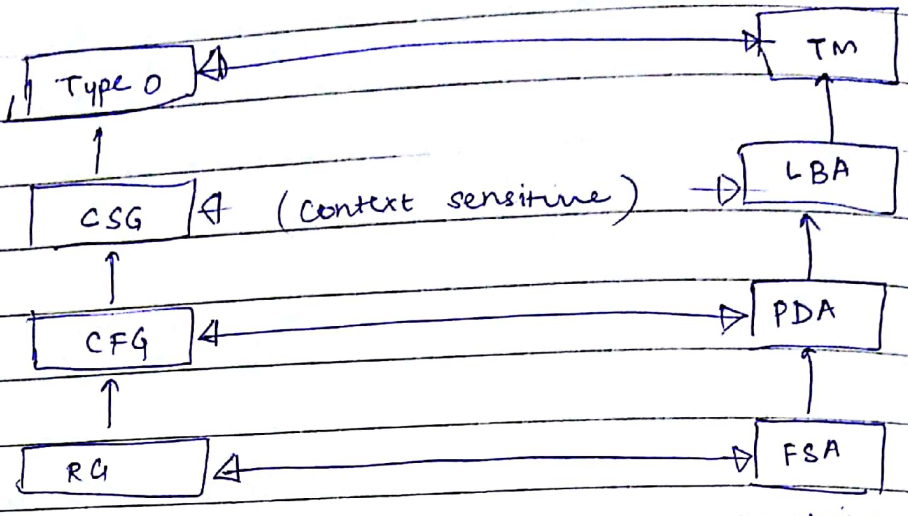
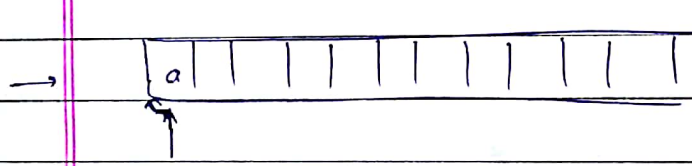
PUSHDOWN AUTOMATA

Grammar Hierarchy:

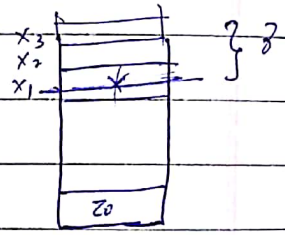


machine hierarchy

Chomskian Hierarchy of Languages



Transition table



Stack

not only sees 'a' but also top of stack (symbol at top of stack)
(i/p symbol)

Transition :

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

PDA : $\delta(q, a, X) = (q', \gamma)$

↓

X can be replaced by sequence of symbols called $\gamma = X_1 X_2 X_3 \ldots X_k$

↓

can be empty (ϵ)

↓

{X is deleted / X is popped up}

Teacher's Signature

* **Automata** : Finite representation of an infinite no. of things.

$a^n b^n$: not accepted by a FA;

↳ don't have memory, don't know count of 'a' occured

↳ can't move to b till a is completed.

↓

Possible in case of Pushdown Automata.

→ can store a in stack & whenever get a 'b', pop out 1 'a' out of the stack.

* Everything FA is a PDA (without any use of stack)

$\{a^n b^n ; n \geq 1\}$. accepted by PDA

$\{a^n b^n c^n ; n \geq 1\}$: not " " "

Q Every finite subset of Non-Regular set is Regular. (→ Regular)

Yes.

Q Let L be a RL. What can you say about L^R ?

Yes

Q. Union of RL is RL

Yes

Q. Infinite Union of Regular set is Regular

No.

Teacher's Signature

12/3/18

PDA

DATE: / /
PAGE NO.:

Introd^n to Automata Theory — Hopcropt, Ullman, Motwani

DFA = NDFA = ϵ-NDFA (can convert from either one to anyone)

Deterministic PDA : DPDA < NPDA (Power)

PDA : $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ — Γ: stack, Z_0: starting stack alphabet

Q → finite set of all states

Σ → finite set of i/p symbols / alphabets

Γ → finite set of stack alphabets

q_0 → Initial state

Z_0 → start stack symbol

F → set of all final states

NPDA : $\delta \rightarrow \delta : Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ [$2^{Q \times \Gamma^*}$: set of all subsets of $Q \times \Gamma^*$]

(Γ^* : string of symbols over Γ)

(go to multiple states ⇒ 2^m (subset))

2 ways of accepting automata :

→ Acceptance by final states

→ Acceptance by Empty Store : whenever stack is empty & you've read the whole i/p string.

ID of a PDA : (q, w, x) / (q, w, β) — instantaneous descrp^n

q: current state; w: i/p string; x / β: top of stack

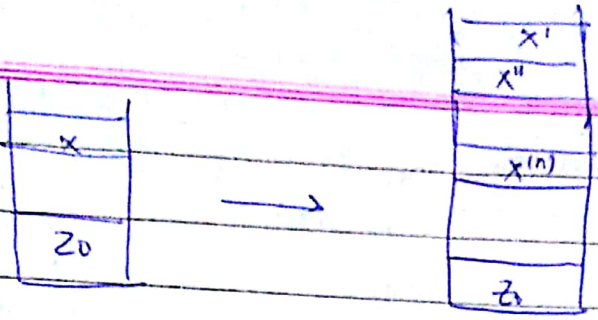
δ : $(q, aw, x) \vdash (q', w, x')$

$\delta(q, a, x)$ includes (q', α) — because it's non-deterministic

q goes to q'; aw goes to w ⇒ no need to show

$\alpha = x' x'' x''' \ldots x^{(n)}$ — string of symbols going to be top of stack

Teacher's Signature



$\alpha \in \Gamma^*$

if $\alpha = \wedge$ → just popping out X

$\alpha = X_1$ → replacing X by X_1

① Acceptance by final state :

(q_0, w, Z_0) $\quad w = a^n b^n$

starting ID

can be any symbol in stack

$L = \{ w \mid w \in \Sigma^* \text{ and } (q_0, w, Z_0) \vdash^* (q_f, \in, X) \}$

where $q_f \in F$, $X \in \Gamma$

② Acceptance by Empty Store :

$L = \{ w \mid w \in \Sigma^* \text{ and } (q_0, w, Z_0) \vdash^* (q, \in, \in) \}$

0 | 1

→ $L = \{ ww^R \mid w \in \{0,1\}^* \}$

$\delta(q_0, 0, Z_0)$ includes $(q_0, 0Z_0)$ ⇒ Pushing 0 in Z_0

need to store 0 because have to check pattern

$\delta(q_0, 1, Z_0)$ includes $(q_0, 1Z_0)$

Teacher's Signature

at any point of time, it may say it is finished. Now, check for WR ⇒ go to new state q_f & no change in the stack

→ final state

To accept all string ←

$\delta(q_0, \in, Z_0)$ includes (q_f, Z_0) → Non-deterministic now

→ tos

$\delta(q_0, 0, 0) = (q_0, 00)$

$\delta(q_0, 0, 1) = (q_0, 01)$ → tos

$\delta(q_0, 1, 0) = (q_0, 10)$ → tos

$\delta(q_0, 1, 1) = (q_0, 11)$ → tos

→ at any time it may say w is finishing so reach to q_1

$\delta(q_0, \in, 0|1) = (q_1, 0|1)$

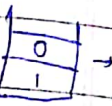
Reverse: (WR)

$\delta(q_1, 0, 0) = (q_1, \in)$ → pushing $\in$ in stack

$\delta(q_1, 1, 1) = (q_1, \in)$

$\delta(q_1, \in, Z_0) = (q_2, Z_0)$

1001

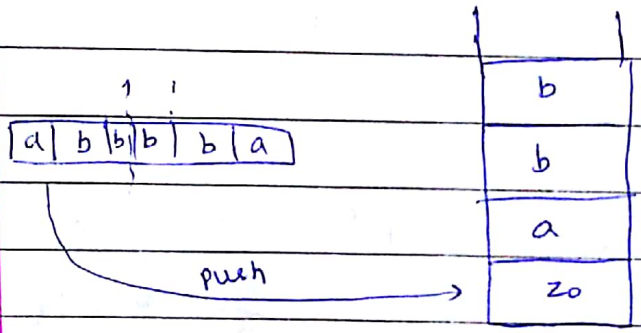


→ got w in stack

If have 0, tos will also be 0

↓

final state accept it



after this, pop &

q_0 : for reading w

when checking for w^R : go to q_1

From q_1 : check if string & tos are same. If so, pop the tos.

→ If a string is accepted by PDA, there will be atleast 1 path to reach final state.

→ At any moment, it has equal probability to either remain on same state or move to next state

$Q = \{q_1, q_0, q_2\}$ $\quad \Sigma = \{0,1\}$

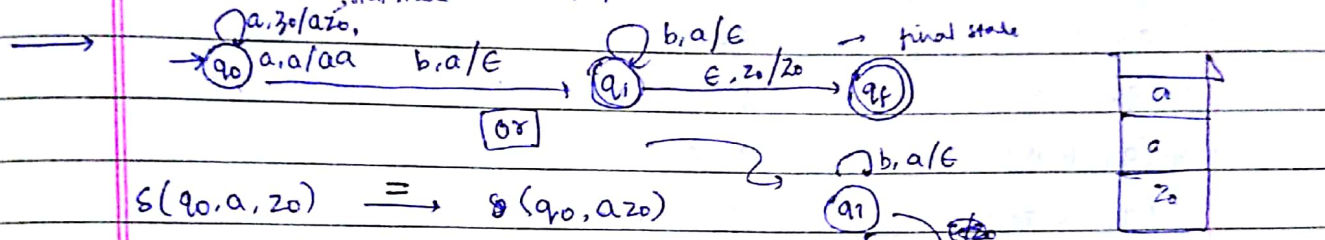
$F = \{q_f\}$ $\quad \Gamma = \{Z_0, 0, 1\}$

Eg: $L = \{a^n b^n \mid n \geq 1\}$ → here, we can find when a will begin and b will start (without counting)

↳ order dhyaan rakhna h → b aane pe state change

When reads a → push

When reads b → pop 1 a with 1 b



→ Acceptance by empty stack

$\delta(q_0, a, Z_0) \Rightarrow \delta(q_0, aZ_0)$

$\delta(q_0, a, a) \Rightarrow \delta(q_0, aa)$

$\delta(q_0, b, a) \Rightarrow \delta(q_1, \epsilon)$

$\delta(q_1, b, a) \Rightarrow \delta(q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) \rightarrow \delta(q_f, \epsilon)$ $\quad \delta(q_1, \epsilon, Z_0) \rightarrow (q_f, \epsilon, Z_0)$

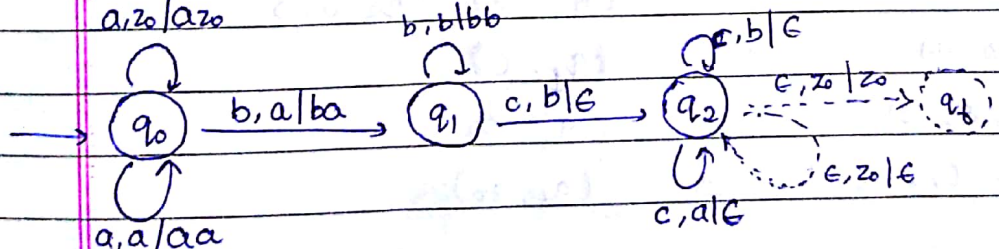
1.) $L = \{a^i b^j c^k \mid i, j, k \geq 1,\ i+j=k\}$

2.) $L = \{a^i b^j c^k \mid i, j, k \geq 1,\ i+k=j\}$

* 3.) $L = \{a^m b^n \mid m, n \geq 1,\ m \leq n \leq 2m\}$

4.) $L = \{a^n b^n \mid n \geq 1\}$

1.) $L = \{a^i b^j c^k \mid i, j, k \geq 1,\ i+j=k\}$



$\delta(q_0, a, Z_0) \rightarrow \delta(q_0, aZ_0)$

$\delta(q_0, a, a) \rightarrow (q_0, aa)$

$\delta(q_0, b, a) \rightarrow (q_1, ba)$

$\delta(q_1, b, b) \rightarrow (q_1, bb)$

$\delta(q_1, c, b) \rightarrow (q_2, \epsilon)$

$\delta(q_2, c, b) \rightarrow \delta(q_2, \epsilon)$

$\delta(q_2, c, a) \rightarrow (q_2, \epsilon)$

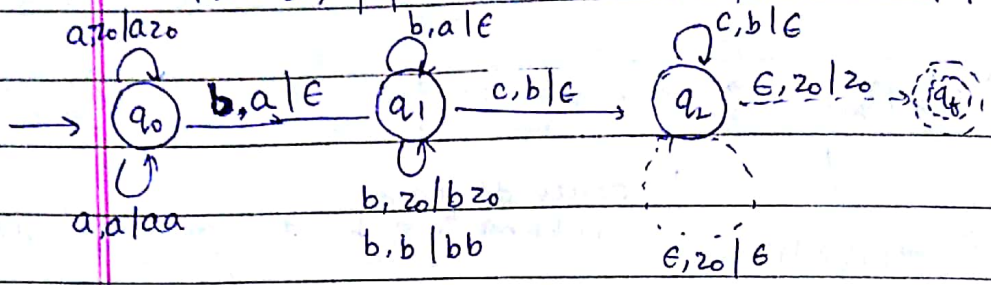
$\delta(q_2, \epsilon, Z_0) \rightarrow (q_2, \epsilon, \epsilon)$

$\delta(q_2, \epsilon, Z_0) \rightarrow (q_f, \epsilon)$

Teacher's Signature

2) $L = \{ a^i b^j c^k \mid i,j,k \geq 1 \;,\; i+k=j \}$

Push a, pop a with b, then push b & pop b with c



$\delta(q_0, a, z_0)$ includes (q_0, az_0)

$\delta(q_0, a, a)$ " (q_0, aa)

$\delta(q_0, b, a)$ " (q_1, ϵ)

$\delta(q_1, b, a)$ " (q_1, ϵ)

$\delta(q_1, b, z_0)$ " (q_1, bz_0)

$\delta(q_1, b, b)$ " (q_1, bb)

$\delta(q_1, c, b)$ " (q_2, ϵ)

$\delta(q_2, c, b)$ " (q_2, ϵ)

$\delta(q_2, \epsilon, z_0) \rightarrow (q_f, \epsilon, z_0)$ $\quad$ $\delta(q_2, \epsilon, z_0) \rightarrow (q_2, \epsilon)$

3.) Push a, Read b ⇒ pop a or read 2 b & pop 1 a.

$\delta(q_0, a, z_0)$ includes (q_0, az_0)

$\delta(q_0, a, a)$ " (q_0, aa)

1st choice $\quad \delta(q_0, b, a)$ " $\{(q_1, \epsilon), (q_0, a)\}$

pop a ; ↳ won't pope a

$\delta(q_1, b, a)$ " $\{(q_1, \epsilon), (q_2, a)\}$

$\delta(q_2, b, a)$ " (q_2, ϵ)

$\delta(q_1, \epsilon, z_0)$ " (q_f, z_0)

$\delta(q_2, \epsilon, z_0)$ " (q_f, z_0)

q_1 → for every b : popping a

q_2 → skipping for 1 b

Signature

6/

DATE : / /
PAGE NO. :

1.) 2.)

→ showing Acceptance by Final states ≡ Acceptance by Empty store

a.) 1 ⇒ 2

let : PDA A = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

with acceptance by final state

(want to construct acceptance by empty store for A)

let L = L(A)

We need to construct A' (new PDA) s.t.

L = L(A') [acceptance by empty store]

$A' = (Q', \Sigma, \Gamma, \delta', q_0, Z_0, F)$

→ Once we go to f.s., we can empty the stack for

$F = \{q_1, q_2, \ldots q_n\}$

δ' should include all transitions of δ in addⁿ to that we use the following:

$\forall q_i \in F$. $\forall \delta'(q_i, \epsilon, X)$ includes (q_{empty}, X) ∴ go to state q_{empty} & empty all the stack

$\delta'(q_{empty}, \epsilon, X)$ includes (q_{empty}, ϵ) → stack empty

⇓ Acceptance by Empty store

By construction, we see that $Q' = Q \cup \{q_{empty}\}$

$L(A') = L(A)$

b.) 2 ⇒ 1

We have : $B = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \phi)$ → final state take parameters ni nhi

B is accepting by empty store.

let L = L(B)

We need to construct B' (new PDA) s.t.

L = L(B')

Teacher's Signature

$\Gamma' = \Gamma \cup \{Z_0'\}$

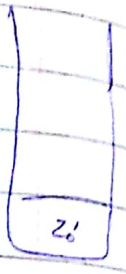
$Q' = Q \cup \{q_0', q_{final}\}$

$B' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F)$

B' accepts by final state.

→ When stack becomes empty, make it reach to F.S.

When start processing for B', put Z_0' in stack
Z_0' is initial symbol



$\delta'(q_0', \epsilon, Z_0')$ includes $\delta(q_0, Z_0 Z_0')$

from q_0, we can apply all transitions of B.

$\delta(q, \epsilon, Z_0')$ includes (q_{final}, Z_0') $\quad F = q_{final}$

$q \in Q'$ ⇓ B ka stack is empty ⇒ go to final state

* Z_0' included to check when B ka stack becomes empty.
* Moreover δ' includes all transitions of δ.

→ By construction,

$L(B) = L(B')$

Eg. $L = \{ w \in \{a,b\}^* \mid n_a(w) = n_b(w) \}$

$L = \{ w \in \{a,b\}^* \mid n_a(w) = n_b(w)$ for any prefix w' of w $n_a(w') \geq n_b(w') \}$ → Dyck Set (Dyck language)

a b a b a b

→ ( ) ( ) ( ) : well formed parenthesis

(( ))) : not " "

a a b b b

)) (((: " " "

b b b a a a

no. of open brackets ≥ no. of closed brackets in prefix of string

Teacher's Signature

PDA ban skta h ⇒ CFL h
Otherwise, nhi h

→ whenever see a → push in stack
b → pop out of stack

CFG
|
CFL —subset→ DCFL (jiska DPDA ban jaaye)
↓
PDA

for some CFL, we can draw DPDA — for all CFL, we can draw NPDA

DPDA ⊆ NPDA (*Every NPDA can't be converted into DPDA)

PDA = FA + Stack → used because it is 0 address (add. don't 'ue to be specified for del^n / ins^n, always happens on top of stack)

Transition func^n

1) DPDA : $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$ (Γ = tos)

2) NPDA $\delta : Q \times \Sigma \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$

Basic Op^ns on Stack:-

① PUSH 1

|a|b|

push kiya h

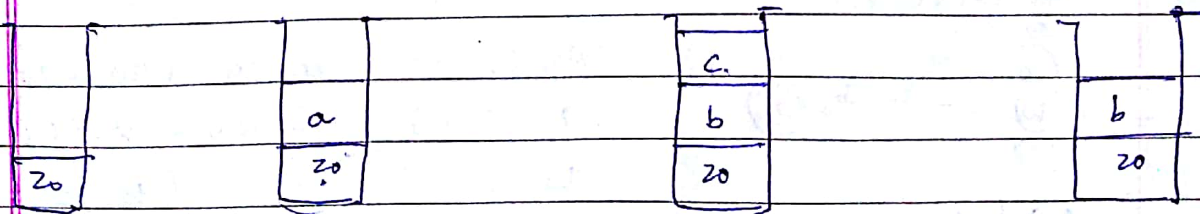
$q_i \xrightarrow{a, Z_0 / aZ_0} q_j$ $(Z_0 \rightarrow aZ_0)$

② POP

|a|

$q_i \xrightarrow{a, c / \epsilon} q_j$

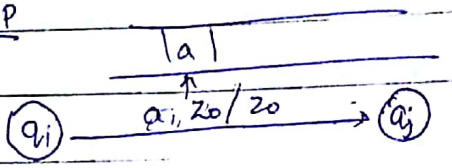
sirf tos dikhega, c pop kiya, naya status q_i pe pata chalega.

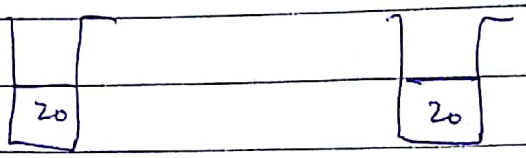


$\delta(q_i, a, Z_0) \rightarrow (q_j, aZ_0)$

$\delta(q_i, a, c) = (q_j, \epsilon)$

③ SKIP





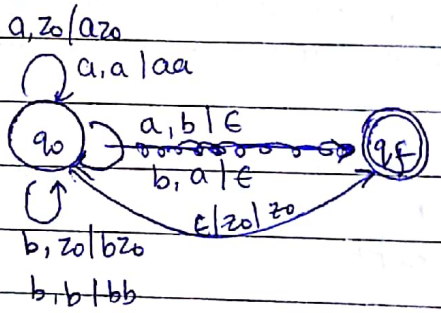
$\delta(q_i, a, z_0) = (q_j, z_0)$

Eg. $L = \{ w \in \{a,b\}^+ \mid n_a(w) = n_b(w) \}$

→ i/p 'a' : a will pop out b, else, push a

→ i/p 'b' : b will pop out a, else, push b.

aababb



(q_0, a, z_0) includes $(q_0, a z_0)$

(q_0, a, a) " (q_0, aa)

(q_0, b, z_0) " (q_0, bz_0)

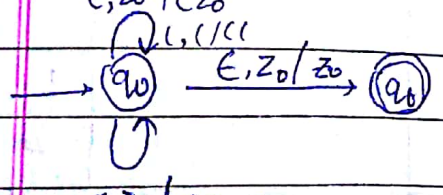
(q_0, b, b) " (q_0, bb)

(q_0, a, b) " (q_0, ϵ)

(q_0, b, a) " (q_0, ϵ)

eg. → p $(q_0, \epsilon, z_0) \longrightarrow (q_0, \epsilon)$ (q_0, ϵ, z_0) " (q_f, z_0)

Eg. : Paranthesis :



$(q_0, (, z_0)$ includes $(q_0, (z_0)$

$(q_0, (, ()$ includes $(q_0, (()$

$(q_0, (,))$ " (q_0, ϵ)

(q_0, ϵ, z_0) " (q_f, z_0)

Teacher's Signature

in $\{wwR\}$, $\{a^m b^n : m < n \leq 2m\}$: non-deterministic PDA.

Can't be accepted by DPDA

DATE : / /
PAGE NO. :

Deterministic PDA (DPDA)

PDA = $(Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

in which

(1) $\forall\ q \in Q$, $a \in \{\Sigma \cup \in\}$ $X \in \Gamma$

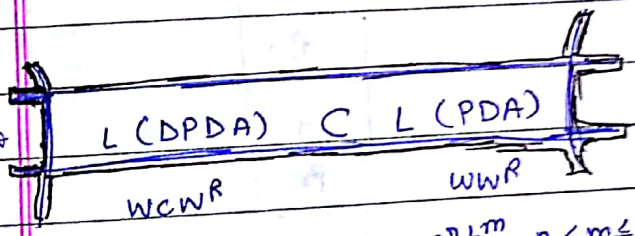
$\delta(q, a, x)$ contains at most a unique pair (q, α)

(2) If $\delta(q, \in, x)$ is non-empty, then $\delta(q, a, x)$ should be empty & vice versa → can't read $\in$ & a at same time

→ $\{wcwR\}$: whenever c is there, we know w is over

⇓ accepted

it'll be a DPDA



Any deterministic PDA can be called as non-deterministic also.

Equivalence of PDA and CFG

1) CFG ⇒ PDA
2) PDA ⇒ CFG

1) CFG ⇒ PDA

Given a CFG, $G = (V, T, P, S)$

P : $A \rightarrow \alpha$, $A \in V$, $\alpha \in (V \cup T)^*$

Construct an eq. PDA A s.t. $L(A) = L(G)$

Teacher's Signature

$A \to \gamma$

Each is in sentential form

$S \xrightarrow{\text{derives}} \alpha A \beta$. $\quad \alpha, \beta \in (V \cup T)^*$

$\to \alpha \gamma \beta$

$\to \alpha B \gamma' \beta$

$\to \alpha \gamma'' \gamma' \beta$

$\to$ aabbaa...

→ producing sentencial form by applying rules in grammar

A B

⇒ We'll construct PDA which accepts by empty store

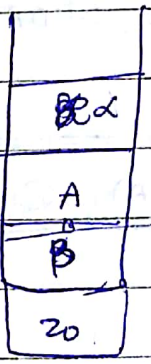
* we use CFG with ~~right~~ left most derivation.

$S \to \alpha A \beta$

α

Pop NT & push its rule.

Pop T.

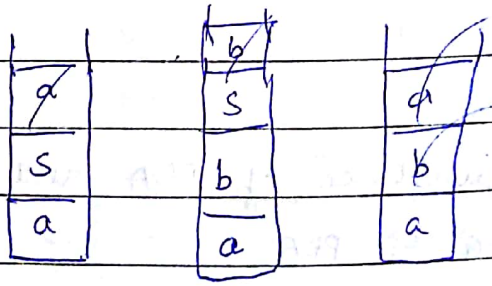


21/3/18

$S \to a S a$

$S \to b S b$

$S \to a$



| a | b | a | b | a |

(make sure i/p symbol & tos are same)

→ It'll accept by PDA.

DATE : / /
PAGE NO. :

F.S. not needed

$P = (\{q\}, T, VUT, S, q, S, \phi)$

δ transition :-

1) ∀ non-terminals (variables) $A \in V$

$\delta(q, \in, A) = \{(q, B) \mid A \to B$ is a prod^n rule in $P\}$

∈ move

2) ∀ terminals $a \in T$

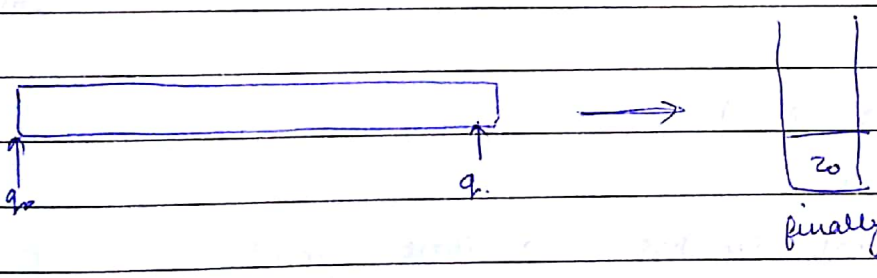
$\delta(q, a, a) = \{(q, \in)\}$

No. of rules ≡ $V + T$

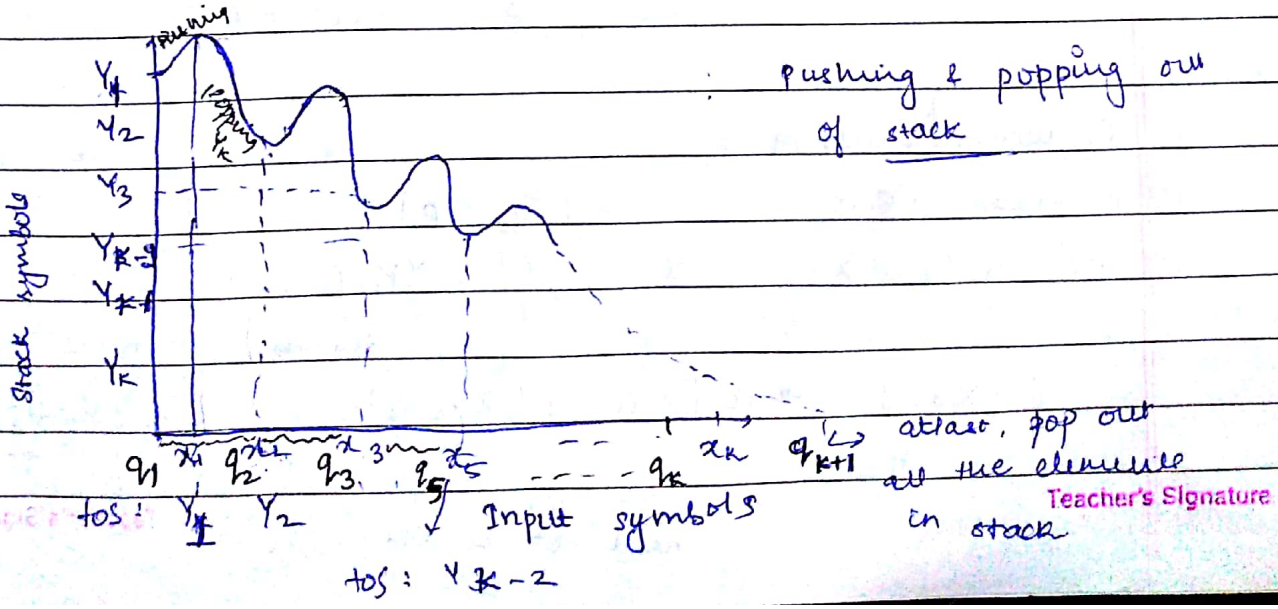
2) PDA ⟹ CFG

(PDA : acceptance by empty stack)

PDA $P = (Q, \Sigma, \Gamma, S, q_0, Z_0)$

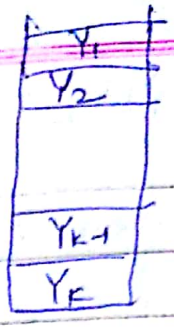


Graph :



: pushing & popping out of stack

Teacher's Signature



b/w $q_1 - q_2$: read x_1

$q_2 - q_3$: read x_2

$q_k - q_{k+1}$: read x_k

→ If I want to pop Y_k → should we popped out $Y_1, Y_2, \ldots Y_{k-1}$ symbols already.

From q_1 to q_2 : we are popping Y_1

→ state : $[q_1 Y_1 q_2]$

Likewise from q_2 to q_3 : $[q_2 Y_2 q_3]$ → Popping each symbol while reading some input symbol

→ $[pXq]$: p is a state
q is a state

$[pXq]$ is a NT to be popped out.

$[q_0 Z_0 q]$: q_0 to q. In meanwhile, pop Z_0 : Empty stack

⟹ $G = (V, \Sigma, R, S)$

V consists of

(1) Special symbol S → Start symbol

(2) All symbols of the form $[pXq]$
where $p, q \in Q$ and $X \in T$

Production rules of P

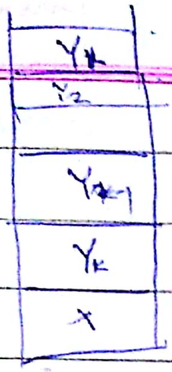
(1) $\forall$ states $p \in Q$ $S \rightarrow [q_0 Z_0 p]$ → can be any state

(2) Let $\delta(q, a, X)$ contains $(r, Y_1 Y_2 Y_3 \ldots Y_k)$
↓ pushing k symbols

$[qXr_k] \rightarrow a\,[r Y_1 r_1]\,[r_1 Y_2 r_2] \ldots [r_{k-1} Y_k r_k]$

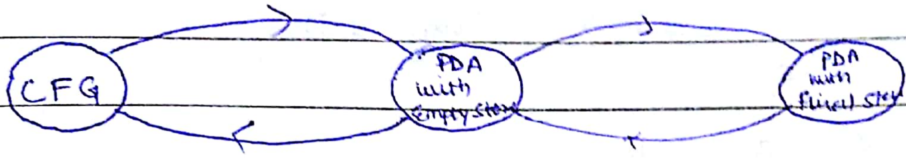
need not be the same

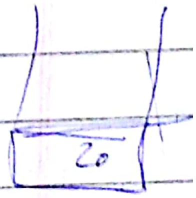
Teacher's Signature



⇒ Pop out all $Y_1, Y_2, Y_3, \ldots Y_k$

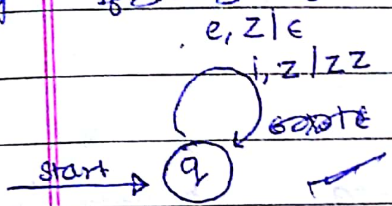
23-03-18



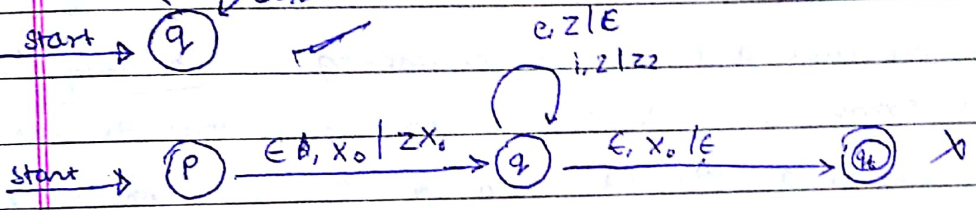


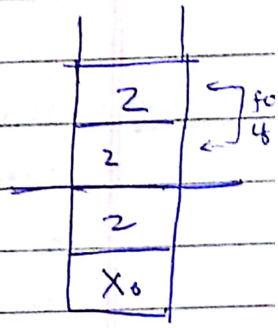
Eg. PDA : $i^n e^{n+1}$ → This is what we need

Eg. ~~If--then--else~~ i → if, e → else — pushing Z when if comes



: By empty stack





⇒ No. of e = No. of i + 1 ⇒ violates rule

PDA ⇒ CFG :

δ for PDA :
δ(q, e, z) has (q, ε)
δ(q, i, z) has (q, zz)

(n rules for all q ∈ Q) (1) S → [qZq] ↑ A

← 2nd Prodn for this

(n^2 rules) (2) [qZq] → i [qZq] [qZq] { have 2 more Z to be popped out }

(3) [qZq] → e — already popping it out, don't have to capture anything else

→ S → A
A → iAA
A → e

$[q_1 Z q_2] \to i\ [q_1 Z q_3][q_3 Z q_4]$ ↓ ~~may vary from 1 to n~~ ⇒ n^2 rules

Chomsky : $A \to BC$: 1 NT by only 1

$A \to a$ ↓ " " " "

DATE : / /

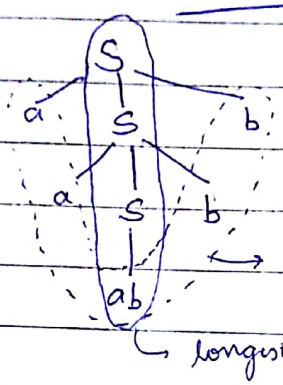
PAGE NO. :

CFG

$S \to aSb \quad \to a^n b^n$

$S \to ab$

→ Parse & Tree



→ |w| : length

↔ yield of the tree (a^3b^3)

↳ longest path

Theorem : Suppose we have a parse tree according to Chomsky Normal Form. $G = (V, T, P, S)$ and suppose that the yield of the tree is a terminal string w. If the length of the longest path is n, then

$$|w| \leq 2^{n-1}$$

28-03-18

Proof ; $n = 1 \qquad A \to a \qquad |w| = 1$

$1 \leq 2^{1-1}$ ✓

→ True for the basis $n = 1$

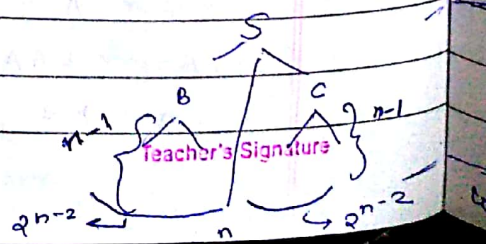
Hypothesis : result is true for $n-1$

→ $|w| \leq 2^{n-2}$

Show for n : $(n > 1)$ → need to start in form (can't have $A \to a$ $(n=1)$)

$S \to BC$ (Chomsky)

length of longest path = n



$|w| \leq 2^{n-2} + 2^{n-2}$

$|w| \leq 2^{n-1}$

Teacher's Signature

Pumping Lemma for CFL

Let L be a CFL. Then there exists a constant n s.t. if $z \in L$ s.t. $|z| \geq n$, then we can write

$z = uvwxy$, subject to following conditions :

(1) $|vwx| \leq n$

(2) $|vx| \neq \epsilon$

(3) $\forall\ i \geq 0 \quad uv^iwx^iy \in L$ (v & x can be pumped any no. of times)

* grammar should be in Chomsky Normal form
  (↑ed by 1 terminal only at 1 step)
  (If language is empty / ϵ ⇒ CNF can't exist for L)
  ↓ lemma always satisfied if this case is considered.

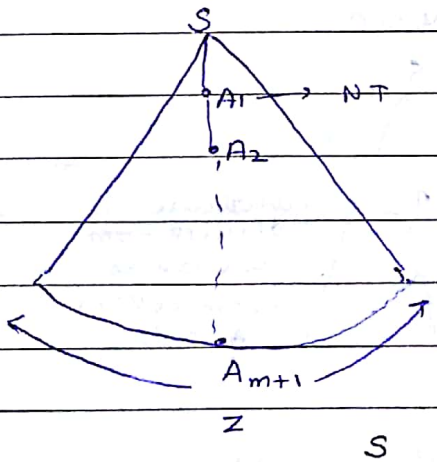
Suppose G is a CFG in CNF. Let G has m variables.

Let $n = 2^m$

Let $|z| \geq n = 2^m$

Using previous theorem,

longest path should have length $\geq (m+1)$ in parse tree of z



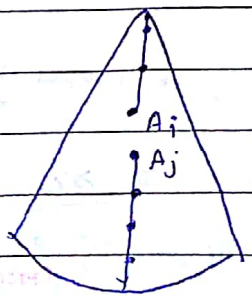
m variables, have

atleast m+1 NT

⇓

Using Pigeonhole,

atleast 2 NT are same

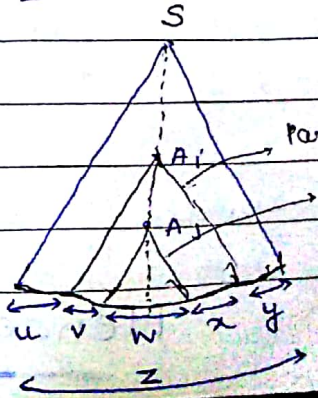


$A_i = A_j$

yield of A_i = yield of B + C

can't have longest path $> n-1$

⇒



Parse tree from A_i

" " " A_j

Teacher's Signature

1.) Max. length b/w A_i & A_j = m

→ yield : $2^{m-1} \leq n \rightarrow 2^m$

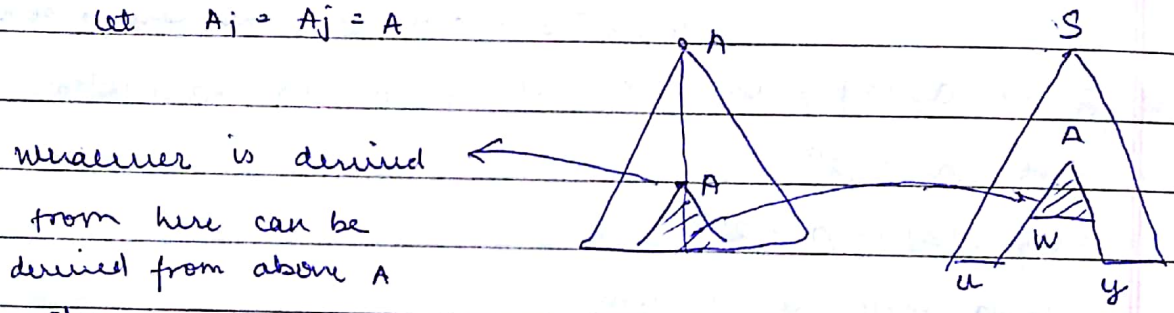
→ $|vwx| \leq n$

2.) either v or x should not be ∈.

if $vx = \in$ (both are ∈) ⇒ both yields of A_i & A_j are same which is not possible because we have atleast 1 derivation (atleast of length 1) in b/w.

3.) A_i & A_j are one & the same

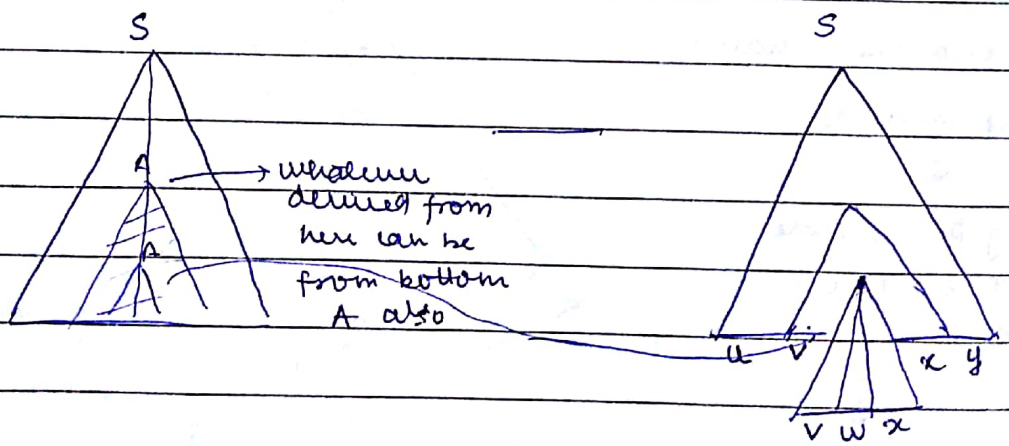
Let $A_j = A_j = A$



whatever is derived from here can be derived from above A also

→ $uwy \in L$

→ Vice - versa



→ $uv^2wx^2y \in L$

It looses hold on #x & #v

likewise, $uv^iwx^iy \in L \quad \forall\, i \geq 0$

Hence Proved

Teacher's Signature

8

a babble cc

a a

DATE: / /
PAGE NO.:

Eg.

(1) $L = \{ a^n b^n c^n \mid n \geq 1 \}$ → not CFL

Let us assume L is context free language.

by pumping lemma, $\exists$ m s.t.

$z \in L$ st $|z| \geq m$ then $z = uvwxy$ &

all 3 conditions are satisfied.

Let us take the string $a^m b^m c^m$

$a^m b^m c^m = uvwxy$

s.t. $|vwx| \leq m$

→ v can consist of only a's & b's (not c's)
(comb^n of)

x " " b's & c's.

→ v can consist of only a's / only b's / only c's / a's & b's / or b's & c's
(can't have all 3 symbols since $|vx| \leq m$)

→ x can also consist of same values.

⇒ $v = a^i \quad x = a^j$

$(i=0) \rightarrow uv^i wx^i y : \rightarrow uwy = a^{m-i-j} b^m c^m \notin L$ ✗ Not true

violates cond^n of Pumping Lemma
($i \& j \neq 0$: cond^n $vx \neq \epsilon$)

→ Not a CFL.

↳ a's & b's : $(a^i b^j)^i$: can't be accepted in given language (a can't come after b)

→ have to tell for all comb^ns

show for 1 & show it is similar for other.

Teacher's Signature

Using pumping lemma, p.t. following L are not CFL.

(2) $L = \{ a^i b^j c^i d^j \}$

(3) $L = \{ ww \mid w \in \{a, b\}^* \}$

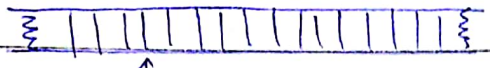
(2) $a^i b^j c^i d^j = \cancel{xy}\ uvwxy$

why maths not included in Nobel prize

→ Every problem can be related to decision problem

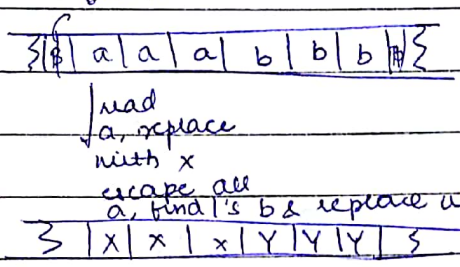
TURING MACHINES

→ Infinite tape :



↑ ↳ can move to the right as well as to the left → use of flexibility
Δ replacement of symbol is also allowed

Eg. $L = a^n b^n$



→ Here, counting is done using matching.

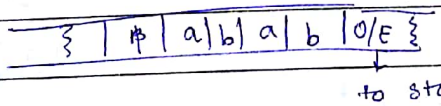
4/4/18 →

Turing machine can be treated as :

(1) Accepting Device (like FA, PDA) → automata → Decision Problem (Y/N)

(2) I/O Device

count b odd/even



TM = $(Q, \Sigma, \Gamma, \delta, q_0, \phi, F)$

Q = set of finite no. of states

Σ : input alphabet

Γ : tape alphabet (includes Σ also) $\Sigma \subset \Gamma$

q_0 : start state

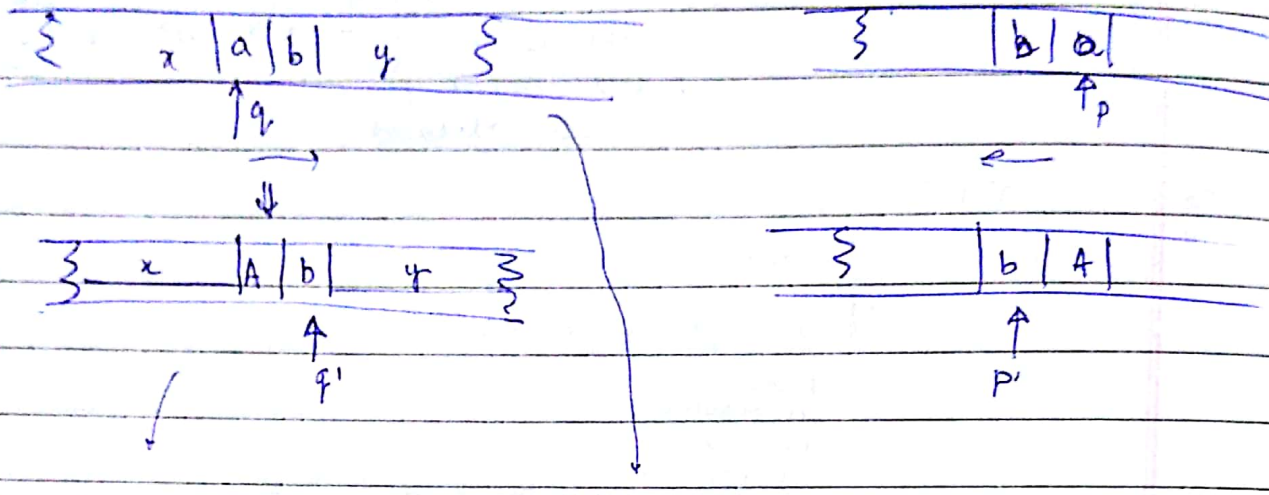
ϕ : blank symbol

F : set of final states → make sense only in case of Accepting Device

Non-deterministic $\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L, R\}}$ ↳ moving Left / Right

Teacher's Signature

$\delta(q, a)$ includes (q', A, R) , $\delta(p, a)$ includes (p', A, L)



Instantaneous Description : $xAq'by$

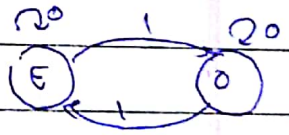
ID : $xqaby$

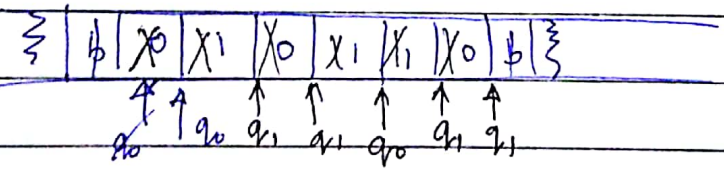
$xAq'by \longrightarrow xqaby$ after applying above rule.

Turing Machine as I/O device

1) Parity checker : check whether even no. of 0's or odd no. of 1's

$\Sigma = \{0, 1\}$





X : already read that symbol

read 1 → move next state

(q_1, b) → odd

(q_0, b) → even

$\delta(q_0, 0)$ includes $\delta(q_0, X, R)$

$\delta(q_0, 1)$ includes $\delta(q_1, X, R)$

$\delta(q_1, 0)$ includes $\delta(q_1, X, R)$

$\delta(q_1, 1)$ includes $\delta(q_0, X, R)$

$\delta(q_0, b)$ " $(Halt, E, -)$ ← ans

⇓ Halt

$\delta(q_1, b)$ " $(Halt, 0, -)$

→ Halt always in this case. But in some cases, it may not halt if not accepted.

→ procedure need not halt in case of all i/p (Turing machine) (O/P Yes: Halt, No: may not halt)

algo needs to " " " ".

Perfect no. : n is perfect if sum of all divisors except no. = n

eg. $6 = 3 + 2 + 1$

$28 = 1 + 2 + 4 + 7 + 14$

Q: Is there a perfect no. > n ? ⇒ Check $n+1 \xrightarrow{NO} n+2 \xrightarrow{No} n+3 \ldots$

⇓ may not end also

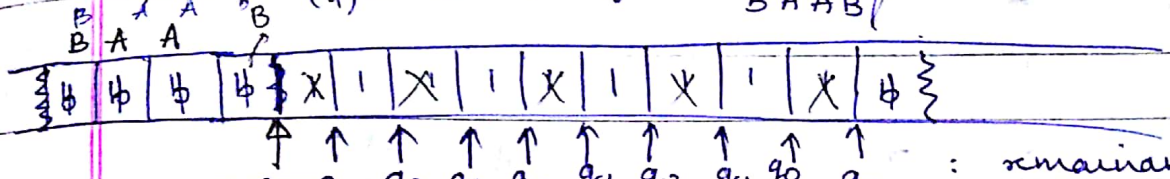
may give procedure but not algo.

DATE : / /
PAGE NO. :

Example 3 Unary to Binary Conversion:

9 1's ⇒ (9) ⇒ Binary : 1001 (q_0)

BAAB



read all symbols : if odd no. of 1's ⇒ 1 will come at last Posn

2	9	
2	4	1
2	2	0
2	1	0
	0	1

for quotient : alternate X : count 1's

Transitions :

$\delta(q_0, 1)$ = (q_1, X, R)

$\delta(q_1, 1)$ = $(q_0, 1, R)$

$\delta(q_1, \sqcup)$ = (q_2, ϕ, L) → q_1 से $\sqcup$ ⇒ odd no. of 1's ⇒ '1'

→ to add remainder

$\delta(q_2, 1/X)$ = $(q_2, 1/X, L)$ → don't do anything, just move

$\delta(q_2, \sqcup)$ = (q_4, B, R) → Put B

left wala

$\delta(q_4, Z_1)$ = (q_4, Z_1, R) , $Z_1 \in \{A, B, Z\}$

$\delta(q_4, 1)$ = (q_1, X, R)

$\delta(q_0, \sqcup)$ = $(q_3, \sqcup, L)$ → even no. of 1's ⇒ A

$\delta(q_3, 1/X)$ = $(q_3, 1/X, L)$ → just move to left

$\delta(q_3, A/B)$ = $(q_3, A/B, L)$

$\delta(q_3, Z)$ = (q_3, Z, L) ; $Z \in \{1, X, A, B\}$

$\delta(q_3, \sqcup)$ = (q_4, A, R) → Put A

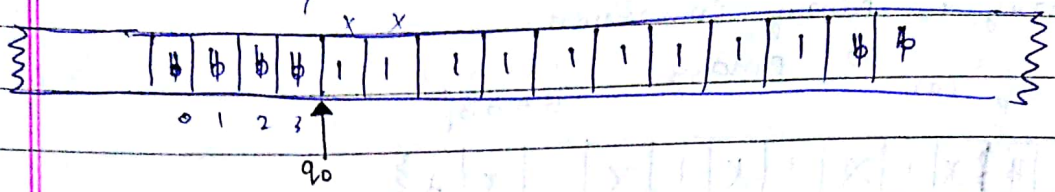
$\delta(q_0, X)$ = (q_0, X, R)

$\delta(q_1, X)$ = (q_1, X, R)

$\delta(q_4, \sqcup)$ = (Halt, -, -)

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

Other way : treat this as a binary counter : incrementing binary no by 1



Encounter 1st 1 → B at ϕ (3) (1)

2nd 1 → move L : see B → B → A (10)

(2) ϕ → B

3rd 1 → move L : see A → A → B (11)

q_0 → search 1's (A/B)

q_1 → move to left & write A/B

if B → B → A

go left ϕ → A

go left → ϕ → B (100)

$\delta(q_0, 1) = (q_1, X, L)$ ⇒ you want to count no. of 1's

$\delta(q_1, \phi) = (q_0, B, R)$

$\delta(q_0, X, R) = (q_0, X, R)$ → skip

$\delta(q_1, X) = (q_1, X, L)$

$\delta(q_1, B) = (q_1, A, L)$

~~$\delta(q_1, \phi) = (q_1, B, R)$~~ $\delta(q_0, \phi) = (\text{Halt}, -, -)$

$\delta(q_1, A) = (q_0, B, R)$

→ Only 2 states are used here.

(1) v/s (2). { like Running time }

No. of cells ∴ No. of cells you have to traverse

~~ will be less is more (skip ones) (sub ones)

① : ↑ ϕ. ② : ↓ → { like memory }

Teacher's Signature

Half Infinite Tape ≡ Full Infinite Tape

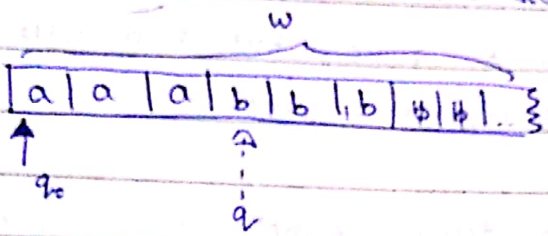
DATE: / /
PAGE NO.:

Turing Machine as an Accepting Device

→ We assume tape is finite on left side, infinite on right side

→ output is from left ~~state~~ most side

→ head is at left side in the starting.



$= (Q, \Sigma, \Gamma, \delta, q_0, ␢, F)$

Instantaneous Description : aaa q bbb { in state q & going to read b }

Starting " " : $q_0 w$

$q_0 w \vdash^* \alpha_1 q_f \alpha_2$, $\alpha_1, \alpha_2 \in \Gamma^*$ $w \in \Sigma^*$, $q_f \in F$

→ If this is possible

⇓

w is accepted by TM

→ tape alphabets ≢ input alphabets

*. The head may not even need all the string ($\alpha_1 q_f \alpha_2$).

↳ If you want to read all i/p symbols : you've to take care

$L(M) : \{ w \mid q_0 w \vdash^* \alpha_1 q_f \alpha_2$ s.t. $\alpha_1, \alpha_2 \in \Gamma^*$, $q_f \in F \}$ ($w \in \Sigma^*$)

↳ Type 0 language/ Recursively Enumerable language (RE) ↳ can be counted

→ using same set of transition table again & again

Teacher's Signature

11/4/18

PAGE NO. :

Turing Machine :- accesses i/p (x) & gives o/p ($f(x)$)

↓

defines a func^n

T.M. not only defines a func^n, but also tells how that func^n is derived → T.M. is beyond a func^n ↓ gives algo also

b Set of all T.M. which defines func^{n}s → Turing computable Func^{n}s

Features of T.M.

(1) Finite Control (can store some info.)

↓

set of states (eg. even / odd no. : 2 states q_0 and q_1)

(2) Marking some symbols on tape is like storing some info.

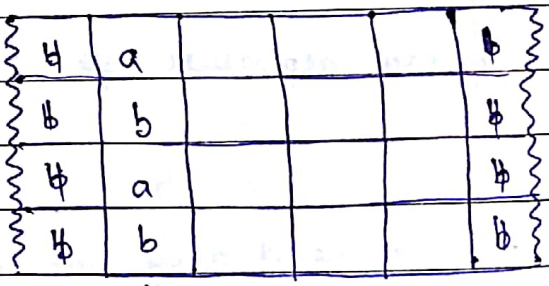
(Eg. : Quotient : 'x' se 1 ko alternatively replace kr rhe hue)

(3)

Single Track T.M.

(1 Head)

similar to



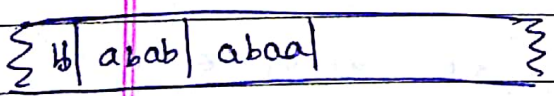
Multi track T.M.

(different from multitape)

(1 Head)

↓

can read all symbols in that col.



* Multitrack can be converted into single track

↳ single track is always multitrack

* Both are equivalent

MT TM ≡ ST TM

↓

good when don't want to have chunsiles

Teacher's Signature

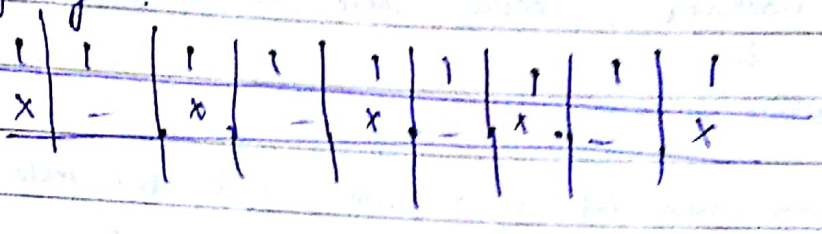
Accepting : semi infinite tape

I/O : infinite tape

PAGE NO.:

DATE : / /

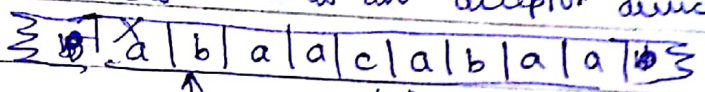
eg : → In unary eg. :



(3) : Single Track TM ⟺ Multitrack TM

(4) : Checking of Symbols :

eg. : → WCW as an acceptor device $w \in \Sigma$



→ q_a (have read a : go to q_a)

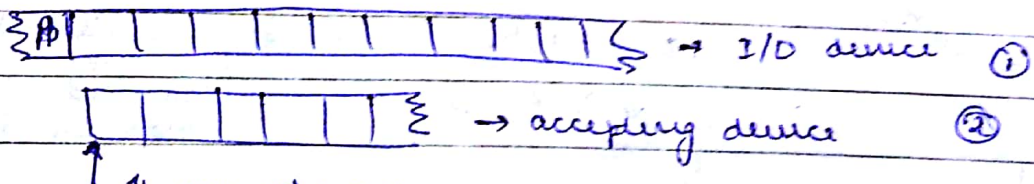
after c, if get a : go to q_a' then start moving left until x

Halt : when only C & all other x.

go to left till get x : move R
if get C : also check if all are x in Right → Accept (Halt)

(5)

Generalised TM

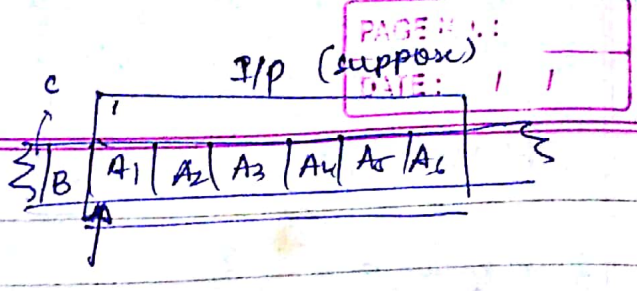


If you move left : TM will abruptly stop : won't accept string

whatever you can do in (2) ⇒ you can do in (1)

in (1) → Put a marker before front so that TM won't go beyond that cell. By that way : it becomes one side finite

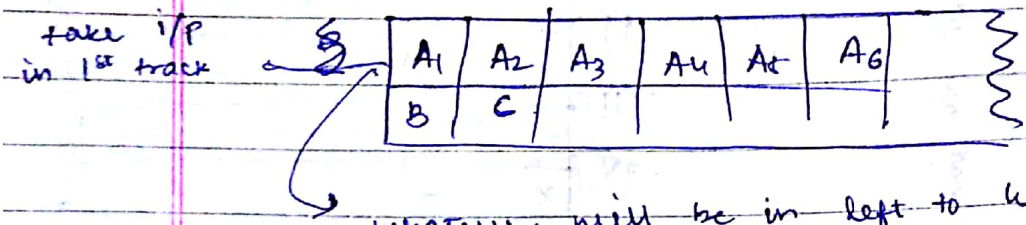
but vice versa doesn't hold. (1 ⊄ 2)



For nice verse

② → Take 2 track

take I/P in 1st track



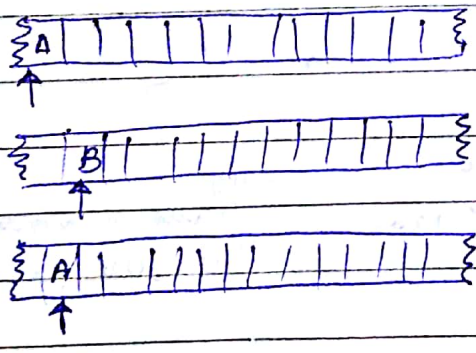
whatever will be in left to head : bottom track
right : top track

but in infinite tape : (R → R) {B, C}

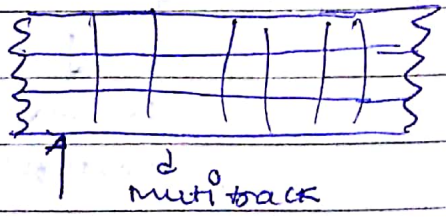
but here : (L → R) {B, C}

1.) One-sided Infinite tape ≡ Both sided Infinite tape

2) Multitape TM ⟺ Single Tape Turing Machine



: Each tape has its own tape header



↳ All 3 heads are independent

↳ State will be same

$\delta(q_1, A, B, A) \longrightarrow (q', A', B', A'', R, L, R)$

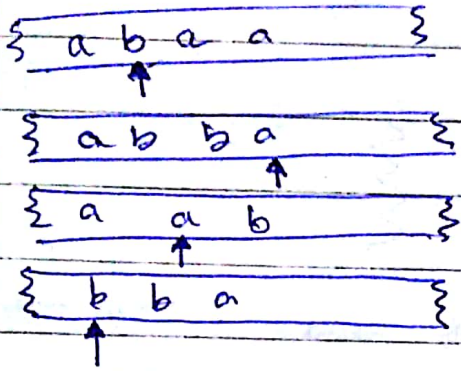
q_1 : state is same

R, L, R : define movement ~~at each state~~ for each tape

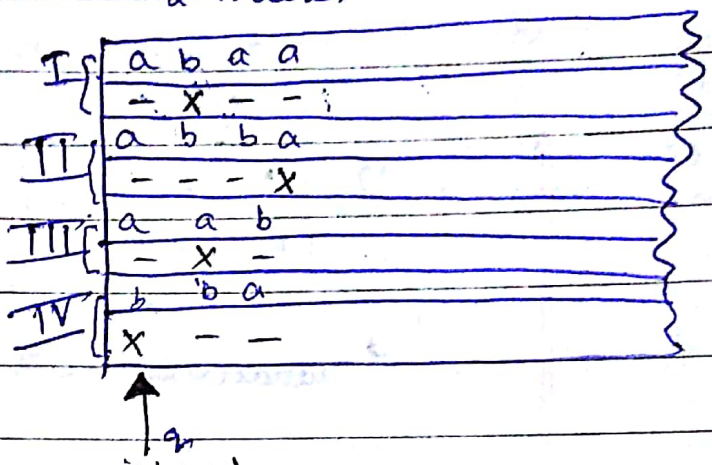
PAGE NO.:
DATE:

Multitrack 1

Multitape



For every tape, I'll have a tracks



if head is here, it has to see where is X.

top track : i/p symbol

bottom track : mark X where header is

$(q, \underline{b}\ \underline{a}\ \underline{a}\ \underline{b}\ \underline{a}, 4)$

where X is — counter, how many are filled

Now, it has to simulate after reading b, what 1st tape will read. — & so on for each tape

Suppose 2nd tape : a → A & moves to left

$\delta(q,$

For 1 transition :

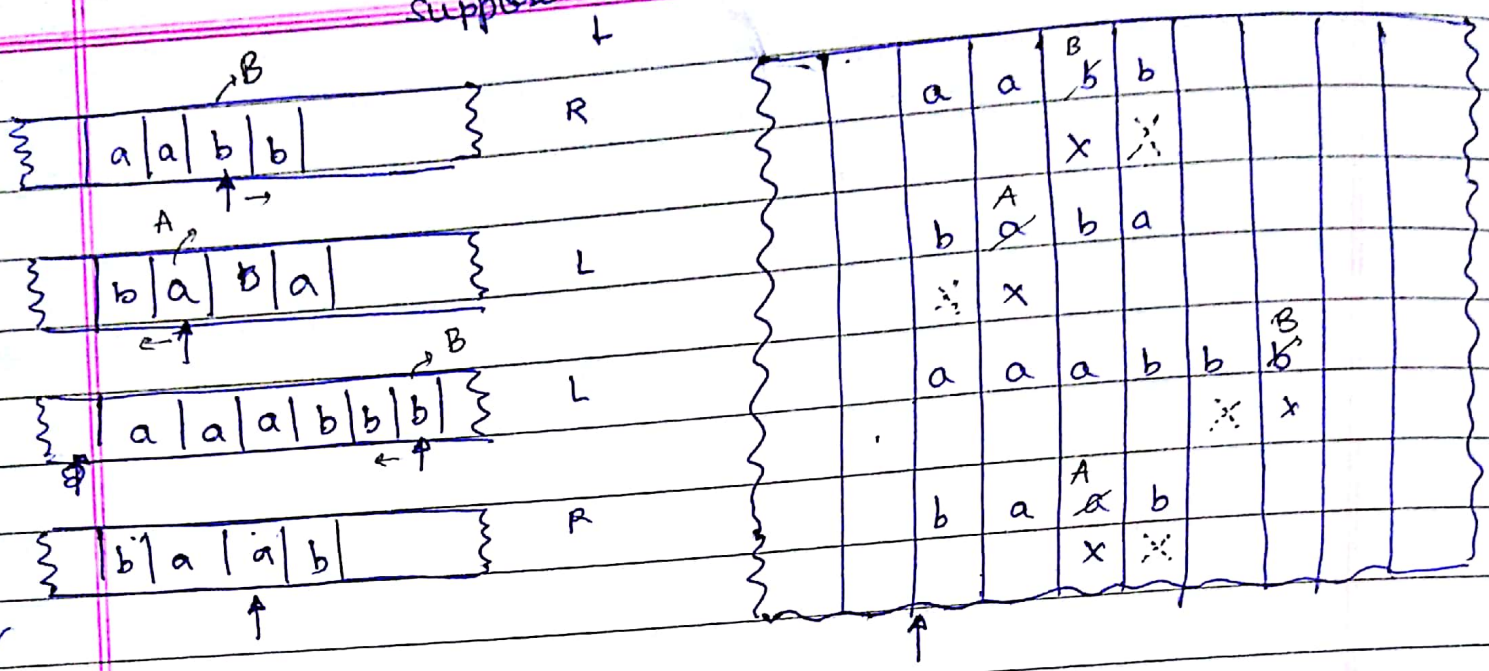
$\delta(q, \_\_\_\_\_) \longrightarrow (\_\_\_\_\_)$

for this, you may have to move full length of 1st tape

may have to move back to left full length

4/18

Suppose movement:



$\delta(q, baba) = q'(BABA, RLLR)$

First, it need to move through all strings to get X.

$\delta(q, \underline{b}\ \underline{a}\ \underline{b}\ \underline{a}, \underline{\ })$

counter (increment as you get X)

1st pass : L → R (to get X)

2nd pass : R → L (replace b → B & move to right) → 1st tape

⇓

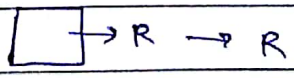
1st X will come for 3rd tape : have to store which one is finished also

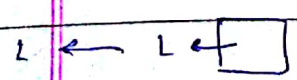
↓

replace by B & move to L

→ 2 passes.

⇒ If there are n movements in multitape, no. of movements in multitrack → "Polynomial" (not exponential)





Distance = 2 4

DATE: / /
PAGE NO.:

Deterministic T.M. and Non-Deterministic T.M.

→ NTM : $\delta(q_1, a) = \{ (q', A, R)^1, (q'', B, L)^2, (r, C, R)^3 \}$

$\delta(p, b) = \{ (p', B, R)^1, (r, B, L)^2 \}$

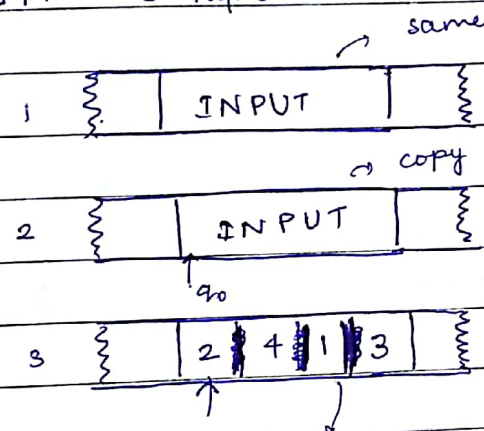
{ | INPUT | }

$

Have to simulate this NTM using DTM.

↳ find transition which has max^m tuples. → label them

↳ DTM : 3 tapes



→ when you're in q_0, you've to apply 2nd tuple for that → q_1
From q_1, you've to apply 4th choice

if # : TM will stop.

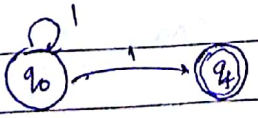
if reach q_f :→ accepted.

If you don't accept : generate another no. & check if it accepts or not
erase 2nd & 3rd & generate another no. ⇒ have to repeat many times for same input

↳ No. of transitions will be : "Exponential"

NTM ⟺ DTM (Power is same)
(Transitions no. of will be exponential)

→ $\delta(q_0, \#) = \{ (q_0, 1, R), (q_1, 1, R) \}$ $q_1 \in F$



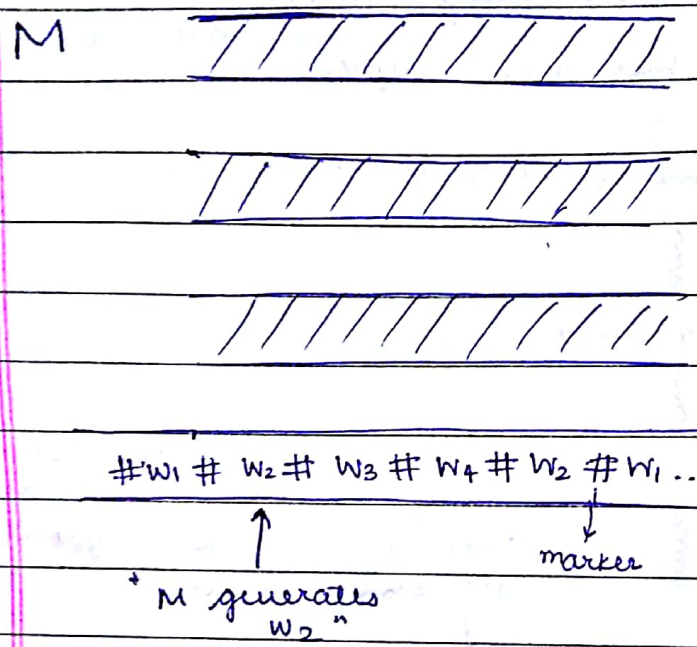
: can generate any no. of 1's (unary no.)

Teacher's Signature

→ if NTM ends : DTM also ends

if NTM goes to ∞ loop : DTM also goes to ∞ loop

3) Turing Machine as a Generating Device : somewhat similar to i/o device



if ∃ i s.t. $w_i \in$ Output tape, we say that 'M generates w_i' ($w_i \in G(M)$)

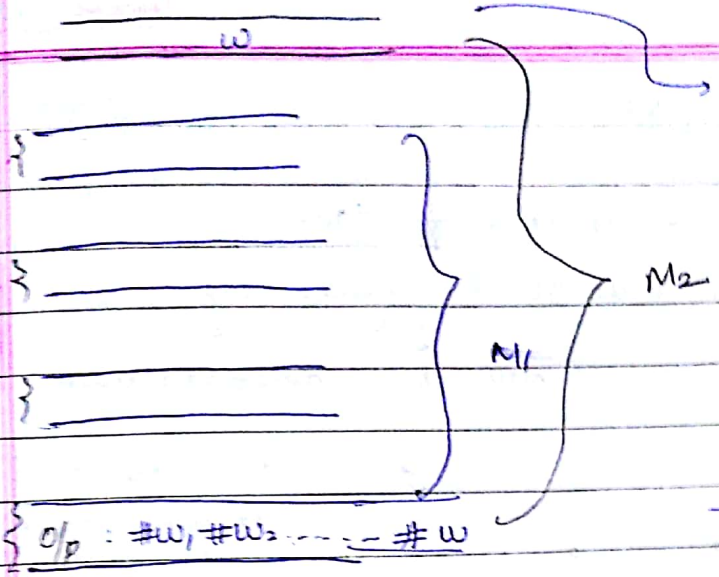
Thm: $L = T(M_1)$: (language accepted by T.M. M_1) for some T.M. M_1 iff $L = G(M_2)$ for some TM M_2.

(language accepted by TM = language generated by TM)

$L = G(M_1)$ —①

$L = T(M_2)$ —②

1) ① ⇒ ② ; name generating TM.



→ have i/p in 1st tape

→ Once it generates, checks if $w_1 == w$ — Yes, Halt

↓ No

move to w_2

- when generate in M1 → can get accepted in M2

2) ② ⟹ ①

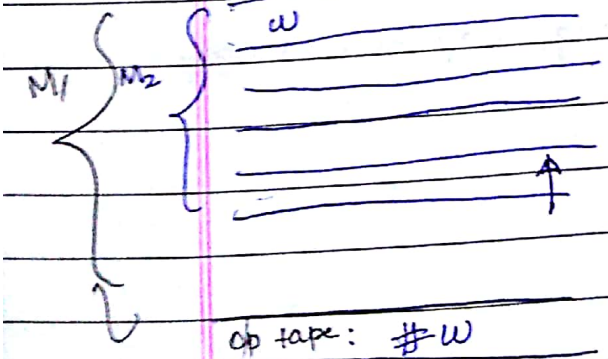
non-recursive (if accepted : Halt
not accepted : Infinite loop may go on

TM accepts that language when it halts anyway, (even if not accepted)

a) Let us assume it is recursive language L.

L is a recursive language if ∃ a TM, M s.t. L(M) = L and L halts on all inputs

Suppose : M_2 is recursive



Teacher's Signature

b) M_2 is non-recursive

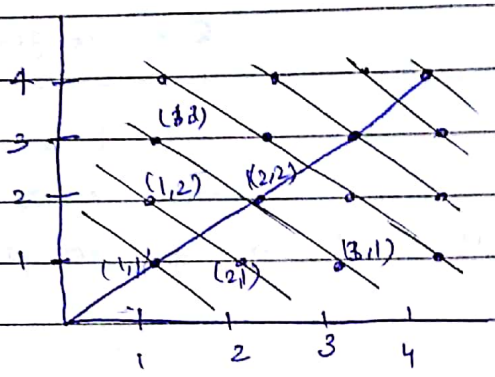
if accepted : can write in o/p tape

rejected : will go in infinite loop

now to check

will use : "diagonalisation"

like proving Q is uncountably infinite



diagonalise acc to sum

Here,

include

$\langle i, j \rangle$: generate pair (i, j) (randomly)

can use many no. here

will take w_i (i^{th} string accepted by language)

will generate it for j steps

(generate w_i within j steps)

if not generated in j steps

(can change i & j accordingly)

M_1

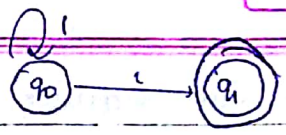
M_2

O/p tape

include

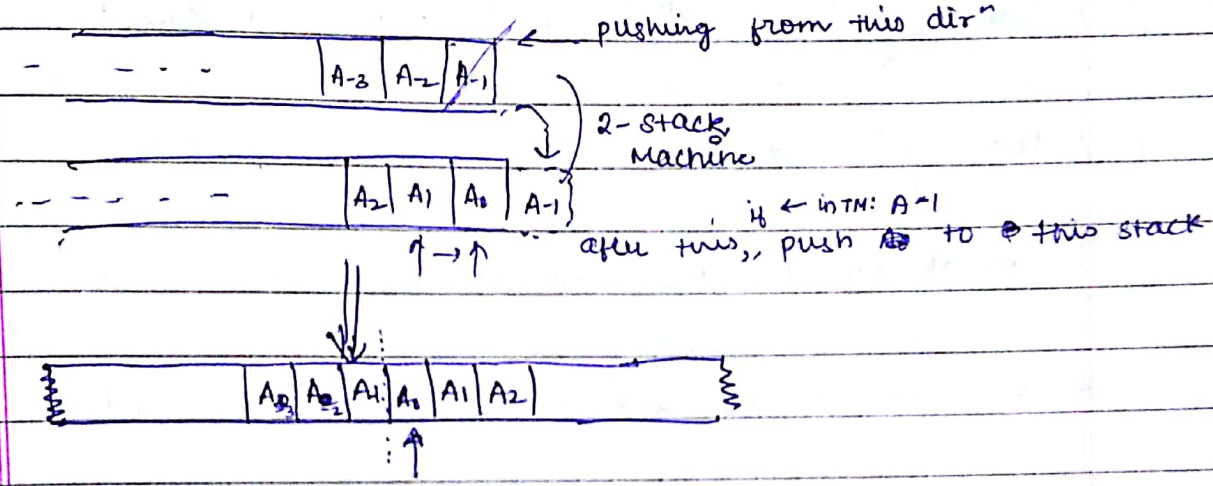
1 $\Longleftrightarrow$ 2

Teacher's Signature

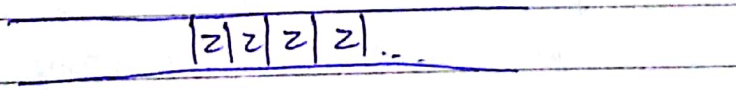
→ can generate i & j easily using : 

Restrictive Models of TM

1.) 2-Stack Machine ≡ TM

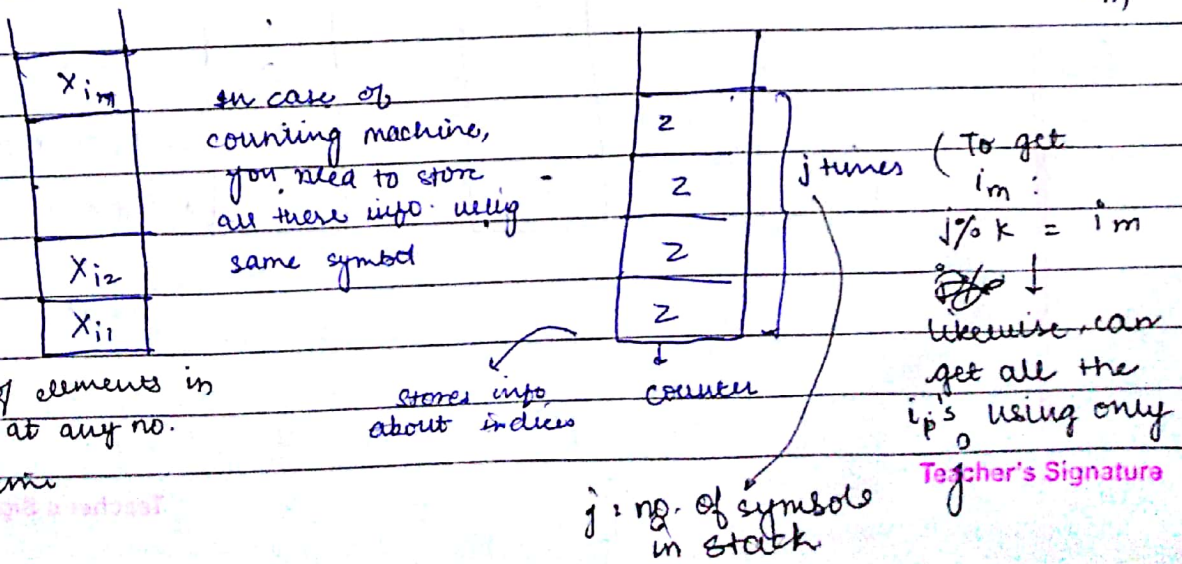


2.) Can have 1 symbol in stack and still get the T.M. using 4 counters. : Counting Machine (counting no. of z's)



1-stack by 2-Counters (2-stack : 4 counters → TM)

$\Gamma = \{x_1, x_2, \ldots, x_{k-1}\}$, $j = k^{m-1} i_1 + \ldots + k^2 i_{m-2} + k i_{m-1} + i_m$



Teacher's Signature

↳ we will require another counter in case of push/pop opn.

↓

stack with only 1 symbol

13/4/18

$j \rightarrow$ unique representation

$j \% K = i_m$

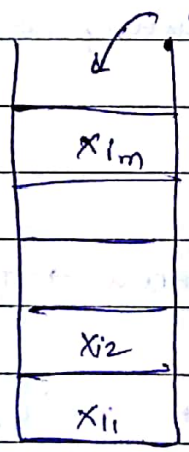
$\text{Quotient}\left(\frac{j}{K}\right) = K^{m-2} i_1 + K^{m-3} i_2 + \cdots + K i_{m-2} + i_{m-1}$

$\text{Rem}\left(\frac{j}{K}\right) = i_m$

divide by K, Rem will be i_{m-1}

→ You can get all indices $i_m, i_{m-1}, i_{m-2}, \ldots, i_1$

↳ X_r needs to be pushed

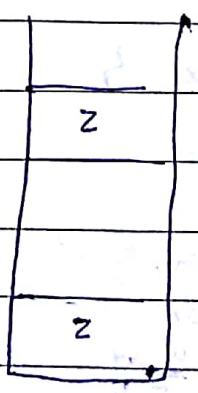


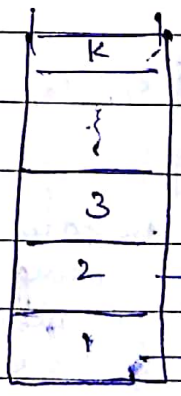
Represent' for this

$j \rightarrow jK + r$

you need to know how many z you need to put

for 1 z → you need to put K z's j times





Teacher's Signature

DATE : / /
PAGE NO. :

↳ X_{im} needs to be popped

↳ divide by k.

1 stack ⇔ 2 ctrs

T.M. ⇐ 2 stack ⇔ 4 ctrs

⇒ 4 ctrs → T.M.

Hence Proved

{ Even 2 counters can form a T.M. }

** For every T.M. , ∃ an equivalent T.M. that uses only 2 alphabets (symbols)

↓

Base of binary (Anything can be written as a comb^n of 0's & 1's)

⇓

becomes a language over 0 & 1.

Restrictions

1) 2 - stack Machine = T.M.

2) 4 - counter = T.M.

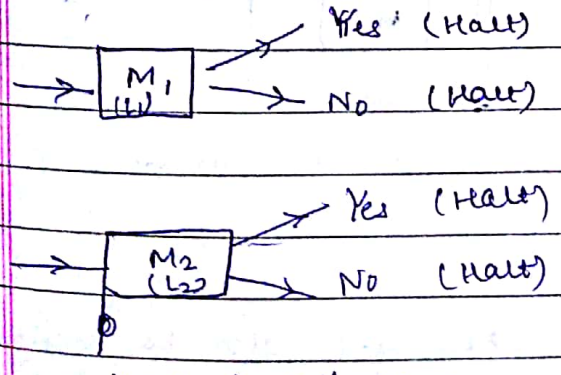
3) T.M. can use 2 alphabets symbols

4) T.M. without rewriting

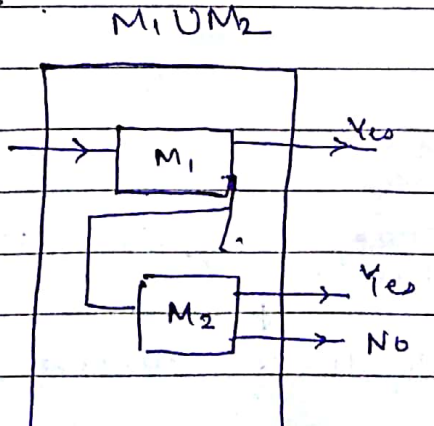
Recursive Sets

L_1 , L_2 : Recursive languages

1) Is $L_1 \cup L_2$ Recursive language ?



Halt in all case



Halt in all case

↳ $L_1 \cup L_2$ ⇐ (Recursive)

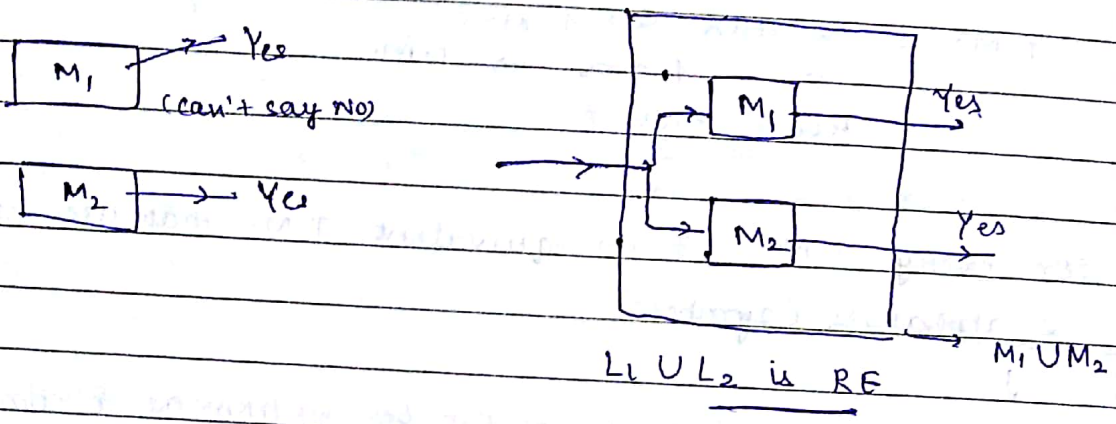
Teacher's Signature

Objective Quiz
20 Ques – 40 marks
Closed Book

DATE: / /
PAGE NO.:

if accepted : Halt
If not " : may not Halt

2) Is L_1 & L_2 Recursively Enumerate (RE), $L_1 \cup L_2$ = ??
(Every recursive is recursively enumerable but not vice versa)

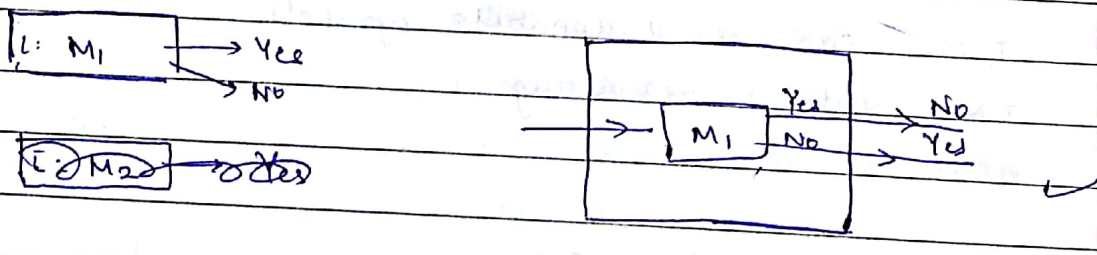


$L_1 \cup L_2$ is RE

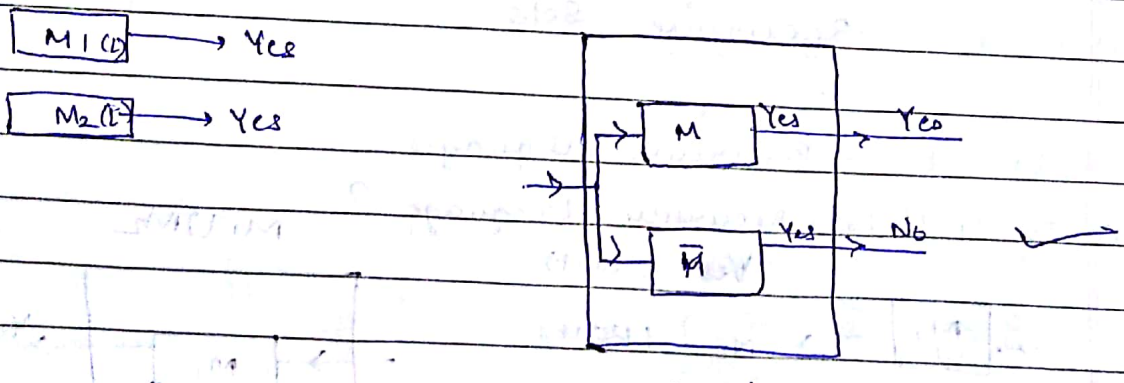
3) If L : recursive, is $\bar{L}$ also recursive ?
4) If L & $\bar{L}$ are RE, then L is recursive ?

20/9/18

3)



4)



⇒ L is recursive

Check for : $L_1 \cap L_2$, $L_1 - L_2$
L_1, L_2 : Recursive

↓

If L_1 is recursive, its subset will also be recursive.

Teacher's Signature

DATE : / /
PAGE NO. :

Some definitions

1. Problem :

→ Given a graph, does it have Hamiltonian cycle?

I/P : G is graph

v_1 ~~vertices~~ is a vertex of graph

Problem 1 : Does ∃ a Hamiltonian circuit starting & ending with v_1 (Yes/NO)

Decision problem (Yes/NO)

If graph has Hamiltonian cycle → it's " graph.

Problem 2 : Find the Hamiltonian circuit ~~in~~ G starting & with v_1 (Exact path)

General Problem (Decision + some other problem)

2. Instance (of a problem) : taking specific eg. (of graph)

: Instance of graph

→ Consider problem 1 : Halt with Y/N. It'll end ⇒

decidable : TM that you can design that decision problem will always end with Y or N.

Can convert in Binary form :

T.M. ✓ | 0 ... 1 | 0 .... 1 | 1 ... 0 | Yes & Halt, accept O/P corresponding to Y

Input — Problem — Output

✗ | | | 0 1 | don't " ... & Halt NO

NO

This T.M. becomes recursive

Teacher's Signature

T.M. : decidable

Accept / don't accept

Yes / No : O/p (L is accepted or not)

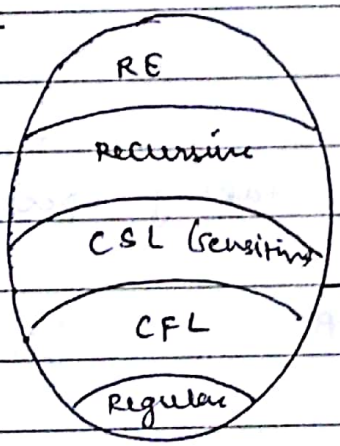
↳ Halts in all cases

↳ Suppose for Yes → Halts (T.M.)

NO → goes to ∞ loop

→ **Undecidable** (Recursively Enumerable)

Hierarchy:



→ LBA (Linear bounded Automata)

⇓

TM with finite tape

→ linearly bounded : tape size can grow by linear func^n on i/p

You can't ~~decide~~ design an algorithm which decides whether another algorithm halts or not : Halt Problem

↓

~~Undecidable~~

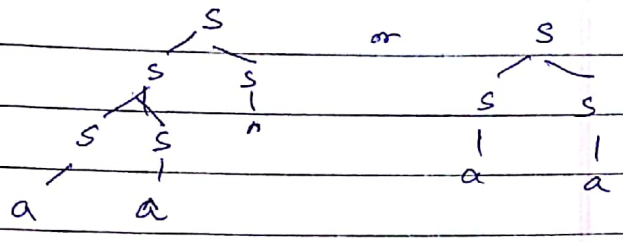
Syllabus for Quiz : Till today

DATE : / /
PAGE NO. :

Ambiguity

↳ $S \to aSb$
$S \to ab$
⇒ Unambiguous → n-1 times → 1 time (to get $a^n b^n$)

↳ $S \to SS$
$S \to a$
⇒ ambiguous



Ambiguity of CFL ⇒ Undecidable problem (AMB)

→ If it is ambiguous ⇒ it'll say Yes
But can't say NO : Undecidable.

G is a CFG
Whether G is ambiguous or not : Undecidable

Reducability :

P_{known} → Undecidable (know that it's undecidable)
$P_{unknown}$ → TO prove (by reducing Pknown to Punknown)

Reduction method — Instance of P_{known} —Reduce it / Convert it→ Instance of Punknown → Unknown ①

If you're able to show this :

↳ If ~~$P_{unknown}$ → Decidable~~ ~~⇒ P_{known} → decidable~~
~~if ① exists~~

$P_{known} \sim P_{unknown}$ (reduce Pknown to Punknown)
↓
Undecidable ⇒ Undecidable

Teacher's Signature