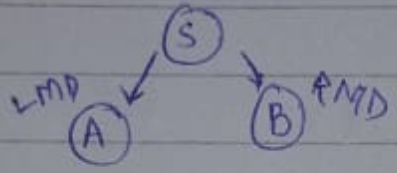
8/Apr/19 (99)

→ S → AB
A → aaA | λ
B → Bb | λ

* Grammar is ambiguous b/c we have both LMD & RMD.

S ⇒ AB



→ Required Grammars
- Right Recursive.
- Unambiguous.
- Deterministic.

→ CFG is represented by PDA.

→ IF_STMT → if EXPR then STMT
| if EXPR then STMT else IF_STMT.

9/Apr/19

(100)

"Simplification of CFG's".

→ To achive simplified CFG's we will use following steps.

1. Remove nullable variables
2. Remove unit-productions
3. Remove useless variables.

} // Substitution Rule.

eg. $A \to xBz$ →①
$B \to y_1$

Substitute $B \to y_1$ in production (1).
$A \to xy_1z$

eg $S \to aB$
$A \to aaA$
$A \to abBc$
$B \to aA$.
$B \to b$.

$S \to ab \mid aaA$
$A \to aaA \mid abbc \mid abaAc$.

① Remove nullable production :-

λ - production : $A \to \lambda$.

eg $A \to B$
$B \to C$
$C \to \lambda$

} // Nullable production.
(Null production)

eg $S \to aMb$
$M \to aMb$
$M \to \lambda$

} // Nullable variable.
(Null - production).

$S \to ab \mid aMb$
$M \to ab \mid aMb$.

② Removing a unit production :-

Unit production : $A \to B$.

eg $S \to aA$
$A \to a$
$A \to B$
$B \to A$
$B \to bb$

$S \to aA \mid aB$
$A \to a$
$B \to bb$.

(3) Removing a Useless Production:

Useless Production:

$S \to aSb$

$S \to \lambda$

$S \to A$

$A \to aA$ → useless. / non-ending

eg $S \to A$

$A \to aA$

$A \to \lambda$

$B \to bA$ → unreachable.

String $\in I^*$

String $\in (NT+T)^*$

"Normal Forms of CFG's"

→ Chomsky Normal Grammar (CNF)

$A \to BC$ — 2 non-terminal

$A \to a$ — 1 terminal.

* More powerful.

eg $S \to \Lambda$

* Chomsky Normal Forms are good for parsing and proving theorems
  - parsing → evaluating a structure → valid / invalid.

→ Greinbach Normal Form:

$A \to a V_1 V_2 \cdots V_K$

non-terminal goes to single terminal followed by a non-terminals ....

eg

$S \to abSb$

$S \to aa$

$S \to a T_b S T_b$

$S \to a T_a$

$T_a \to a$

* Goods for parsing.

CFG $\xrightarrow{\text{easy}}$ CNF
CFG $\xrightarrow{\text{difficult}}$ GNF
} // good for parsing.

11 | Apr | 19

⊢ : Push Down Automata : ⊣
(PDA).

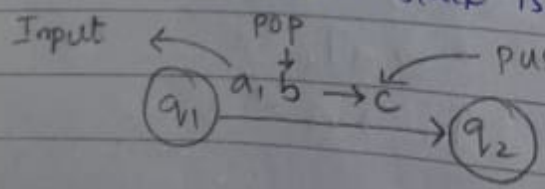
→ Input String
→ FA
→ Stack

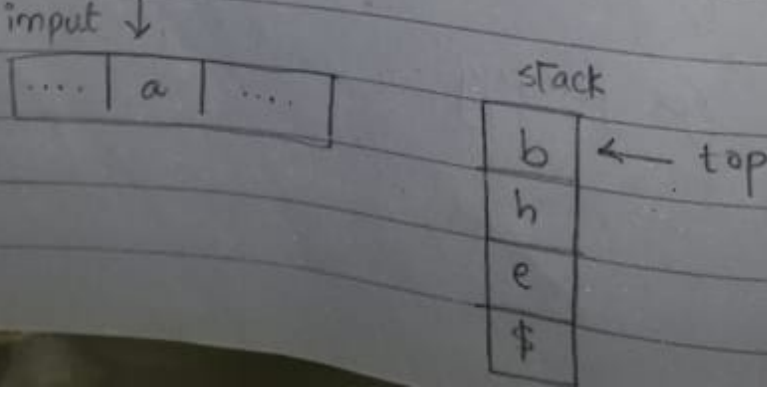
PDA
NPDA — DPDA

* NPDA is more powerful then DPDA.

* Initial Stack Symbol :=

→ stack head = $
-- to prevent underflow.
-- indicates stack is empty.





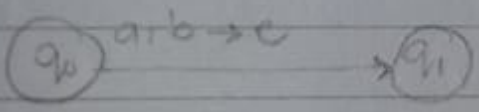
PDA:

$M(Q, \Sigma, \Gamma, \delta, q_0, z, F)$.

where Q = states
Σ = input alphabet
Γ = stack alphabet
δ = transition function
q_0 = initial state
z = stack start symbol
F = Accept states

$\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$

eg



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

eg $S \rightarrow aSb$

a
S
b
$

← top of stack

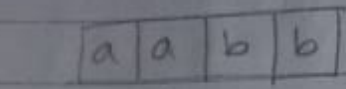
eg PDA for $a^n b^n$, $n \geq 0$.

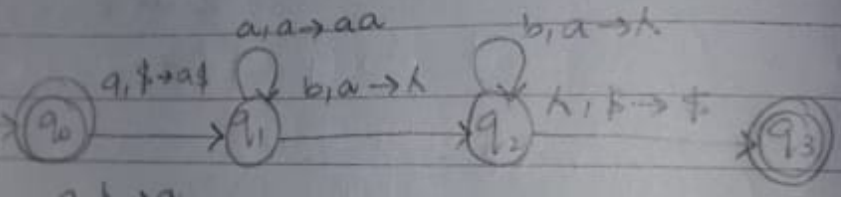
$L = \{\lambda, ab, aabb, \ldots\}$

① Jb tk 'a' mile, push kren & jb 'b' mile to pop kren. if stack is empty ($), it means a = b.

aaa bbb

push pop





* Acceptance of a string:

A string is accepted if there is a computation such that:
All the input is consumed &
The last state is an accepting state.

NOTE: We dont care about stack at accepting state.

eg $V \cdot V^R$ → Palindrome.

Odd Palindrome: $W\,C\,W^R$

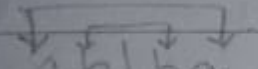
Where $C \in \{a, b\}$

$W \in \{a, b\}^*$

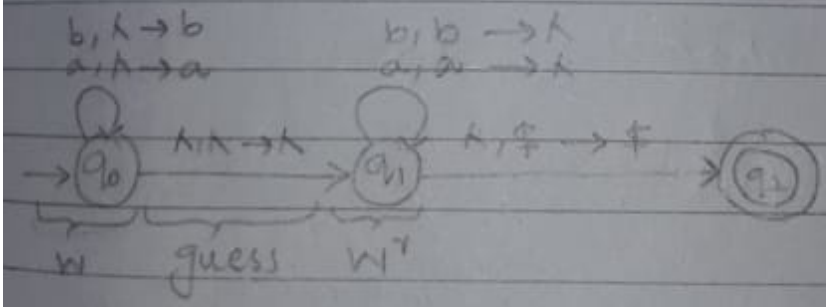
Even palindrome: $W \cdot W^R$

where $W \in \{a, b\}^*$

$L = \{\wedge, aa, bb, abba, baab, \ldots\}$

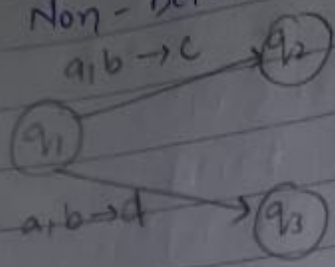


→ push till centre then pop

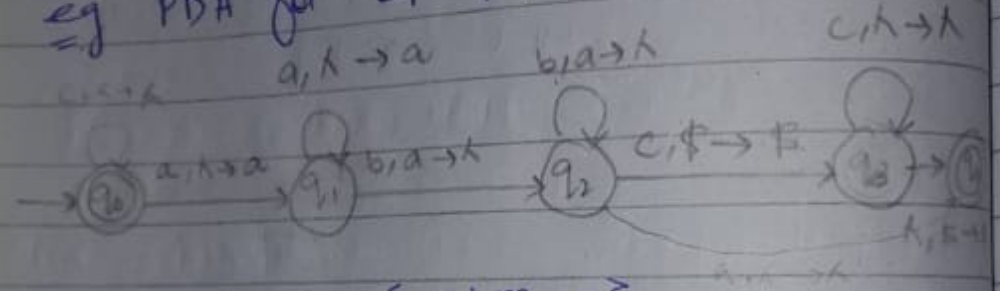


* There isn't any DPDA for this language.

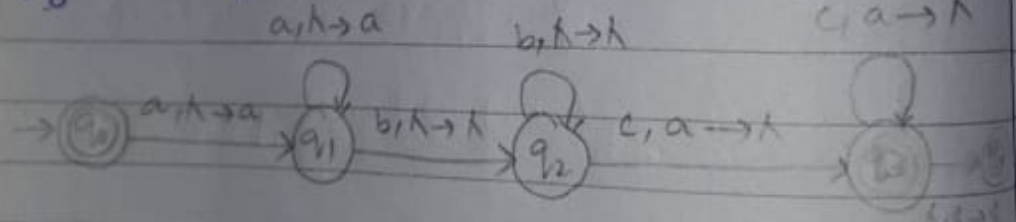
Non - Determinism.



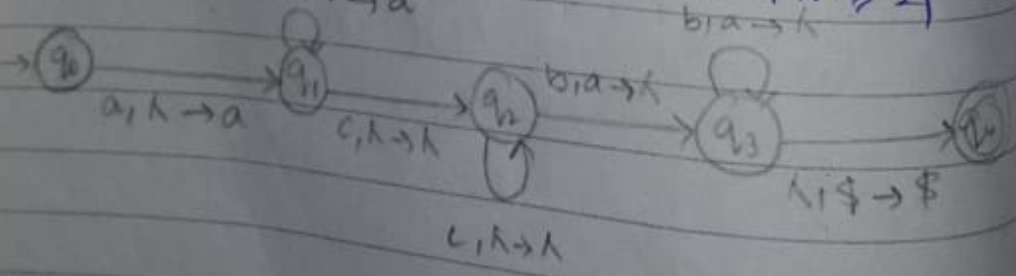
eg PDA for $L_1 = \{a^n b^n c^m\}$



eg PDA for $L = \{a^n b^m c^n\}$

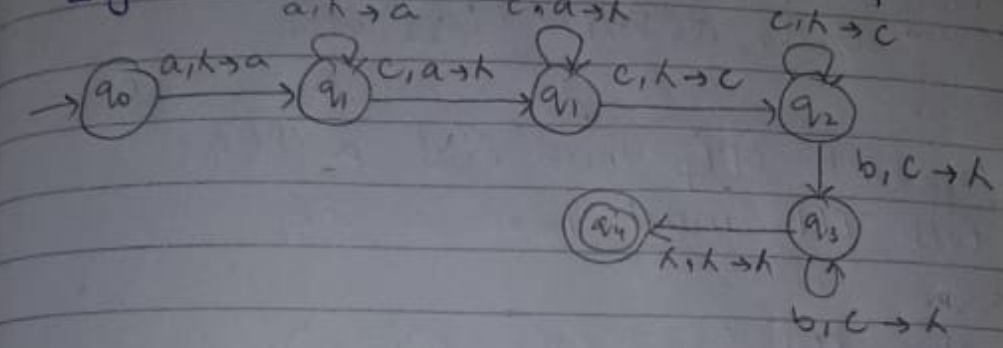


eg PDA for $L = \{a^n c^m b^n, n, m \geq 1\}$

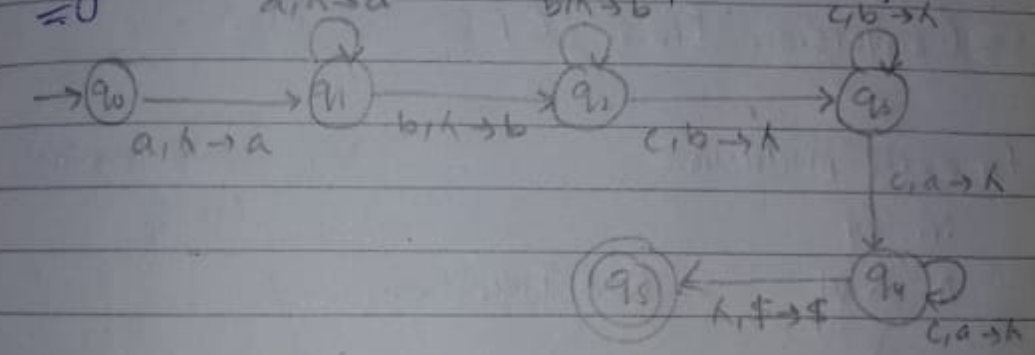


$a^n c^n . c^m b^m$

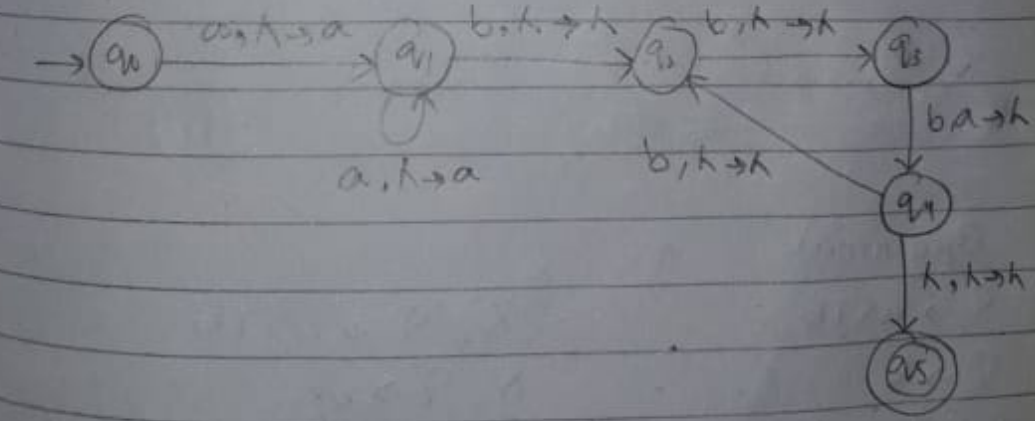
eg $L = \{ a^n c^{n+m} b^m \mid n, m \geq 1 \}$



eg $L = \{ a^n b^m c^{n+m} \mid n, m \geq 1 \}$



eg $L = \{ a^n b^{3n} \mid n \geq 1 \}$



Transition function :
$\delta(q_1, a, W_1) = \{(q_2, W_2)\}$.

* for each CFL there exist a PDA
* Each CFL must have a CFG.

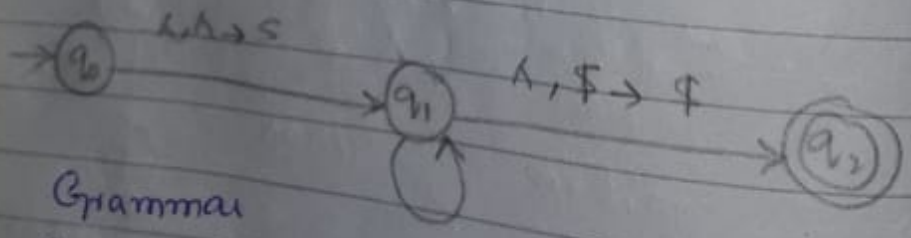
Non CFL → eg: $a^n b^n c^n$.

Conversion from CFG to PDA :-

For each production in G
$A \to W$
↓
$\lambda, A \to W$

For each terminal in G
a
↓
$a, a \to \lambda$

Add Transitions



Grammar
$S \to aSTb$
$S \to b$
$T \to Ta$
$T \to \lambda$

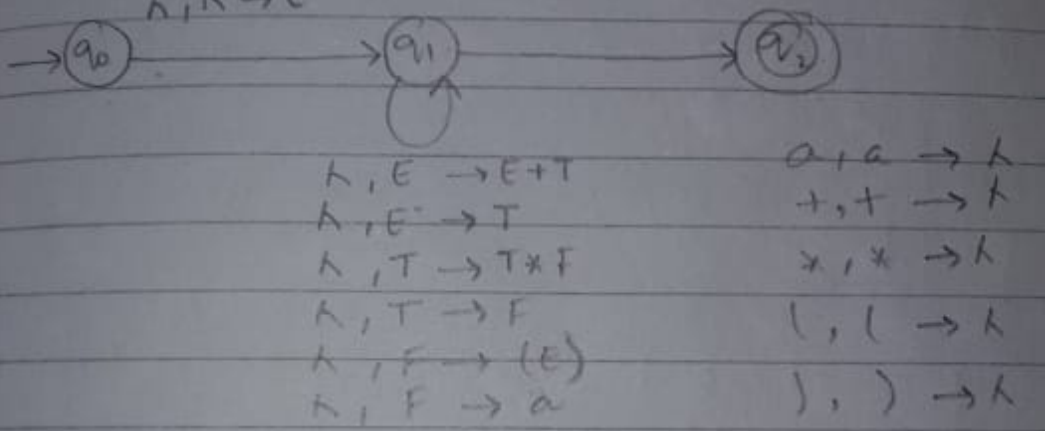
$\lambda, S \to aSTb$
$\lambda, S \to b$
$\lambda, T \to Ta$
$\lambda, T \to \lambda$

$a, a \to \lambda$
$b, b \to \lambda$

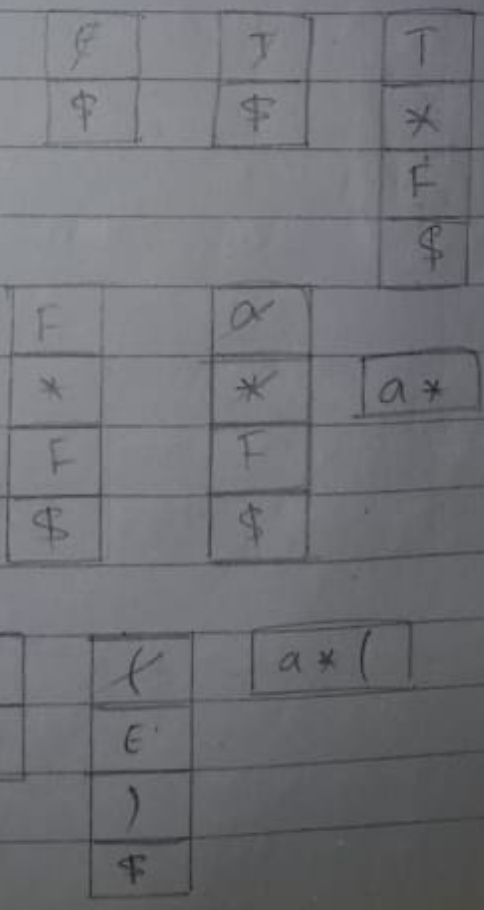
16/Apr/19

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a$

Read $a*(a+a)$.



$E \rightarrow T$
$\rightarrow T*F$
$\rightarrow F*F$
$\rightarrow a*F$
$\rightarrow a*(E)$
$\rightarrow a*(E+T)$
$\rightarrow a*(T+F)$
$\rightarrow a*(F+a)$
$\rightarrow a*(a+a)$



E
+
T
)
$

T
+
T
)
$

F
+
T
)
$

a
+
T
)
$

a*(a+

T
)
$

F
)
$

a
)
$

a*(a+a)

Properties of the CFL :-

① Positive closure for CFL :-

The operation that cause the language to be in CFL is called positive closure.

- If L_1 & L_2 are context free then $L_1 \cup L_2$ is also CFL.

i.e : L_1 & L_2 are closed under union.

eg $L_1 = \{a^n b^n\}$ $\quad S_1 \to a S_1 b \mid \lambda$

$L_2 = \{WW^R\}$ $\quad S_2 \to aS_2a \mid bS_2b \mid \lambda$

$L_1 \cup L_2 = \{a^n b^n\} \cup \{WW^R\}$

$S \to S_1 \mid S_2$ ← represents union.

- If L_1 & L_2 are context free then there concatenation is also CFL.

i.e L_1 & L_2 are closed under concatenation

eg $L_1 \cdot L_2 = \{a^n b^n\}\{W \cdot W^R\}$

$S \to S_1 S_2$.

- If a language is context free then its * operation is also CFL.

eg $L = \{a\}$ $\quad S \to a$

$L^* = \{\lambda, a, aa, \dots\}$

$S \to aS \mid \lambda$

eg $L = a^n b^n$. $\quad S \to aSb \mid \lambda$

$L^* = (a^n b^n)^*$

$S \to aSb \mid \lambda$

$S_1 \to SS_1 \mid \lambda$

② Negative Closure for CFL:
The operation that cause the language not to be in CFL.

- L_1 & L_2 are context free but $L_1 \cap L_2$ is not CFL.

eg $L_1 = \{a^n b^n c^m\}$ $= CFL$

$S \to AC$

$A \to aAb \mid \lambda$

$C \to cC \mid \lambda$

$L_2 = \{a^n b^m c^m\}$ = CFL.

$S \to AB$

$A \to aA \mid \lambda$

$B \to bBc \mid \lambda$

$L_1 \cap L_2 = \{a^n b^n c^n\}$. $\neq$ CFL

- If L is context free, its complement is not necessarily CFL.

$\overline{L_1 \cup L_2} = L_1 \cap L_2 = \{a^n b^n c^n\} \Rightarrow$? CFL

Note: $\boxed{CFL \cap RL = CFL}$

Proof:

CFL $\cap$ RL = CFL

↓ PDA ↓ DFA ↓ PDA

$M_1 \Rightarrow$ PDA $\quad M_2 \Rightarrow$ DFA

$\lambda, b \to c$

$(q_{11}) \longrightarrow (q_{12})$

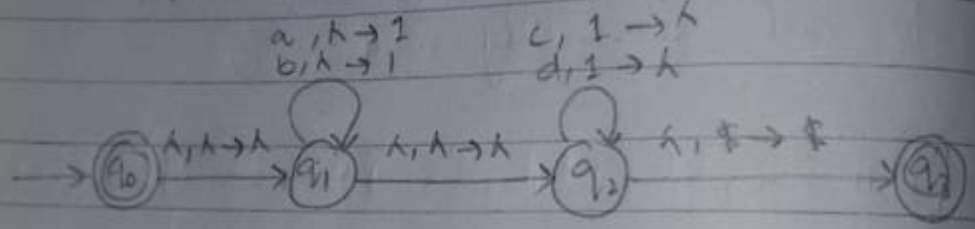
* initial of M_1 + initial of M_2 = M
* final of M_1 + final of M_2 = M

eg $L_1 = \{ w_1 w_2 : |w_1| = |w_2| , w_1 \in \{a,b\}^*, w_2 \in \{c,d\}^* \}$.

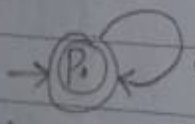
Push 1's → $(a+b)^*$. POP 1's → $(c+d)^*$

PDA ⇒ M1

$a, \lambda \to 1$
$b, \lambda \to 1$
$c, 1 \to \lambda$
$d, 1 \to \lambda$

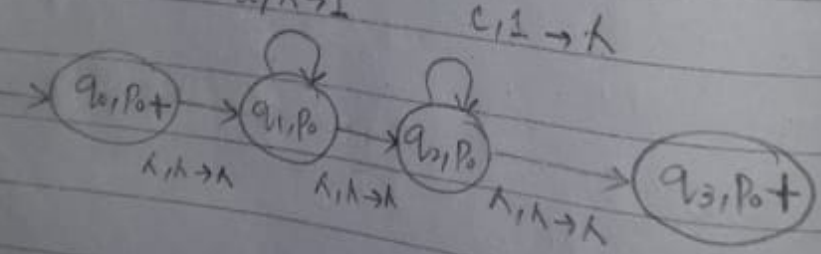


DFA ⇒ M2



PDA ⇒ M

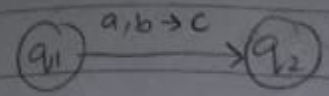
$(a+b)^*(c+d)^* \cap (a+c)^* \Rightarrow a^n c^n$



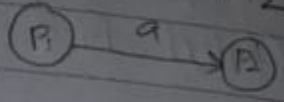
18/Apr/19

1

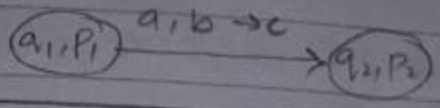
PDA = M1



DFA = M2

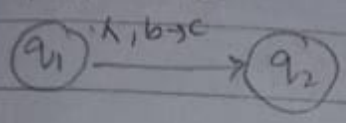


PDA → M



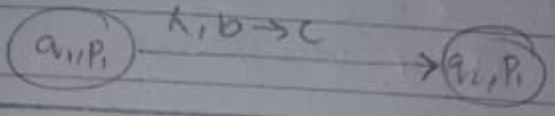
2

PDA = M1



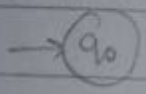
DFA = M2

PDA → M

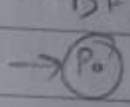


3

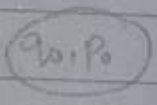
PDA = M1



DFA = M2



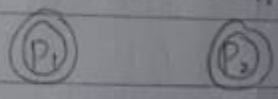
PDA → M



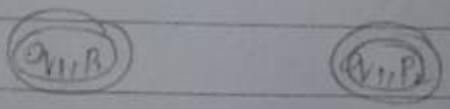
4

PDA = M1

DFA = M2



PDA → M



5

22/Apr/19 Pumping Lemma For Context Free Languages

* Pumping Lemma is used to prove non-CFL
* Used for infinite language.
* One non-terminal that can be repeated multiple times.

$$W = u \cdot v \cdot w \cdot x \cdot y \cdot z$$

① $W = u(v)^i w x (y)^i z \in L (\forall i \geq 0)$
② $|vxy| \leq m$
Where $|W| \geq m$.
③ $|vy| \neq \varepsilon$

Let p = # of production x

eg#01 $a^n b^n c^n$

Let $m = 12$.
$W = a^{12} b^{12} c^{12} \geq m$.

$= a^6 \cdot a^6 b^3 c^3 \cdot b^6 c^{12}$
$u \quad vxy \quad z$
$|vxy| \leq m$ & $|v.y| \neq \varepsilon$

$= (a^6)(a^6)^i \cdot b^3 (b^3)^i \cdot (b^6 c^{12})$

i=0 $\Rightarrow$ $a^6 \cdot b^3 b^6 c^{12} \notin L$

23/Apr/19

① Let L be in CFL & there exist a string of pumping length 'm'.

$m = P + 1$

where P = # of production * Largest R.H.S of production.

$S \to AB$

$A \to aAB/a$ $\Rightarrow 3 \times 4 = 12$.

$B \to bb$

② $w \in L$, $|w| \geq m$.

③ $w = u.v.x.y.z$.

where $|vxy| \leq m$ & $|vy| \geq 1$.

④ $w = u.(y)^i.x.(y)^i z$.

$\forall i \geq 0 \Rightarrow w \in L$.

eg#02 $L = \{ww \mid w \in \{0,1\}^*\}$.

$w = 01 \Rightarrow ww = 0101$

$m = 5$

$w = 0^m 1^m 0^m 1^m$.

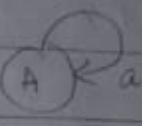
$w = \underbrace{0^5}_{u}\underbrace{1}_{v}\underbrace{1}_{x}\underbrace{1}_{y}\underbrace{1}_{v}\underbrace{0^5}_{z}\underbrace{1^5}_{}$

eg# 3 $L = \{a^i b^j c^i d^j\}$.

Turing Machines

* Type - 3 Grammar. → Regular Languages
  $\alpha \rightarrow \beta$
  where $\alpha \in$ single Non-terminal.
  $\beta \in |NT + T| \leq 2$ (FA's)

eg $A \rightarrow Aa \mid aA \mid a$



* Type - 2 Grammar. → CFL
  $\alpha \rightarrow \beta$.
  where $\alpha \in$ single Non-terminal.
  $\beta \in (NT + T)^*$ (PDA's)

eg $A \rightarrow aaA$
$A \rightarrow AB \mid \Lambda$
$B \rightarrow bbb$

* Type - 1 Grammar. → Context Sensitive Language.
  $\alpha \rightarrow \beta$
  where $\alpha \in (NT + T)^+$
  $\beta \in (NT + T)^*$

eg $aAb \rightarrow axb$.

Turing Machine + Finite tape
(Linear bounded Automata)

* Type - 0 Grammar → Recursive & Recursive Enumerable Language.

$\alpha \rightarrow \beta$

Where $\alpha \rightarrow (NT + T)^+$

$\beta \rightarrow (NT + T)^*$

eg $aAb \rightarrow axb$.

Turing Machine with infinite tape.

A Turing Machine :-

→ Consist of
- Tape (File)
- Control Unit (Finite Automaton).

→ Operations on - tape
- * Reads a symbol
- * Write a symbol
- * Moves left or right.

→ States & - transitions.

* Deterministic machine.
* Can have multiple transitions.
* No outgoing transition from final state.
* Halting: Machine halts if there is no possible transition.

* Acceptance → Machine halts in a final state.

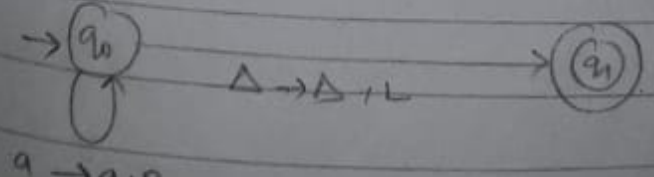
* Rejection → Machine halts in a non-final state.
(OR) Infinite loop.

Type 0

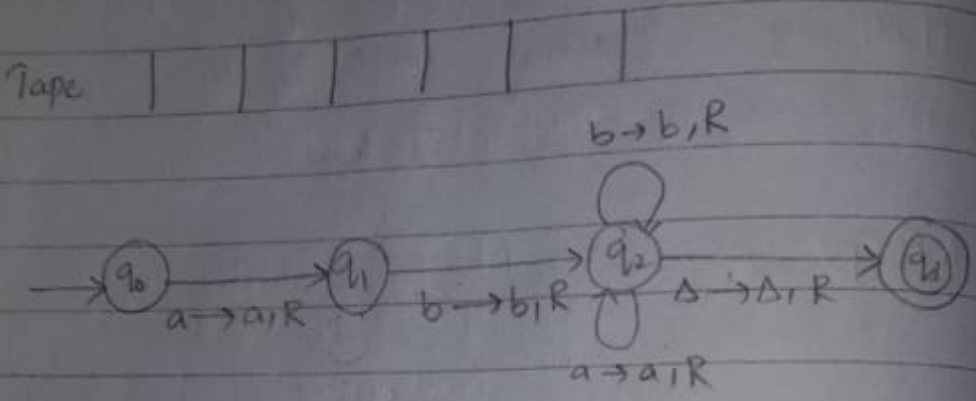
Recursive	Recursable Enumerable
Accept + halt	Accept + halt.
Reject + halt	Reject + infinite loop

eg Turing Machine that accepts a*.

Tape | Δ | Δ | a | a | Δ | Δ | .....

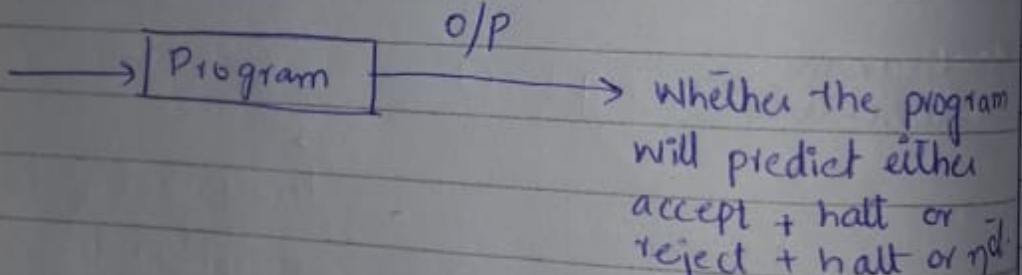


eg TM for L = {all strings in {a, b}* with prefix ab}.



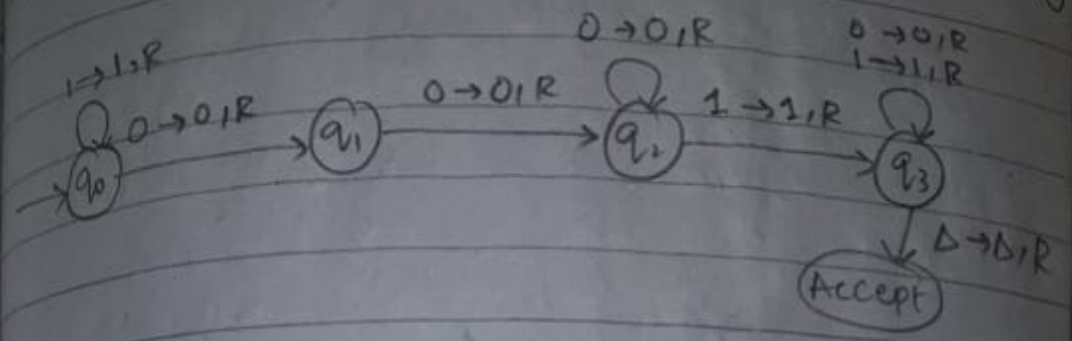
25 | Apr | 19

→ Non-recursive enumerable :

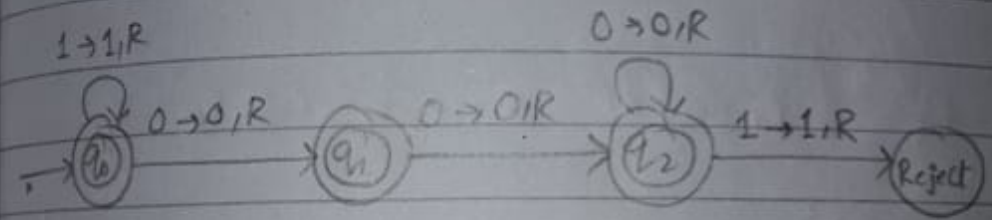


→ Turing machine expressed all the computable programs.
If program is not expressed by Turing machine then it cannot be a computable program.

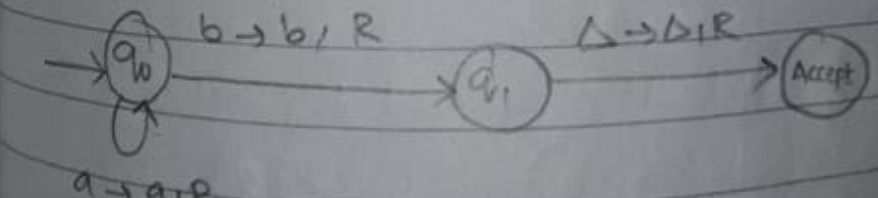
eg: All binary strings containing substring 001.



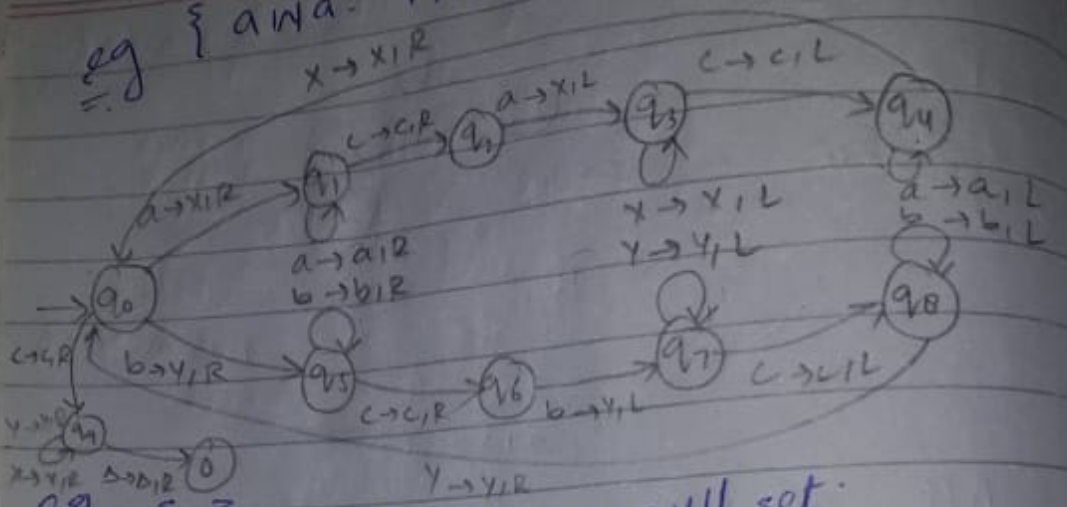
eg: All binary strings without substring 001.



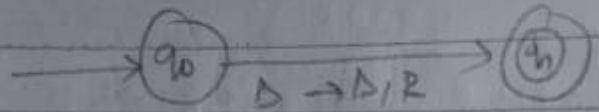
eg: $\{a^n b : n \geq 0\}$



eg $\{awa : w \in \{a,b\}^*\}$.

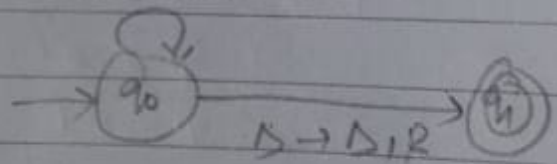


eg $\{\}$ => Accepting a null set.

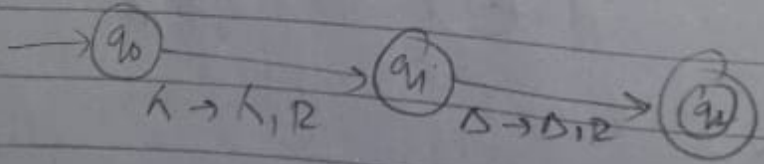


eg $\{a,b\}^*$

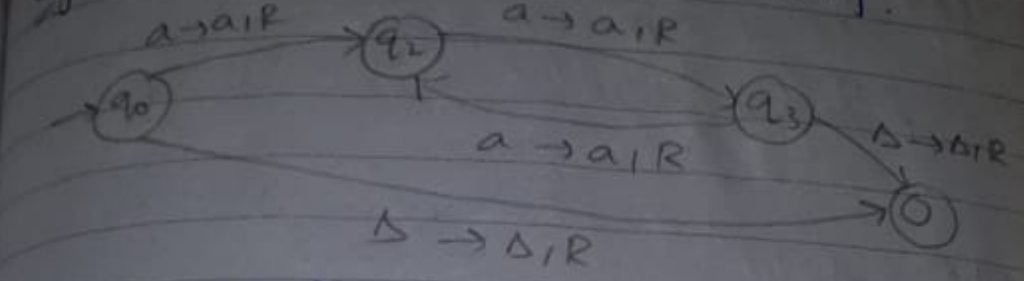
b → b,R
a → a,R



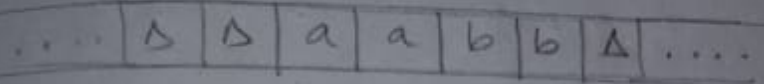
eg $\{\lambda\}$.



eg $\{x : x \in \{a\}^* \text{ & } |x| \text{ is even}\}$.



eg $\{a^n b^n\}$



a a a b b b

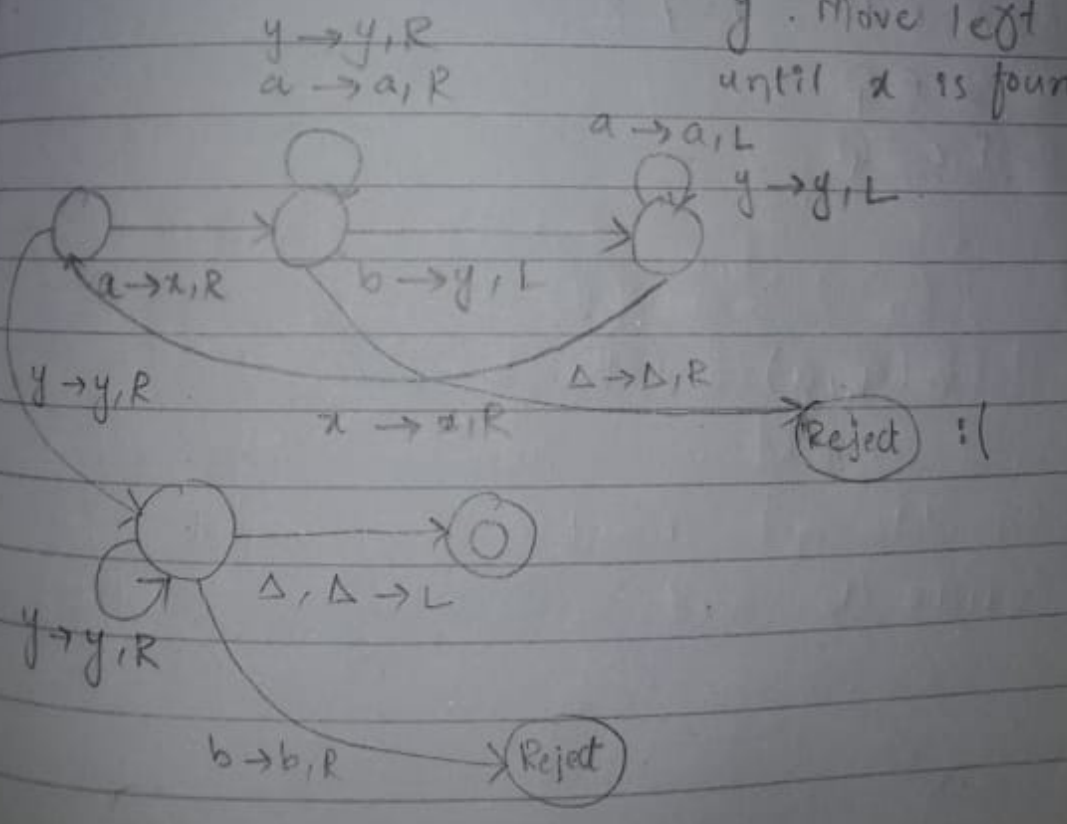
x → y

x ← y

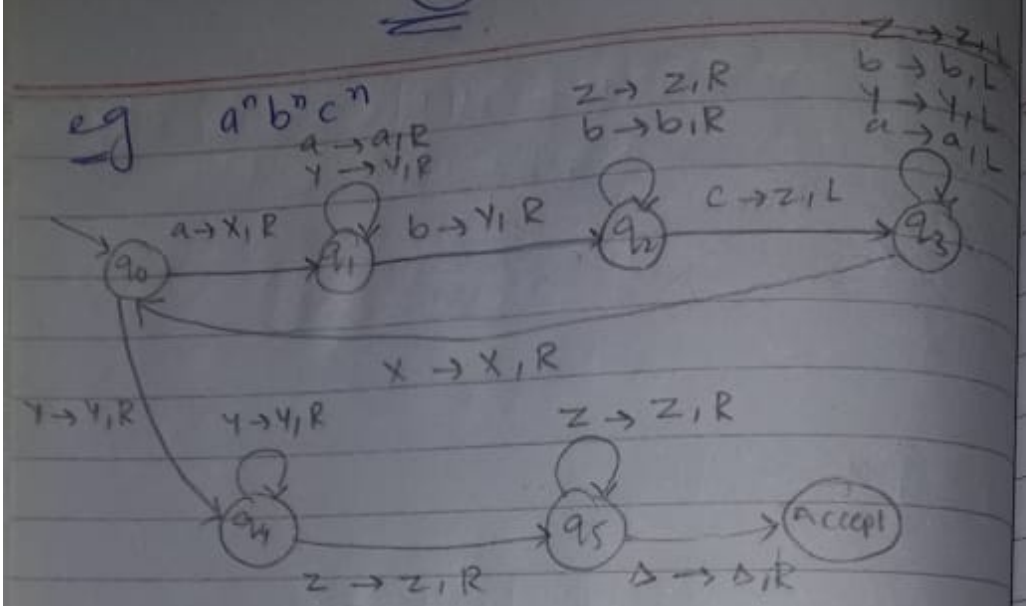
→ y

Find a, write x, move Right until b is found.

Find b, then write y. Move left until x is found



eg $a^n b^n c^n$



30/Apr/19

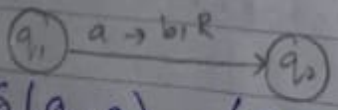
$M = (Q, \Sigma, T, \delta, q_0, \Delta, F)$.

$\Sigma \subseteq T$

$\Delta \in T$, $\Delta \notin \Sigma$

$q_0 \in Q$

$F \subset Q$.



$\delta(q_1, a) = (q_2, b, R)$

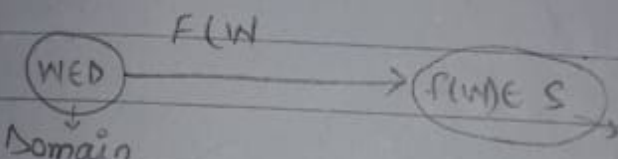
read, write, move

on state q1 reading an input 'a', write 'b' and move right at state q2

→ Standard Turing Machine
- Deterministic
- Infinite tape

Computing Functions With Turing Machines :

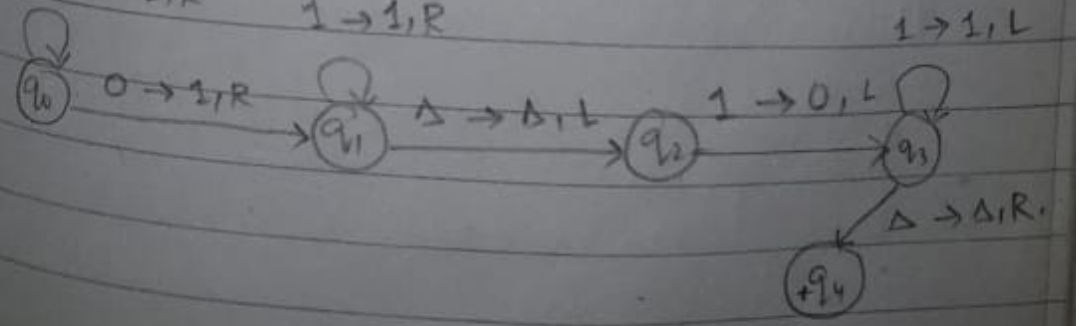
eg: square, factorial, addition etc.



A function f is computable if there exist a turing Machine M.

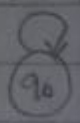
eg #1 $f(x, y) = x + y$ (unary operation)

Input : x O y → seperator
Output : xy O



→ Increment & decrement machine.

eg#2 Draw a Turing machine to add multiple unary nos.



eg#3 $f(x) = 2x$

Input string : x
Output string : xx

Algo:
① Replace every 1 with $.
② Repeat
→ Find right most $, replace it with 1.
→ Go to right end, insert 1.
Until no more $ remain.

eg #4 Combination of Turing machine

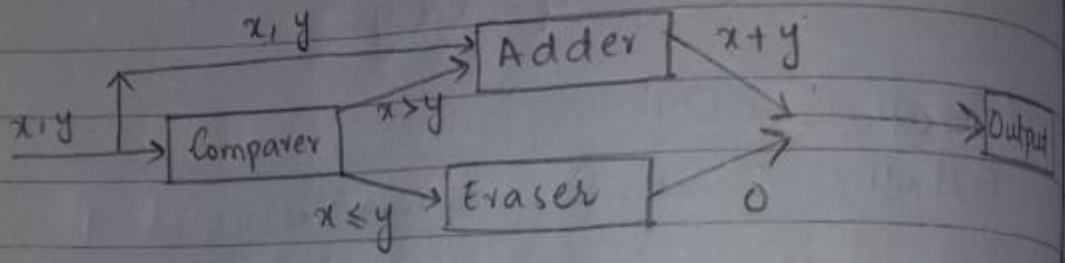
$$f(x,y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{if } x \leq y \end{cases}$$

Input : x 0 y
Output : 1 or 0.

- Repeat
  Match a 1 from x with a 1 from y
  Until all of x or y is matched.
- If a 1 from x is not matched
  erase tape, write 1 $(x > y)$
else
  erase tape, write 0 $(x \leq y)$.

eg HS

$$f(x,y) = \begin{cases} x+y & \text{if } x > y \\ 0 & \text{if } x \leq y \end{cases}$$



Imp

- Binary addition, subtraction, multiplication
- Primality Checking
- Factorial calculation.

Variations of the Standard Model.

→ Simulation:

① Stay-Option Machine:

- Read, write, stay.

stay = move right, then move left.

- Have same power as standard Turing machine.

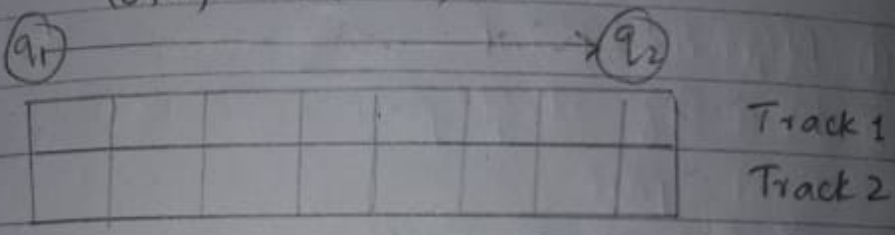
- Less time complexity.

2nd/May/19

② Multiple -track -tape :

- One tape with multiple track.
- Reading multiple symbols & writing multiple symbols.
- One read/write header.

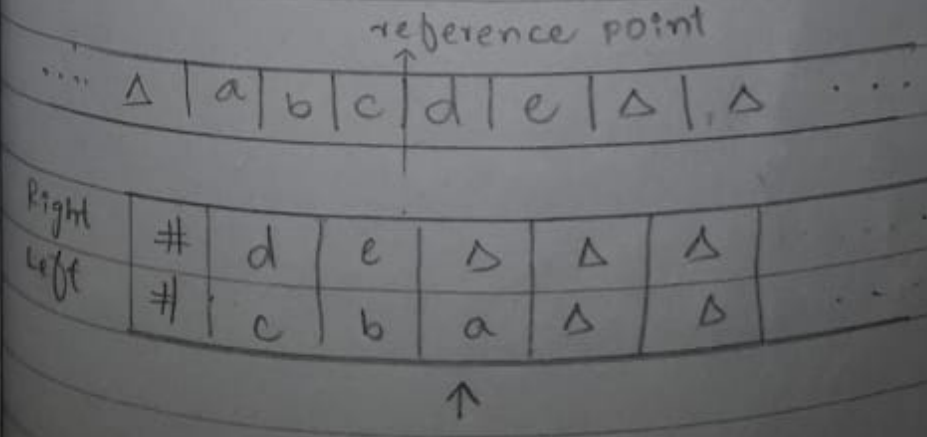
$(b, a) \longrightarrow (c, d), L$

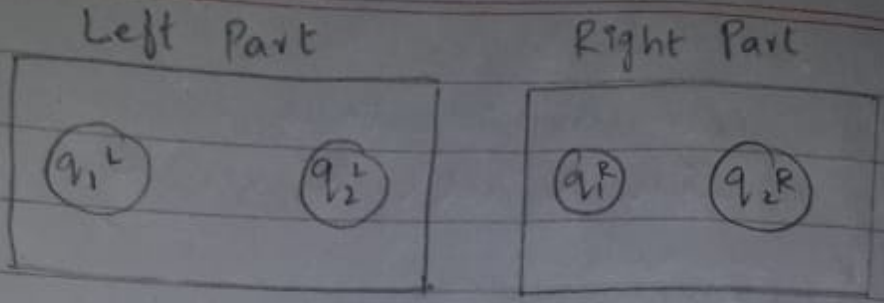


③ Semi- Infinite tape :

- Have same power with standard model
- Insert a special symbol # at left of input string.
- Add self-loop to every state (except states with no outgoing transitions)

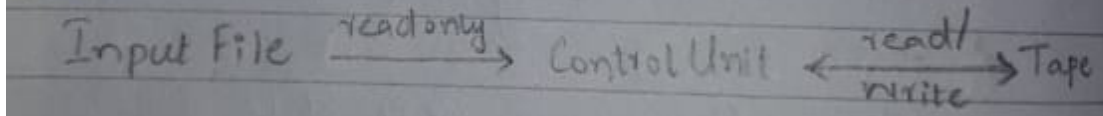
Standard Machine to SemiInfinite With 2 -tracks :





① Reference point.
② FSM of Left part ≠ FSM of right part
③ Connect both FSM's

④ The Off-Line Machine:



1. Copy input file to tape.
2. Do computations as Turing machine.

#	a	b	c	d
#	0	0	1	0
	e	f	g	Δ
	0	1	0	0

- Return to reference point
- Find current input file symbol
- Find current tape symbol.
- Make transitions.

(5) Multitape Turing Machines.

- Multiple read write heads
- Can move in multiple direction at a time.

$(b, f) \rightarrow (g, d), L, R$

$q_{11} \rightarrow q_{12}$

eg $a^n b^n$

Tape 1 → all a's
Tape 2 → all b's
} only 1 transition required

much faster.

Multiple tape in standard model.

Tape 1

Δ	a	b	c	Δ

Tape 2

Δ	e	f	g	h	Δ

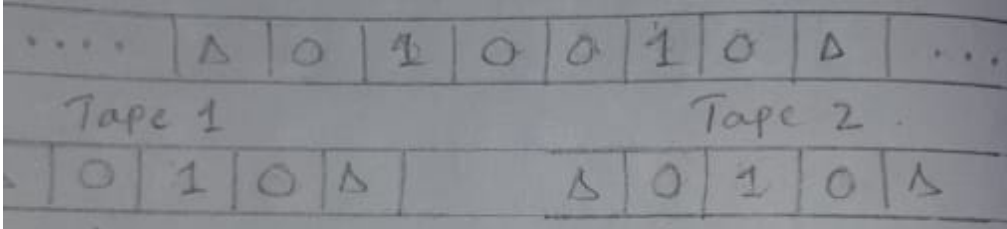
#	a	b	c
#	0	1	0
#	e	f	g	h
#	0	0	1	0

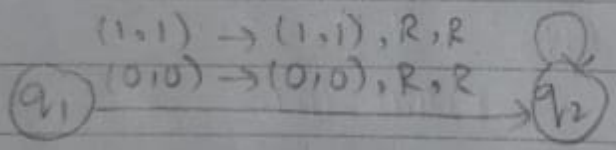
Standard Model.

- Return to reference point
- Find current symbol on Tape 1
- Find current symbol on Tape 2
- Make transition.

eg Draw two tape turing machine for $W^R = W$ define over strings 0,1.

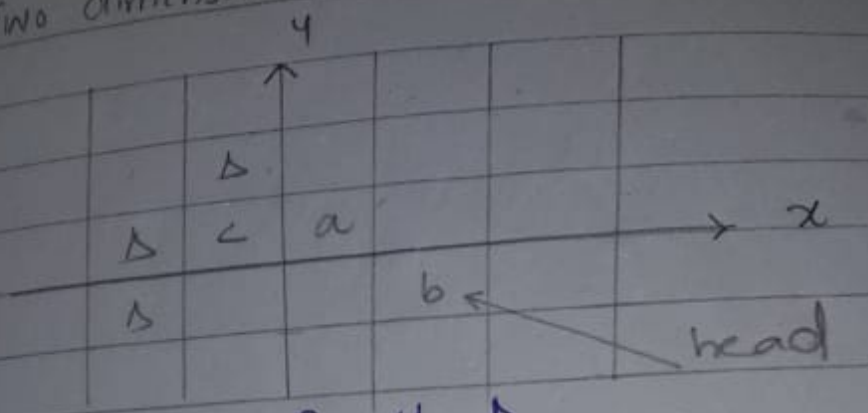
→ Assume palindrome is of even length





⑥ Multi Dimensional Turing Machine:

Two dimensional tape



Moves : L, R, U, D.

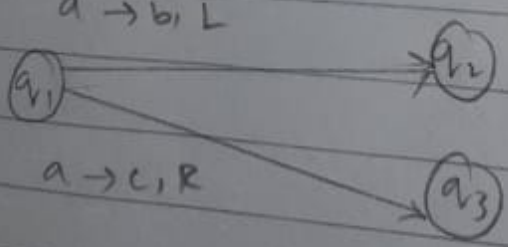
Standard Model

a				b				c			symbols
1	#	1	#	2	#	-1	#	-1	#	1	coordinate

$a \rightarrow (1,1)$

$(a(1,1)) \longrightarrow (b(2,1), R)$

⑦ Non-Deterministic Turing Machine:



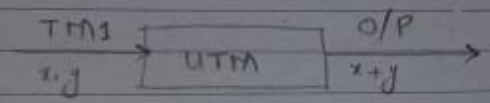
Two tape model.

6/May/19

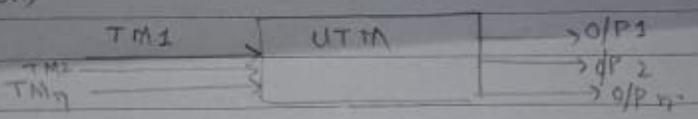
Universal Turing Machine

→ Attributes

* Reprogrammable Machine.
* Simulates any other turing machine.



(OR)



Input : Description of Machine (All 7 tuples)
Initial Tape.

We encode M as a string of symbols.
e(T) e(z)

Alphabet Encoding

Symbols: a, b, c, d
Encoding: 1, 11, 111, 1111

Head Move Encoding } Transitions. $\delta(q_1, a) = (q_2, b, L)$
Encoding 10110 1101
↑
seperator

Move L R S
Encoding 1 11 111.