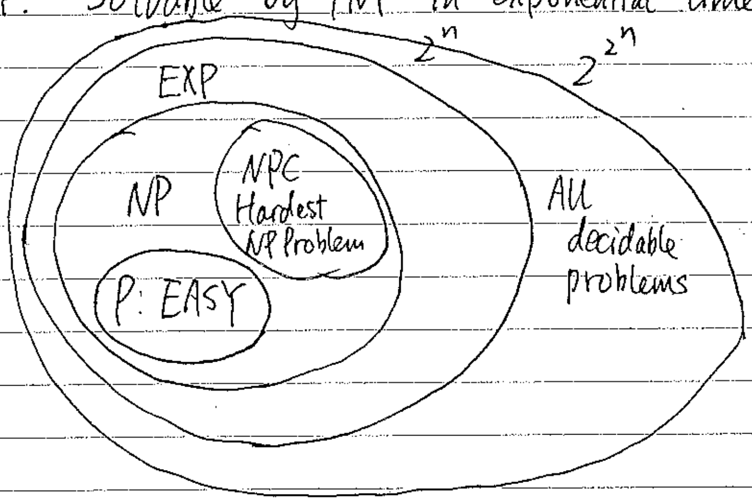
$NP \subseteq P$?

$P \stackrel{?}{=} NP$

P: problems solvable by TM in poly-time.

NP: solvable by NTM in poly time

EXP: Solvable by TM in exponential time



NPC = NP complete

TM is polynomially bounded if for any input x, any configuration C

$(s, \triangleright\sqcup x) \vdash^{T} C$ T: # of steps;

$T \leq P(|X|)$ $|X|$: Input Size

$P(\,)$: polynomial

Example:

(1) * Determine if DFA M accept w

* Time $= |w|$

polynomial

(2) * Determine if n (Integer) is prime   inputsize $S = \log_2 n$ ~~n=b~~

```
for (i=0; i<n/2, i++)
  if (n mod i == 0)
      return no;
return, yes
```

not in this course

Running time interms of value of

* Here: Running time interems of size of n

* Time $O(n) = O(2^S)$

Cryptography: n 300 digits

1,000,000,000

NPC Problems

* Large class of problems, no poly-time algorithm found

* proved: If one NPC Problem solvable in poly-time

ALL NPC problems solvable in poly-time

* Belief: NPC problem can't be solved in poly-time.

* you: Show your problem is NPC

Class $P = \{L : L$ decidable by TM in poly-time $\}$

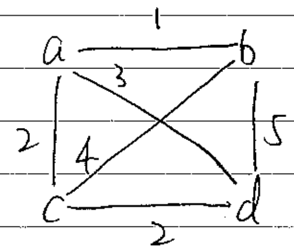
$L \subseteq \Sigma^*$

$w \in \Sigma^*$ $w \in L$?

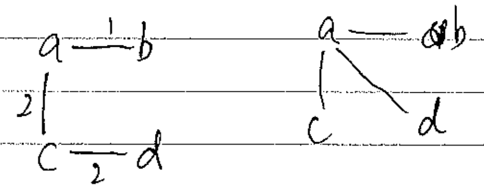
Can be answered in $p(|w|)$ time

Optimization : problem : optimize objective function eg. MST

Decision Problems : yes or no



spanning trees :



G has spanning tree has weight ≤ 6 ? yes Desicion version

≤ 3 ? no

Inputsize of G $|V|+|E|$

DST $= \{$ "(G, K)" : G has spanning tree of weight $\leq k \}$

DST $\in P$?

$f(S)$

DKnapsack $\in P$? sizeofbag

Time $O(nW)$

Iput Size : $S = \log_2 W + \sum_{i=1}^{n} \log_2 w_i + \sum_{i=1}^{n} \log_2 V_i$

n prime

$O(n)$ $O(2^S)$

Closure Property of P

$L_1, L_2 \in P \Rightarrow L_1 \cap L_2 \in P$

Proof: $x \in L_i$, decided by M_i (TM) in $O(|X_i|^{c_i})$ $(i=1,2)$

construct M for deciding if $x \in L_1 \cap L_2$

Tape1: $\triangleright \sqcup x$ simulate M_1 M halts Y iff M_1 & M_2 halt on y

Tape2: $\triangleright \sqcup X$ simulate M_2

Running Time of M: $O(|X_1|^{c_1} + |X_2|^{c_2}) = O(|X|^{c_1+c_2})$

$\underline{NP}$.

No.

* NTM is poly-bounded if for any input X, any configuration C

$$(S, \triangleright \sqcup X) \vdash^{T} C$$

$T \leq p(|X|)$, $\quad p(\ )$ is polynomial

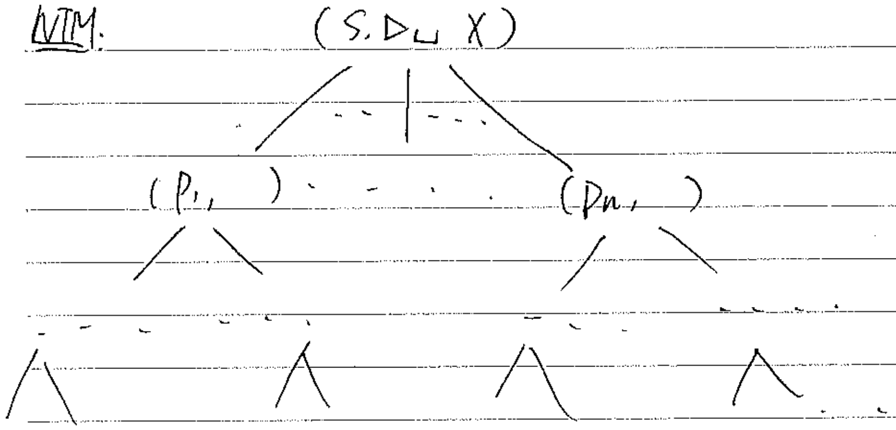
* No branch takes more than poly-time

* each branch takes poly-time
  ALL branches might takes more than poly-time

TM:

$(S, \triangleright \sqcup X) \vdash \cdots \vdash \cdots \vdash \cdots \vdash \cdots$

NTM:



* $P \subseteq NP$

Example: (COMPOSITE)

* Is X a composite number?

NTM:

* Nondeterministically choose (m, n)
* Test: $m \cdot n = X$?
  return Y/N

TM: for ($i=1$, $i < X/2$; i++)
  if $X \bmod i == 0$
    return yes
  return no;

Exponential

		Size
m	1011	4
n	× 101	3
	1011	
	0000	
	1011	

3×4

$O(|n||m|)$

§ 4/29

$P \equiv \{L, L$ decidecble in p-time by $\underline{\underline{TM}}\}$

Determine "$w \in L$?" in time $P(|w|)$

$NP = \{L$ decidable in P-time by $\underline{\underline{NTM}}\}$

$P \subseteq NP$

Is x composite?

* Non-deterministically choose m, n — decidable NTM $\xrightarrow{} \subsetneq NP$
* check $mn = x$?
  - yes → return yes
  - no

return no.

231	yes
61	~~yes~~ no

$x \in L$ — x is a yes-input

$x \notin L$ — x is a no-input

is Composite

$x = 231$ yes input

certificates
- $m = 3, \quad n = 77$
- $m = 7 \quad n = 33$
- $m = 11 \quad n = 21$

~~$x = 61$~~

~~$x = 61$~~

* certificate: for yes-input
  * sth. allows to verify x is indead yes-input

* verification.
  Input: x, certificate
  Do: verify x is yes-input

If certificate verification takes p-time, then problem $\in NP$

* Nondeterministically pick certificate
* verify.

To show problem $\overset{\in}{\text{in}}$ NP
  * Find certificate
  * prove verification takes p-time

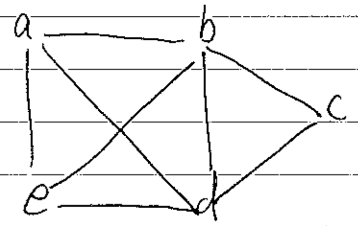
D Subset Sum $\in NP$

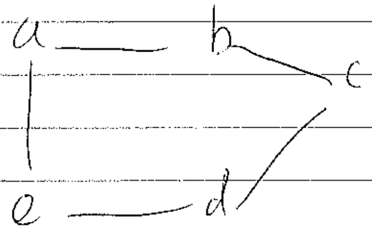
$\{1, 2, 3, \cdots, 10\}$

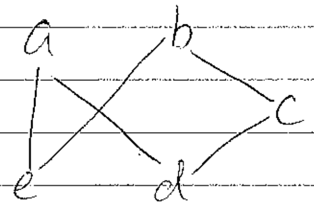
$C = 20$ $10+5+4+1 =$ yes give a subset I'll add
verification: take sum
poly-time

$C = 100$ no

D Hamiltonian Cycle

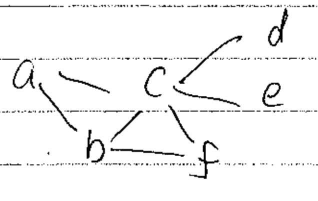






size of graph $S = |V| + |E|$
  checking very easy

★ Vertex cover.



{b, c} V.C.

give a set of verteice as certificate
easy to verify

P : closed under all operations

NP: complement ?
$L \in NP$
$\bar{L} \in NP$?

Satisfiability :

$x \to y \equiv \neg x \vee y$

basic three :
$\bar{x}$ not
$x \vee y$ or
$x \wedge y$ and

$f(x,y) = (x \vee \bar{y}) \wedge (\bar{x} \vee y)$
Truth assignment

x	y	f
0	0	1 ← f is satisfiable
1	0	0
0	1	0
1	1	1 ←

$g(x) = \bar{x} \wedge x$ Not satisfiable

Conjunction $\wedge$
Disjunction $\vee$

No.

Date

SAT $\in NP$

Terms:

literals : $x, \bar{y}$ (var (boolean) or negation of var)

clauses: Disjunction of literals

conjunctive Normal Form : conjunction of clauses
C.N.F

All Boolean formulae can be transformed into CNF

K-CNF Formula in CNF where each clause has k or fewer ~~literature~~ literals

3-SAT $\in NP$ 2-SAT $\in$ ~~N~~ P

§Lecture 22:

Reduction from problem A to problem B
* soln for B can be used to solve A
* If A is undecidable, then B undecidable.

* $L_1, L_2 \subset \Sigma^*$
* $f: \Sigma^* \to \Sigma^*$
  - $x \in L_1 \Leftrightarrow f(x) \in L_2$
  - $f(x)$ computed in $p(|x|)$ time

f is called a poly-time reduction from L_1 to L_2

If such that f exist, $L_1 \leq_p L_2$ // suppose algorithm A_2
* for deciding $y \in L_2$

* in $p(|y|)$ time.

Algorithm A: decides if $x \in L_1$

① $y = f(x)$ $\quad p(|x|)$ $\quad |y| = P(|x|)$

② Run A_2 on y $\quad p(|y|) =$ polynomial of $|x|$

$L_1 \quad L_2 \quad L_3$

$x \quad y = f(x) \quad g(y)$

$\underbrace{x \quad y}_{P(|x|)} \quad \underbrace{y \quad g(y)}_{p(|y|) = p(|x|)}$

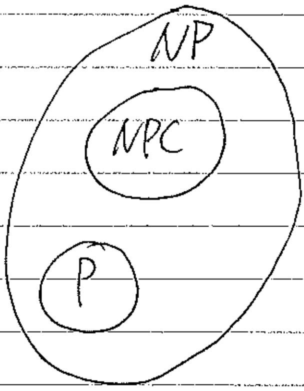
NP-complete (NPC) $\qquad NPC \subseteq NP$

L is NPC if

* $L \in NP$
* $\forall L' \in NP, L' \le_p L$

If L ~~decide~~ decidable in P-time
all NP-problems solvable in p-time $\qquad P \subseteq NP$

$NPC \subseteq NP$



$\begin{cases} P \cap NPC \neq \emptyset \\ \exists L \in NPC, L \in P \Rightarrow NP \subseteq P \\ \Rightarrow NP = P \text{ (believed false)} \end{cases}$

$\begin{cases} P \cap NPC = \emptyset \\ \checkmark \end{cases}$

No.

Date

First NPC Problem

Bounded Tiling
↓
SAT (Cook's Theorem)
↓
3-SAT
↓
Clique
↓
Independent Set
↓
Vertex cover

Known: SAT $\in$ NPC

show: 3-SAT $\in$ NPC

Idea prove SAT $\leq_p$ 3-SAT

ϕ CNF formula where some clauses have > 3 literals

$\phi: x_1 \wedge (\underbrace{x_2 \vee \bar{x}_3}_{a} \vee \underbrace{\bar{x}_4 \vee x_5 \vee x_6}_{b}) \overset{x_1 \wedge}{\equiv} (x_2 \vee \bar{x}_3 \vee y_1) \wedge (\bar{y}_1 \vee \bar{x}_4 \vee x_5 \vee x_6)$

CNF

$\overset{x_1 \wedge}{\equiv} (x_2 \vee \bar{x}_3 \vee y_1) \wedge (\bar{y}_1 \vee \bar{x}_4 \vee y_2) \wedge (\bar{y}_2 \vee x_5 \vee x_6)$

$f(\phi)$ * 3-CNF

* ϕ is satisfiable $\Leftrightarrow f(\phi)$ is satisfiable

A general fact

$a \vee b$ satisfiable $\Leftrightarrow (a \vee y) \wedge (\bar{y} \vee b)$

LHS true iff either a or b true.

Satisfiable: RHS true

$y=1$ b true

or $y=0$ a true