GRAMMAR

* Grammer consist of set of rule to define a language.
* It can be define through 4 Tuples.

  $\langle V, T, P, S \rangle$

  (*) V stands for set of non terminal values or symbols. It can produce a new item or new non terminal value.

  (*) T stands for set of terminal symbols or input alphabet. It can generally denoted through all the symbol and character except capital letter.

  (*) P stands for production rules.

  (*) S is the starting symbol.

• TYPES OF GRAMMER

⇒ Based on string, the Grammer may be recursive or non-recursive.

⇒ Based on derivation tree, the Grammer may be ambigious and unambigious.

⇒ Based on chomsky classification, chomsky classify the Grammer in 4 different types.

(i) Type 0
(ii) Type 1
(iii) Type 2
(iv) Type 3

Gramer Type	Grammer Accepted	Language Accepted	Automation
Type-0	unrestricted	Recurssivly Enumerable Language	Turing Machine (TM)
Type-1	Context-Sensitive	Context-Sensitive	Linear Bounded Automation (LBA)
Type-2	Context free	Context free	Push down Automation (PDA)
TYPE-3	Regular	Regular	finite Automata or finite state Machine

- **TYPE-3 GRAMMAR :-**

(*) Type 3 is a regular grammer and it supports regular language.

(*) Here the grammer must have a single non-terminal in the left side and right hand side consist a single terminal or single terminal followed single non terminal symbol in the production.

(*) The General form of production will be :

$A \to a$
$A \to aB$

$A, B \in V$
$a \in T$

(*) $S \to \epsilon$ is allowed iff S is not produce in right hand side of any production rule.

Example-

$X \to \epsilon$
$X \to aY \mid b$
$Y \to b$

} Type 3

- **TYPE-2 GRAMMAR :-**

(*) Type-2 Grammer generates context-free Language.

(*) The production rule must be in the form of :

$A \to \gamma$ where $A \in V$
$\gamma \in (V \cup T)^*$

Example -

$X \to \epsilon$
$X \to aY \mid b$
$Y \to Ba$

• TYPE-1 GRAMMER :-

* Type 1 is called context-sensitive Grammer and the language support called context-sensitive language.
* It is recongnized by linear bound automata.
* The production rule must be in the form of

$\alpha A B \longrightarrow \alpha \gamma B$,

where $\alpha, B, \gamma \in (V \cup T)^*$

γ should not be empty.

$A \in V$

(1) $A \rightarrow \epsilon$ is allowed if A does not appear right sand of any production rule.

$ABa \rightarrow aBCAaba$

• TYPE-0 GRAMMER :-

* Type-0 is a most restricted type of grammer
* production rule of Type-0

$\alpha \rightarrow \beta$

where $\alpha, \beta, \in (V \cup T)^*$

$\alpha \neq \epsilon$

• REGULAR GRAMMER :-

⇒ The regular Gremer is type-3 Grammer agin classifier into two categories.

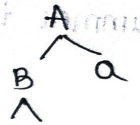
(1) Left linear grammer.
(2) Right linear grammer

(1) Left Linear Grammer -

If the right hand side of any production contains a non-terminal in left most position then the grammer is said to be left linear Grammer

* Here the derivation tree will expand towards left direction.

$A \longrightarrow Ba$

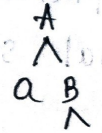


(2) Right Linear Grammer -

If the right hand side of any production contains a non-terminal in right most position then the grammer is said to be right Linear Grammer.

* Here the derivation tree will expand towards right direction.

$A \longrightarrow aB$



Conversion from Grammer to FA :-

RULES

The tuple of the Grammer are $\langle V\ T\ P\ S \rangle$ Where as the tuple of finite Automata are $\langle Q, \Sigma, \delta, q_0, f \rangle$ by considering the tuples, we can find Similarity between grammer and finite Automata.

$$\langle V\ T\ P\ S \rangle$$
$$\langle Q\ \Sigma\ \delta\ q_0\ F \rangle$$

To convert Grammer to finite Automata followings are the steps :-

Step1 :-
S which is the starting symbol in the grammer, will be considered as starting state in to finite automata.

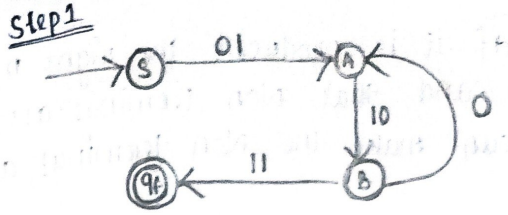
Step2 :-
for each production in the form of $q_i \rightarrow w\,q_j$, will be converted to transition of $\delta(q_i, w) \rightarrow q_j$.
for the production rule in the form of $q_i \rightarrow w$, that can be represented as $\delta(q_i, w) \rightarrow q_f$, where q_f is the final state.

(*) Convert the following Grammer to finite Automata

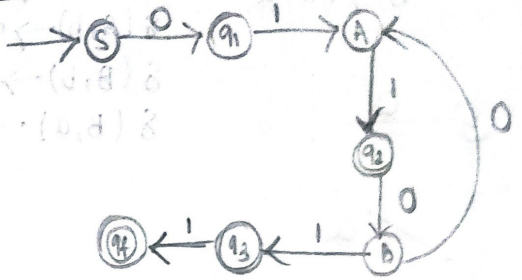
$S \rightarrow 01A$, $A \rightarrow 10B$, $B \rightarrow 0A \mid 11$

$\delta(S, 01) \rightarrow A$
$\delta(A, 10) \rightarrow B$
$\delta(B, 0) \rightarrow A$
$\delta(B, 11) \rightarrow q_f$

Step1



Step2

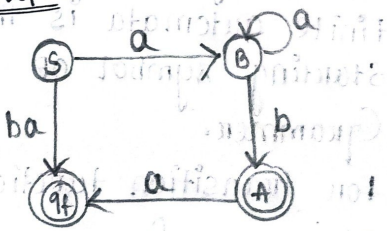


$S \rightarrow 01A$
$\rightarrow 0110B$
$\rightarrow 01100A$
$\rightarrow 0110010B$
$\rightarrow 011001011$
(left most derivation)

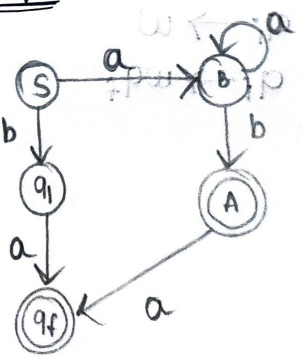
∴) This string also accepted by finite Automata.

(*) $S \rightarrow aBc \mid ba$
$B \rightarrow BA \mid aB$
$A \rightarrow a \mid \epsilon$

Step1



Step2



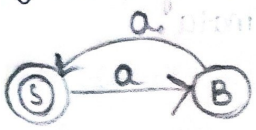
$\delta(S, a) \rightarrow B$
$\delta(S, ba) \rightarrow q_f$
$\delta(B, b) \rightarrow A$
$\delta(B, a) \rightarrow B$
$\delta(A, a) \rightarrow q_f$
$\delta(A, \epsilon) \rightarrow A$ → final State

NOTE :-

- for any Non terminal if it is produced the right most osition of any production - and that Non terminal also producing ϵ then we can make the Non terminal as Final state.

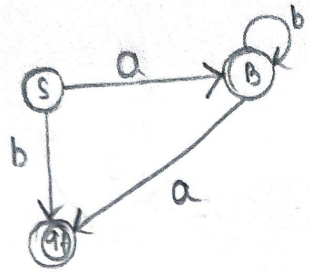
:) $S \rightarrow aB \mid \epsilon$
$B \rightarrow aS$

$\delta(S,a) \rightarrow B$
$\delta(B,a) \rightarrow S$
$\delta(S,\epsilon) \rightarrow q_f$



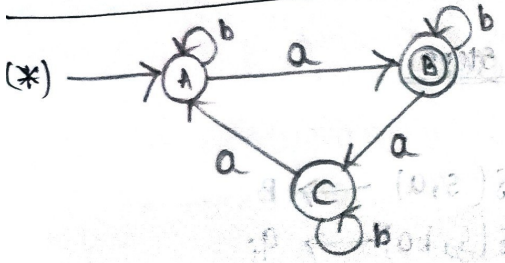
(*) $S \rightarrow aB \mid b$
$B \rightarrow bB \mid a$

$\delta(S,a) \rightarrow B$
$\delta(S,b) \rightarrow q_f$
$\delta(B,b) \rightarrow B$
$\delta(B,a) \rightarrow q_f$



FINITE AUTOMATA TO GRAMMER :-

(*)



$\delta(A,b) \rightarrow A$
$\delta(A,a) \rightarrow B$
$\delta(B,b) \rightarrow B$
$\delta(B,a) \rightarrow C$

$\delta(C,a) \rightarrow A$
$\delta(C,b) \rightarrow C$

$A \rightarrow BB \mid aB \mid a$
$B \rightarrow bB \mid ac \mid \epsilon$
$C \rightarrow aA \mid bc$

(*) Starting state of finite automata is the starting symbol of Grammer.

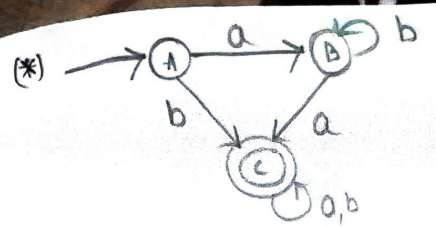
for transition function

$\delta(q_i, w)$ = final state

$q_i \rightarrow w$

$q_i \rightarrow wq_f$

(*)



$A \rightarrow aB \mid bC \mid b$
$B \rightarrow bB \mid ac \mid a$
$C \rightarrow ac \mid bc \mid \epsilon$

or

$ac \mid bC \mid a \mid b$

UNIT PRODUCTION :-

(*) A production is said to be unit production IF right hand side (RHS) of Rule contain only one non terminal.

$A \to B$, where both A & B are non terminal.

$A \to B$
$B \to a \Rightarrow A \to a$

(Q) $S \to AB \mid d \mid A \mid B \mid \epsilon$
$A \to a$
$B \to bA \mid d \mid b$

Remove a :-
$S \to AB \mid d \mid B \mid \epsilon \mid a$
$A \to a$
$B \to bA \mid d \mid b$

Remove b :-
$S \to AB \mid d \mid A \mid bA \mid d \mid b$
$A \to a$
$B \to bA \mid d \mid b$

Final production -
$S \to AB \mid \epsilon \mid a \mid bA \mid d \mid b$
$B \to bA \mid d \mid b$
$A \to a$

Chomsky Normal form.

A grammer is said to be CNF, if all the product^n are either of $A \to a$ or $A \to BC$ where $A, B, C \in V$ and $a \in$ terminal

* $S \to \epsilon$ is allowed if S does not appear right side of any production.

$A \rightarrow \underbrace{a}_{\beta_1} | \underbrace{A_2 A_3 A_4}_{\alpha} | \underbrace{A_4 A_4 A_4}_{\beta_2}$

$A_2 \rightarrow aZ \mid A_4 A_4 A_4 Z$
$Z \rightarrow \epsilon \mid A_3 A_4 Z$
$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$
$A_4 \rightarrow b$
$A_3 \rightarrow d$

$A_2 \rightarrow aZ \mid bA_4A_4Z$
$Z \rightarrow e \mid dA_4Z$
$A_1 \rightarrow aZA_3 \mid bA_4A_4ZA_3 \mid bA_4$
$A_4 \rightarrow b$
$A_3 \rightarrow d$

(Q) Remove left recursion from the grammer

$A \rightarrow aB \mid AbD \mid cdA$
$B \rightarrow b$
$D \rightarrow d$

$A \rightarrow \underbrace{aB}_{\beta_1} | \underbrace{AbD}_{\alpha} | \underbrace{cdA}_{\beta_2}$

$A \rightarrow aBZ \mid cdAZ$
$Z \rightarrow \epsilon \mid bDZ$
$B \rightarrow b$
$D \rightarrow d$

CYK ALGORITHM :- (COCKE YOUNGER KASAMI)

⇒ This algorithm is used to check a string to be accepted by grammer or not.

⇒ To check this thing the grammer must be in CNF.

(Q) $S \rightarrow AB$
$A \rightarrow BB \mid a$
$B \rightarrow AB \mid b$

Given string "abbb"

find out the number of alphabet of

Step 1

(1) find out the number of alphabet in a string.
n=4

(2) Design a table with 4 row and 4 column.

Step 1 may be skip as the grammer is already in CNF.

	4	3	2	1
1	{S,B}	{A}	{S,B}	{A}
2	{S,B}	{A}	{B}	
3	{A}	{B}		
4	{B}			

"a b b b" {start, end}
1 2 3 4

✓(1,1) → A (∵ A → a)
✓(2,2) → B (∵ B → b)
✓(3,3) → B (∵ B → b)
✓(4,4) → B (∵ B → b)
✓(1,2) → (1,1)(2,2) → AB = S, B
(∵ S → AB, B → AB)
✓(2,3) → (2,2)(3,3) → BB = A
(∵ AB → BB)
✓(3,4) → (3,3)(4,4) → BB = A
(∵ A → BB)
✓(1,3) → (1,1)(2,3) ⇒ AA = Φ
or, (1,2)(3,3) ⇒ {S,B}{B} = SB, BB = Φ, A
✓(2,4) → (2,2)(3,4) ⇒ {B}{A} = BA
or, (2,3)(4,4) ⇒ {A}{B}
= AB = S, B
(∵ S → AB, B → AB)
✓(1,4) → (1,1)(2,4) → {A}{S,B} = AS, AB = Φ, B, S
or, (1,2)(3,4) → {S,B}{A} = Φ
or, (1,3)(4,4) → {A}{B} = AB = S, B

$S \to A_1$ | $A_1 \to A_2A_3 \mid A_4A_4$
$A \to A_2$ | $A_2 \to a \mid A_1A_4$
$C \to A_3$ | $A_4 \to b$
$B \to A_4$ | $A_3 \to d$

⇒ Consider, for the production A_i to A_j, i should always less than j.

(A_j always considered as first non-terminal in RHS)

$A_1 \to A_2A_3 \mid A_4A_4$
$A_2 \to a \mid A_2A_3A_4 \mid A_4A_4A_4$
$A_4 \to b$
$A_3 \to d$

Now,
$A_2 \to A_2A_3A_4$ is a case of left recursion, so this has to removed from grammer.

(*) The recursion is a process where a non terminal produce it self in right hand side of production.

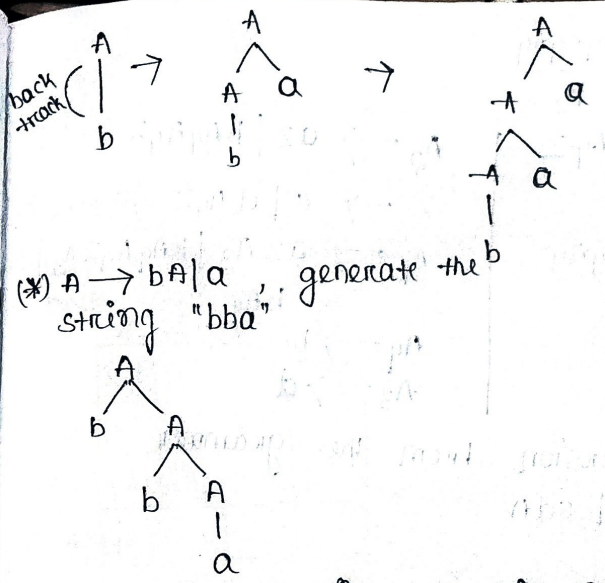
⇒ There are 3 case of Recursion.

(1) Normal : $A \to \alpha A\beta$
(2) left recursion : $A \to A\alpha$
(3) Right recursion : $A \to \alpha A$

⇒ All the three case, difficulty level case is left recursion.

WHY?

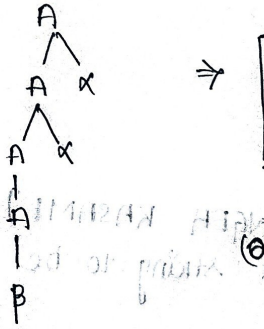
Suppose the production rule is $A \to Aa \mid b$ and generate the string "baa".



(*) $A \to bA \mid a$, generate the string "bba"

But in right recursion, we will not face such kind of problem as we can visualize the first symbol and as per requirement we can choose the production.

(a) $A \to A\alpha \mid B$



⇒ $A \to B\alpha^*$, $\alpha^* \to \epsilon \mid \alpha\alpha^*$ | $A \to BZ$, $Z \to \epsilon \mid \alpha Z$

(a) $A \to A\alpha \mid B_1 \mid B_2$

$A \to B_1Z \mid B_2Z$
$Z \to \epsilon \mid \alpha Z$

(a) $A \to A\alpha \mid A\beta \mid \alpha$

$A \to \gamma\alpha^* \mid \gamma B^*$
$\alpha^* \to \epsilon \mid \alpha\alpha^*$
$B^* \to \epsilon \mid B\beta^*$

$A \to \gamma\gamma \mid \gamma Y$
$X \to \epsilon \mid \alpha X$
$Y \to \epsilon \mid BY$

} two left Recursion

Conversion of CFG to CNF

(1) Remove useless production and unreachable code from the Grammer.
(2) Remove Null production.
(3) Remove unit production
(4) Convert all the production into CNF format.

(Q) $S \rightarrow aAD$
$A \rightarrow aB \mid bAB$
$B \rightarrow b$
$D \rightarrow d$

(1) There is no unit production and null production in grammer. Hence this step may be skip.

(2) After checking the format of CNF, convert all the terminal into non terminal from rest of production.

$S \rightarrow XAD$
$A \rightarrow XB \mid YAB$
$B \rightarrow b$
$D \rightarrow d$
$X \rightarrow a$
$Y \rightarrow b$

$S \rightarrow PD$
$P \rightarrow XA$
$A \rightarrow XB \mid QB$
$Q \rightarrow YA$
$B \rightarrow b$
$D \rightarrow d$
$X \rightarrow a$
$Y \rightarrow b$

(Q) $S \rightarrow \sim S \mid [S \supset S] \mid P \mid d$

$S \rightarrow P$
$S \rightarrow d$
$S \rightarrow XS$
$X \rightarrow \sim$
$P \rightarrow [$
$Q \rightarrow C$
$R \rightarrow \supset$
$T \rightarrow]$

$S \rightarrow \overline{PQ}\,\overline{SR}\,\overline{ST}$

$S \rightarrow UVW$
$U \rightarrow PQ$
$V \rightarrow SR$
$W \rightarrow ST$
$Z \rightarrow UV$

$S \rightarrow ZW$

Greibach Normal form

(*) To convert a grammer to GNF, the grammer must be in CNF.
(*) All the production must be in the form of
$A \rightarrow a A_1 A_2 \ldots$
or
$A \rightarrow a$ means, Right side contain either one terminal symbol or a terminal followed by any number of non-terminal symbol.

(Q) convert the following grammer to GNF.
$S \rightarrow AC \mid BB$
$A \rightarrow a \mid SB$
$B \rightarrow b$
$C \rightarrow d$

⇒ The grammer is already in CNF, hence step 1 may be skip.

(1) Convert all nonterminal into ascending order of A_i, where i is a integer value.

Finally we can conclude that if the starting symbol is available on top most corner of table then, the string is accepted.

So here the string is accepted by Grammer.

Q) S → AB / BC
A → BA / a
B → CC / b
C → AB / a

Given string is "ababa"

(*) Here number of alphabet is 5.
So we should draw a matrix of 5 row & 5 column

	5	4	3	2	1
1	{S,A}	{B}	{B}	{S,C}	{A}
2	{B}	{S,C}	{A,S}	{B}	
3	{B}	{S,C}	{A}		
4	{S,A}	{B}			
5	{A}				

" a b a b a "
1 2 3 4 5

one combination

✓(1,1) = a = A, C (∵ A → a)
✓(2,2) = b = B (∵ B → b)
✓(3,3) = a = A, C
✓(4,4) = b = B
✓(5,5) = a = A, C

two combination

✓(1,2) = (1,1)(2,2) = (A,C)(B) = AB, CB = {S,C}, φ
✓(2,3) = (2,2)(3,3) = {B}{A,C} = BA, BC = {S}, {A,S}
✓(3,4) = (3,3)(4,4) = {A,C}{B} = AB, CB = {S,C}, φ
✓(4,5) = (4,4)(5,5) = {B}{A,C} = BA, BC = A, S

three combination

✓(1,3) = (1,1)(2,3) = {A,C}{A,S}
= AA, AS, CA, CS
= φ, φ, φ, φ

or,
(1,2)(3,3) = {S,C}{S,C}
= SS, SC, CS, CC
= φ, φ, φ, B

✓(2,4) = (2,2)(3,4) = {B}{S,C} = BS, BC = φ, S
or, (2,3)(4,4) = {A,S}{B} = AB, SB = {S,C}, φ

✓(3,5) = (3,3)(4,5) = {A,C}{S,A} = AS, AA, CS, CA
= φ
or, (3,4)(5,5) = {S,C}{A,C} = SA, CA, SA, CC
= B

four combination

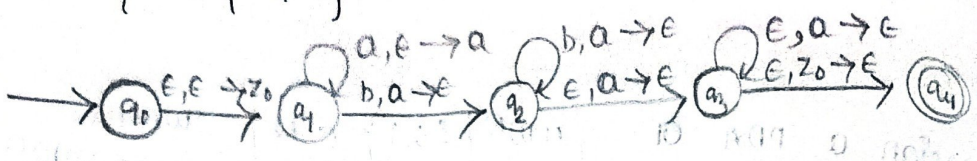
✓(1,4) = (1,1)(2,4) = {A,C}{S,C} = φ
or, (1,2)(3,4) = {S,C}{S,C} = SS, SC, CS, CC = B
or, (1,3)(4,4) = {B}{B} = φ

✓(2,5) = (2,2)(3,5) = {B}{B} = BB = φ
or, (2,3)(4,5) = {A,S}{S,A} = AS, AA, SS, SA = φ
or, (2,4)(5,5) = {S,C}{A,C} = SA, SC, CA, CC = BB

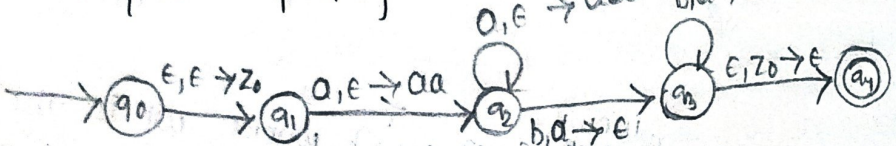
five combination

✓(1,5) = (1,1)(2,5) = (A,C)(B) = AB, CB = (S,C), φ
or, (1,2)(3,5) = (S,C)(B) = SB, CB = φ
or, (1,3)(4,5) = {B}{S,A} = BS, BA = φ, A
or, (1,4)(5,5) = {B}{A,C} = BA, BC = {S,A}

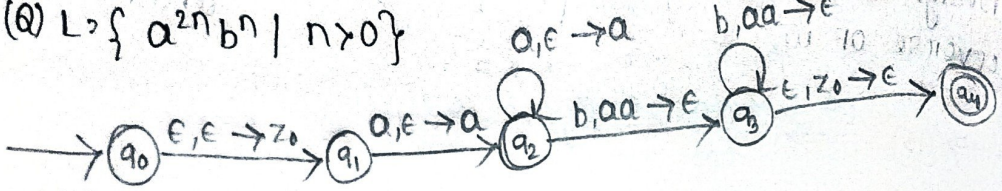
(Q) $L = \{a^m b^n \mid m > n\}$



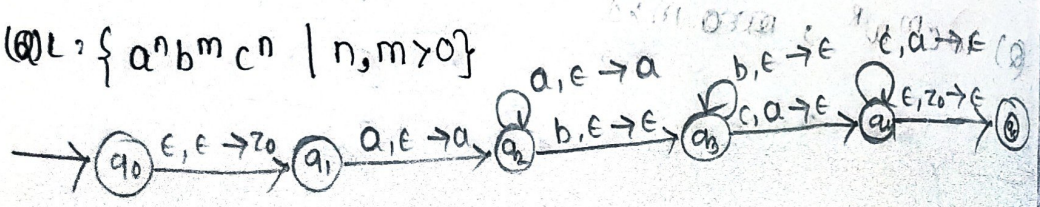
(Q) $L = \{a^n b^{2n} \mid n > 0\}$



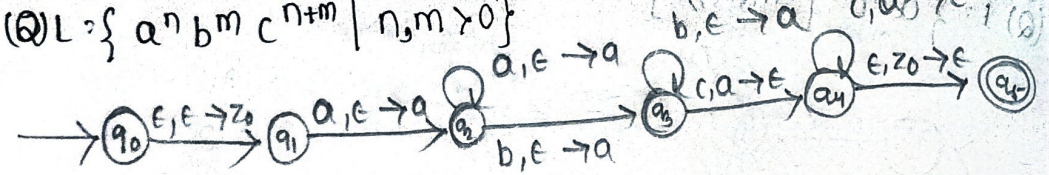
(Q) $L = \{a^{2n} b^n \mid n > 0\}$



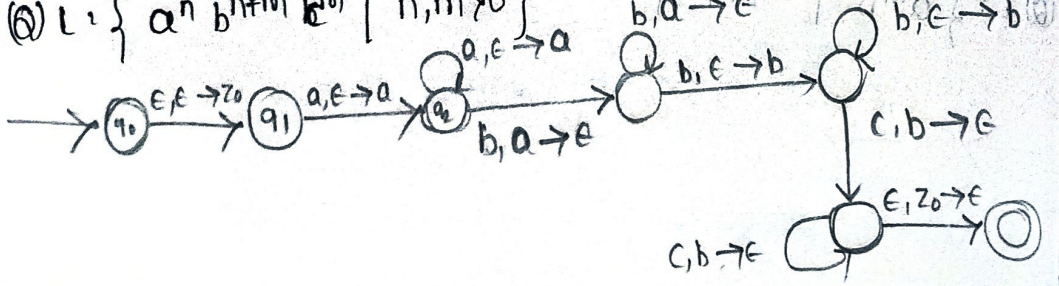
(Q) $L = \{a^n b^m c^n \mid n,m > 0\}$



(Q) $L = \{a^n b^m c^{n+m} \mid n,m > 0\}$



(Q) $L = \{a^n b^{n+m} c^m \mid n,m > 0\}$



Instantaneous Description ID -

* Instantaneous Description, ID describe the current instance of a PDA.
* It can be describe through three toruple.

(q, w, s)

$q \rightarrow$ current state, $w \rightarrow$ Remaining string input

$s \rightarrow$ stack symbol

(*) TYPE OF PDA :-

PDA can be broadly categorized into D-PDA and N-PDA.

(a) NPDA :-

Non Determinastic Push Down Automata

* In this PDA, there is no unique transition for single input symbol.
* Generally in designing of PDA, NPDA is used.

(b) DPDA

Determinastic Push Down Automata

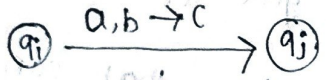
* Similar to DFA, DPDA has also unique tranisition for every input symbol.
* $\delta(q, a, x)$ is defined as unique where as $a \in \Sigma$, , $x \in \Gamma$
* If $\delta(q, a, x) \neq \phi$ then $\delta(q, \overset{\epsilon}{a}, x) = \phi$

  $\delta(q, a, x) = \phi$ then $\delta(q, \epsilon, x) \neq \phi$

How to Represent Transition function :-

Transition function for push down automata is represent through following notation i.e instantaneous description

* Here a triplet format is used i.e (Q W Z) where Q is the state, W is the unutilized input, Z is the stack contains.



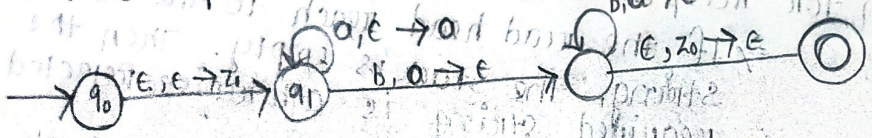
here, a = input symbol
b = poped value from stack
c = pushed symbol into top of stack.

(Q) Design a PDA, that accept the language $L = \{a^n b^n \mid n > 0\}$

NOTE :-

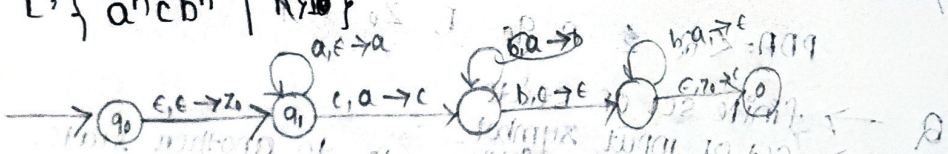
Pushing an element into the stack can be represented as $\alpha, \epsilon \rightarrow \alpha$

Deleting an element from top of the stack can be represented as $\alpha, \alpha \rightarrow \epsilon$

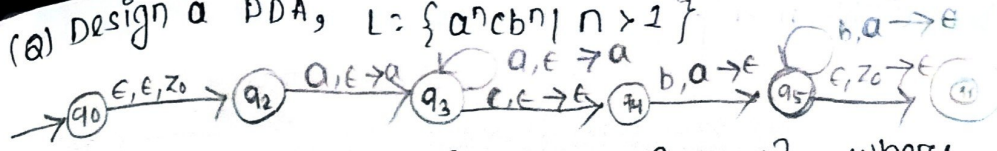


(Q) Design a PDA

$L = \{a^n c b^n \mid n \geq 0\}$

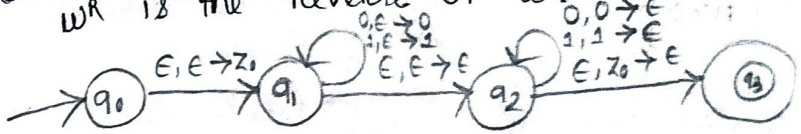


(Q) Design a PDA, $L = \{a^n c b^n \mid n > 1\}$



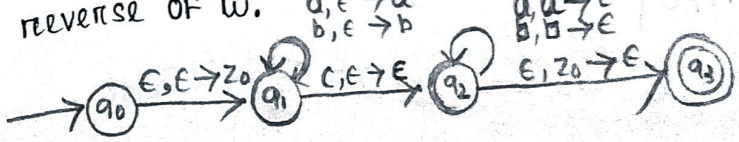
(Q) Design a PDA of $\{ww^R \mid \{0,1\} \in w\}$ where w^R is the reverse of w.

0 1 0 ∈ 0 1 0 (✓)
0 1 ∈ 0 1 (X)

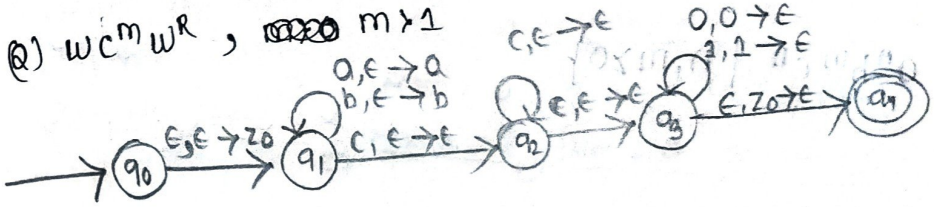


(Q) Design a PDA of $\{wcw^R \mid \{a,b\} \in w\}$ where w^R is the reverse of w.

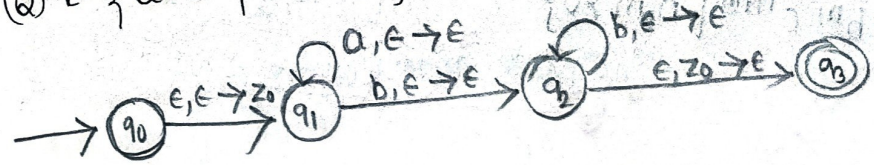
a b a c a b a



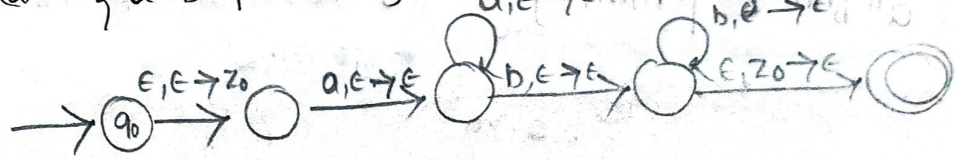
(Q) $wc^m w^R$, m > 1



(Q) $L = \{a^m b^n \mid m, n \geq 0\}$



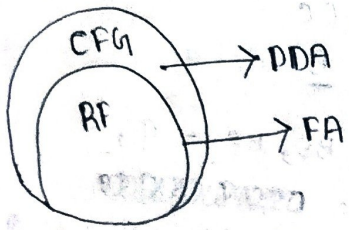
(Q) $L = \{a^m b^n \mid m, n > 0\}$



UNIT-03

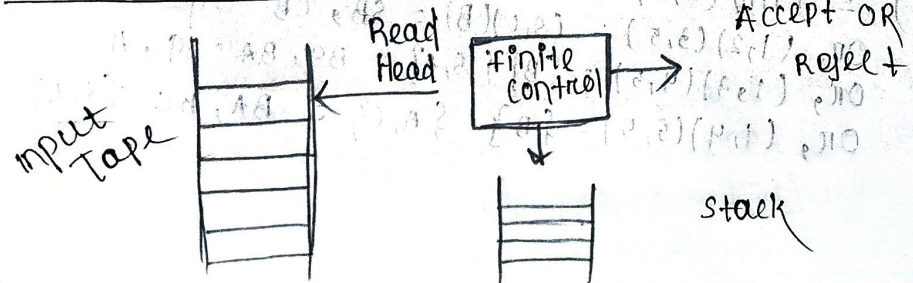
PUSHDOWN AUTOMATA

PDA



* Push down automata is way of implementing CFG in the same way we design finite automata for Regular Grammar.
* finite automata does not contain or contains very limited memory space whereas push down automata contain an infinite stack to store the value.
* The PDA is more powerful than finite Automata
* The CFG can not recognize by finite automata but can recognize by PDA.
* The PDA is also broadly categorize into
  (1) Deterministic PDA (DPDA)
  (2) Non Deterministic PDA (NPDA)

Components of PDA :-



1) Input tape -
* It is a infinite tape contain the input string.
* Every field of the input tape, can contain atmost one symbol.

2) Read Head -
* It read the symbol of input tape.

3) Finite Control -
It read the content from input tape. Then it may push or pop to stack and give it final decision.

4) Stack -
Stack is a infinite data structure that can store the ~~information~~ stack input. Here two operation can be perform.
① PUSH - finite control store an element on the top of stack.
② POP - finite control remove an element from top of stack.

Condition for Acceptance :-
⇒ If the read head reach to the end of string, the stack is empty. Then the required string is otherwise. rejected.

⇒ Push Down Automata defined through 7 tuple.

$$PDA = < Q \ \Sigma \ \delta \ q_0 \ F \ Z_0 \ \Gamma >$$

* Q → finite set of state
  Σ → set of input symbol
  δ → It is used to map from one state to another state,
  q_0 → initial state
  F → set of final state
  Z_0 → starting symbol of stack or end marker
  Γ → input stack symbol

$\delta \rightarrow \delta(q_i, a) \rightarrow (q_j, \chi, D)$

$q_0 \rightarrow$ Initial state, $\in Q$

$f \rightarrow$ Set of final states, $F \subseteq Q$

$B \rightarrow$ Special kind of Symbol is used to represent the blank cell on the input tape.

$\Gamma \rightarrow$ Set of tape symbol.

$$\delta(q_i, a) \rightarrow (q_j, \chi, D)$$

where $q_i q_j \in Q$

$a \in \Sigma$, $\chi \in \Gamma$

$D = \{R, L\}$ // D symbolizes movement of Read, write head

(Q) Design Turing machine that can accept the string '011'.

#	0	1	1	#

$\delta(q_0, 0) \rightarrow (q_1, X, R)$

$\delta(q_1, 1) \rightarrow (q_2, Y, R)$

$\delta(q_2, 1) \rightarrow (q_3, Y, R)$

$\delta(q_3, \#) \rightarrow$ Accept

q_0	(q_1, X, R)	—	—
q_1	—	(q_2, Y, R)	—
q_2	—	(q_3, Y, R)	—
q_3	—	—	Accept

(Q) Design a turing machine that accept the string over 0 and 1 and end with 1.

(Q) S → aAB | bA | aB

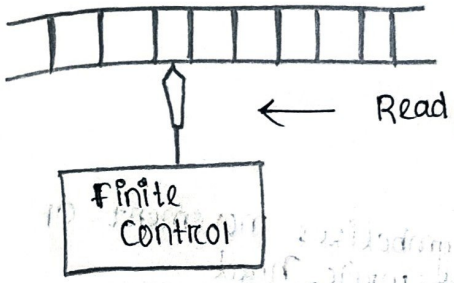
A → aA | b

B → bB | a

String "aaabbba"

TURING MACHINE

* Turing machine can recognize type-0 Grammer.
* @ Alen Turing in year 1936 has invented Turing Machine. He was a english mathemacian.
* Turing machine is a generalized form of PDA.



(*) Here input tape is a infinite tape, that can contain or accomdate infinite input symbol or output symbol. It is containing one special symbol for denoting blank space. (#) ($) --
so it combine the features of input tape and stack in one unit.

(*) Read Write head points the cell in the input tape where the finite control want to perform operation

(*) Readwrite head can move in the both the direction

(*) finite control control all the operation through read / write head as per requirement.

Defining Turing Machine :-

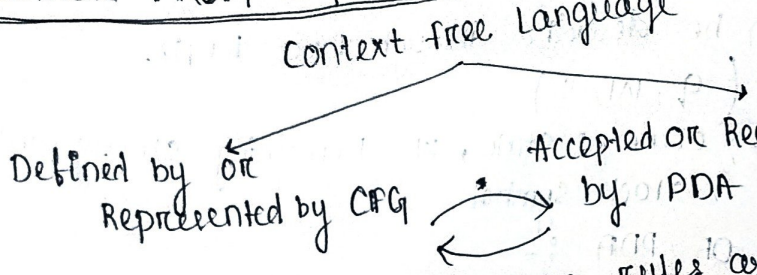
* Turing machine can also define through 7 tuples.

$$\langle Q \; \Sigma \; \delta \; q_0 \; F \; B \; \Gamma \rangle$$

* Q - finite set of states
* Σ - Set of input symbol

— Conversion —

CONVERSION FROM CFG to PDA



⇒ To convert the CFG to PDA two rules are apply as general form of CFG $A \to \alpha$ where A is single non-terminal value and α is any combination of terminal and non terminal value.

RULE:1

(1) for Non-terminal

$$\delta(q, \epsilon, A) \longrightarrow (q, \alpha)$$

RULE:2

(2) for terminal

$$\delta(q, a, a) \longrightarrow (q, \epsilon)$$

(Q) $S \to OS1 \mid 00 \mid 11$, convert this CFG to PDA.

$S \to OS1$	$\delta(q, \epsilon, S) \to (q, OS1)$ — ①
$S \to 00$	$\delta(q, \epsilon, S) \to (q, 00)$ — ②
$S \to 11$	$\delta(q, \epsilon, S) \to (q, 11)$ — ③

for terminals

$\delta(q, 0, 0) \to (q, \epsilon)$ — ④

$\delta(q, 1, 1) \to (q, \epsilon)$ — ⑤

(Q) check for the string 0111 for acceptance or Rejection.

$\delta(q, 0111, SZ_0)$ ① $\delta(q, 0111, OS1Z_0)$ ④

$\vdash \delta(q, 111, S1Z_0)$ ③

$\vdash \delta(q, 111, 111Z_0)$ ⑤

$\vdash \delta(q, 11, 11Z_0)$ ⑤

$\vdash \delta(q, 1, 1Z_0)$ ⑤

$\vdash \delta(q, \epsilon, Z_0)$ (ACCEPT)

(Q) $S \to OSA \mid OA \mid 1B$

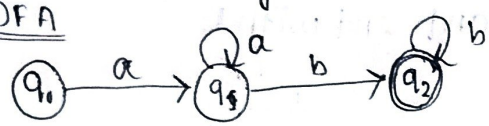
$A \to 1A \mid 0$

$B \to OB \mid 1$

$w =$ "0001000"

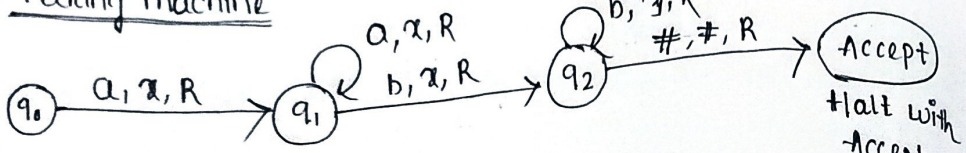
(Q) Design a turing machine for $L = \{a^n b^m \mid n, m > 0\}$

DFA

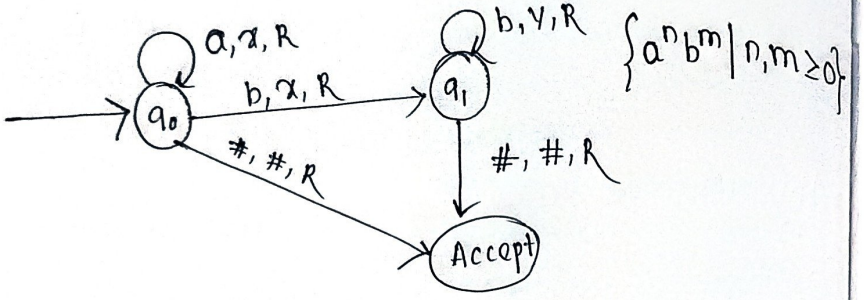


Turing machine

$\{a^n b^m \mid n, m > 0\}$

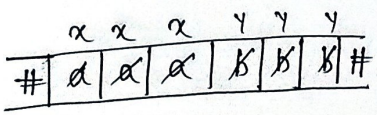


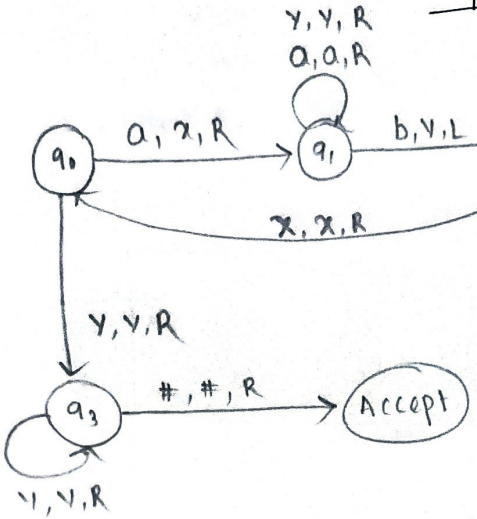
** Halt with Reject - dead state



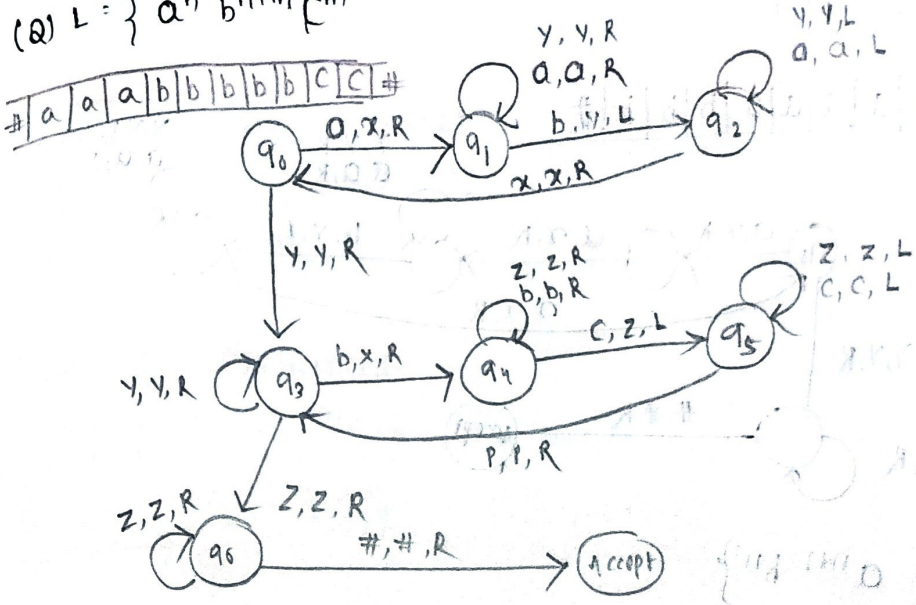
$\{a^n b^m \mid n, m \geq 0\}$

(Q) $L = \{a^n b^n \mid n > 0\}$

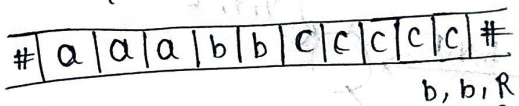


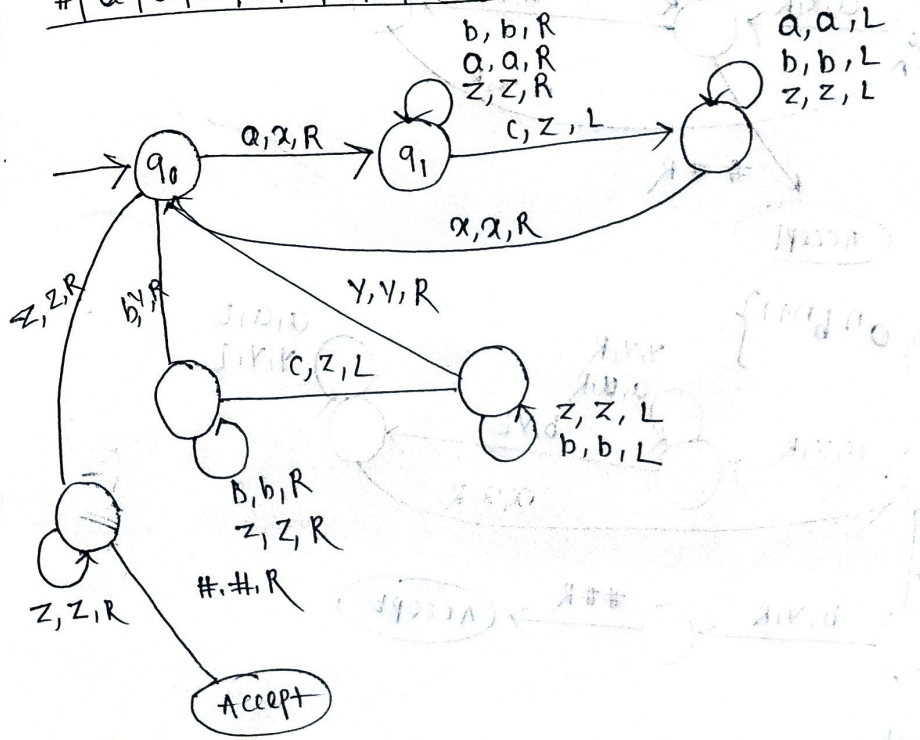


(Q) $L = \{a^n b^{n+m} c^m\}$

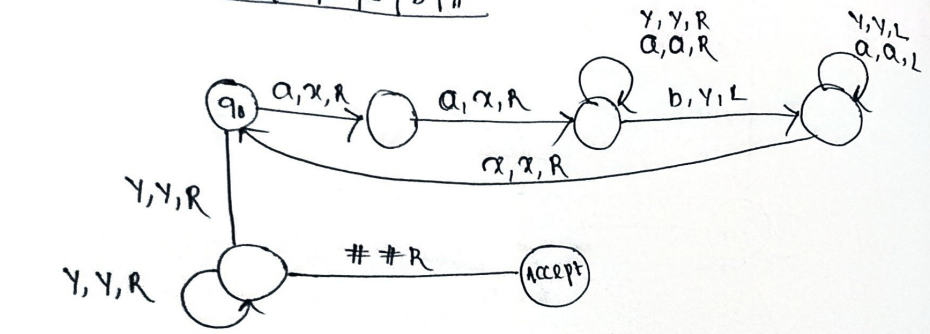


(Q) $L = \{a^n b^m c^{m+n}\}$



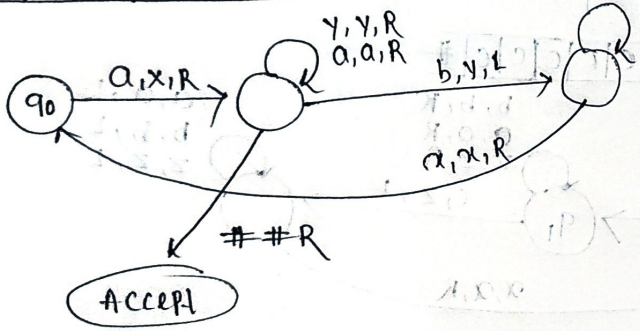


#	a	a	a	a	a	a	b	b	b	#

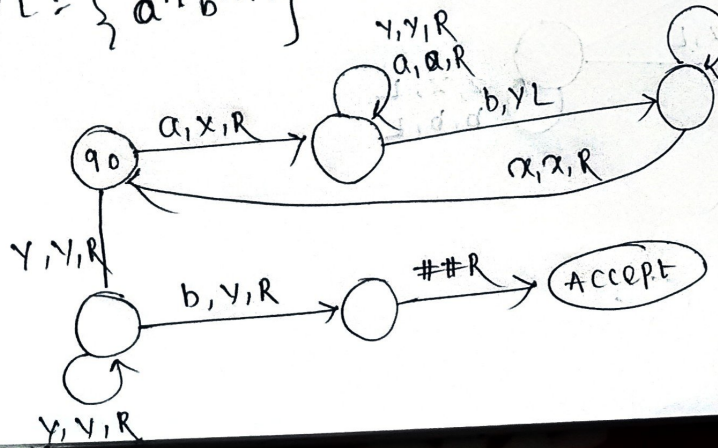


(Q) $L = \{ a^{n+1} b^n \}$

#	a	a	a	a	b	b	b	#



(Q) $L = \{ a^n b^{n+1} \}$



Turing Machine can also used for Evaluation.

Evaluation

1) Find 1's complement
2) Find 2's complement
3) Addition of two number

(1) Find 1's Complement

#	0	1	0	0	1	0	#