→ UNIT-II :- REGULAR EXPRESSIONS & REGULAR LANGUAGES.

→ Regular Expressions: Finite Automata & Regular Expressions, Applications of R.E, Algebraic Laws for Regular expressions, Conversion of finite Automata to Regular expressions.

→ Pumping Lemma for Regular Languages, Statement of Pumping Lemma, Applications of pumping Lemma.

→ Closure properties of Regular Languages: Closure properties of Regular languages, Decision properties of Regular Languages, Equivalence & Minimization of Automata.

→ Regular Expressions:-

The languages accepted by Finite Automata (FA) are regular languages and these languages are easily described by simple expressions called Regular Expressions (R.E). We have some algebraic notations to represent the R.E.

→ The Regular expressions are useful to represent the certain sets of strings in algebraic manner and R.E describes the languages accepted by FA.

→ If Σ is an alphabet then R.E over this can be described by following Rules:-

1. Any symbol from Σ, $\in$ and ϕ are Regular expressions. When we view 'a' in Σ as a R.E, then we denote it by 'a'.
2. The Union of two R.E's R_1 & R_2 written as $R_1 \cup R_2$ (or) $R_1 + R_2$ is also a R.E.
3. The concatenation of two R.E's R_1 & R_2, written as $R_1 R_2$ is also a R.E.
4. The Kleene closure (or Iteration) of a R.E denoted by R^* is also a R.E.
5. If R is a R.E, then (R) is also a R.E.
6. The Regular expressions obtained by applying rules 1 to 5 once or more than once are also R.E's.

Examples:-

(1) If $\Sigma = \{a, b\}$ then

a) a is a R.E (Using Rule 1)

b) b is a R.E (Using Rule 1)

c) a+b is a R.E (Using Rule 2)

d) b^* is a R.E (Using Rule 4)

e) ab is a R.E (Using Rule 3)

f) $ab + b^*$ is a R.E, (Using Rule 6).

(2) Find the Regular Expression for the following:-

a) A language consisting of all the words over {a,b} ending in b.

Sol:-
Regular expression for given language is $(a+b)^*b$

(b) A language consisting of all the words over {a,b} ending in bb.

Sol:-
R.E for given language is $(a+b)^*bb$.

(c) A language consisting of all the words over {a,b} starting with 'a' & ending with 'b'.

sol:-
R.E for given language is $a(a+b)^*b$.

(d) A language consisting of all the words over {a,b} having "bb" as substring.

Sol:-
R.E for given language is $(a+b)^*bb(a+b)^*$

(e) A language consisting of all the words over {a,b} ending with "aab".

Sol:-
R.E for given language is $(a+b)^*aab$.

→ Let $\Sigma = \{a,b\}$ and all the words over

$\Sigma = \{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, \dots\} =$
Σ^* or $(a+b)^*$ or $(a \cup b)^*$ which denotes all strings of a's & b's.

→ The table shows the R.E's & language corresponding to these R.E's.

Regular Expression	Meaning
$(a+b)^*$	set of strings of a's & b's of any length including the NULL string.
$(a+b)^*abb$	set of strings of a's & b's ending with string abb
$ab(a+b)^*$	set of strings of a's & b's starting with the string "ab".
$(a+b)^*aa(a+b)^*$	set of strings of a's & b's having a substring "aa".
$a^*b^*c^*$	set of strings consisting of any no of a's (may be empty string also) followed by any no. of b's (may be empty string also) followed by any no of c's (may include empty string also).
$a^+b^+c^+$	Set of strings consisting of atleast one 'a' followed by string consisting of atleast one 'b' followed by string consisting of atleast one 'c'.
$aa^*bb^*cc^*$	Set of strings consisting of atleast one 'a' followed by string consisting of atleast one 'b' followed by string consisting of atleast one 'c'.
$(a+b)^*(a+bb)^*$	set of strings of a's & b's ending with either "a" or "bb".
$(aa)^*(bb^*)^*b$	Set of strings consisting of even no of 'a's' followed by odd number of 'b's'.
$(0+1)^*000$	Set of strings consisting of 0's and 1's ending with 3 consecutive zero's i.e 000.
$(11)^*$	set consisting of even number of 1's.

→ For the given expression (R.E), what type of language is generated?

→ ① $r = \phi$, $L(r) = \{\}$

② $r = \epsilon$, $L(r) = \{\epsilon\}$

③ $r = a$, $L(r) = \{a\}$

④ $r = a+b$, $L(r) = \{a, b\}$ (Either a or b is generated not both)

⑤ $r = ab$, $L(r) = \{ab\}$

⑥ $r = a+b+c$, $L(r) = \{a, b, c\}$

⑦ $r = (ab+a)\cdot b$, $L(r) = \{abb, ab\}$

⑧ $r = a^+$, $L(r) = \{a, aa, aaa, \cdots\}$

⑨ $r = a^*$, $L(r) = \{\epsilon, a, aa, aaa, \cdots\}$

⑩ $r = (a+ba)(b+a)$, $L(r) = \{ab, aa, bab, baa\}$

⑪ $r = (a+\epsilon)(b+\phi)$, $L(r) = \{ab, b\}$

For $a+\epsilon$, Either u can generate a or ϵ, $\therefore a+\epsilon = \{a, \epsilon\}$

but for $b+\phi$ no choice of generating ϕ as it is null.

$\therefore b+\phi = \{b\}$

⑫ $r = (a+b)^2$, $L(r) = \{aa, ab, ba, bb\}$

$(a+b)^2 = (a+b)(a+b) = \{aa, ab, ba, bb\}$

⑬ $r = (a+b)^*$, $L(r) = \{\epsilon, a, b, aa, ab, ba, bb, \cdots\}$

$(a+b)^* = (a+b)^0 = \epsilon$, $(a+b)^1 = \{a, b\}$, $(a+b)^2 = \{aa, ab, ba, bb\}$

⑭ $r = (a+b)^*(a+b)$, $L(r) = (a+b)^+$

$(a+b)^*(a+b) = \{\epsilon, a, b, aa, ba, ab, bb, \cdots\} \cdot \{a+b\}$

$= \{a, b, aa, ba, ab, bb, \cdots\}$ i.e $(a+b)^+$.

⑮ $r = a^*.a^*$, $L(r) = a^*$, $\because \{\epsilon, a, aa, \cdots\} \cdot \{\epsilon, a, aa \cdots\} = a^*$

⑯ $r = (ab)^*$, $L(r) = \{\epsilon, ab, abab, ababab, \cdots\}$

$\because (ab)^* = \{(ab)^0 . (ab)^1, (ab)^2 \cdots\}$.

→ Regular set:-

Any set represented by a Regular expression is called a Regular set.

Ex:- $a, b \in \Sigma$, then

1) 'a' denotes the set $\{a\}$
2) a+b denotes the set $\{a, b\}$
3) ab denotes the set $\{ab\}$
4) a^* denotes the set $\{\in, a, aa, aaa, \ldots\}$
5) $(a+b)^*$ denotes the set $\{\in, a, b, ab, ba, aa, bb, aaa, aab, \ldots\}$

→ Language associated with Regular Expressions:-

Let R_1 and R_2 denote any R.E's then.

① A string in $L(R_1+R_2)$ is a string from R_1 or R_2 - Union.

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

② A string in $L(R_1R_2)$ is a string from R_1 followed by a string from R_2 - Concatenation:

$$L(R_1R_2) = L(R_1) \cdot L(R_2).$$

③ A string in $L(R^*)$ is a string obtained by concatenating n elements for some $n \geq 0$.

$$L(R^*) = (L(R^*)) = \bigcup_{n=0}^{\infty} L(R)^n.$$

From the above,

① The set R_1+R_2 represents the union of the sets represented by R_1 and R_2.
② The set R_1R_2 represents the concatenation of the sets represented by R_1 & R_2.
③ The set R^* is $\{w_1w_2 \ldots w_n \mid w_i$ is in the set represented by R and $n \geq 0\}$.

→ Note:- By the definition of R.E, the class of Regular sets over Σ is called closed under Union, concatenation and Closure (Iteration) by the conditions 2, 3, 4.

→ Describe the following sets by Regular Expressions:-

① $\{101\}$

101 is obtained by concatenating 1, 0 & 1 $\therefore \{101\}$ is represented by 101.

② $\{01, 10\}$

As $\{01, 10\}$ is union of $\{01\}$ & $\{10\}$, so $\{01, 10\}$ is represented by $01+10$.

③ $\{abb, a, b, bba\}$

The set is represented by $abb+a+b+bba$

④ $\{\epsilon, 0, 00, 000, 000 \cdots\}$

The above set is represented by 0^*

⑤ $\{1, 11, 111, 1111 \cdots\}$

The above set can be obtained by concatenating 1 & any element of $\{1\}^*$. Hence $\{1, 11, 111 \cdots\}$ is represented by $1(1)^*$.

(or)

$\{1, 11, 111, 1111 \cdots\}$ can be represented by $(1)^+$

→ Operations on R.E:-

① Union:- Let L_1 and L_2 be languages by NFA M_1 & M_2 respectively then the Union is represented by $L_1 \cup L_2$.

Ex:- $L_1 = \{001, 10, 111\}$.

$M = \{\epsilon, 011\}$ then

$L \cup M = \{\epsilon, 10, 001, 111, 011\}$

② Concatenation:-

Let L_1 and L_2 be languages accepted by NFA M_1 & M_2 respectively then the concatenation is represented by $L_1 L_2$.

Ex:- $L = \{001, 10, 111\}$

$M = \{\epsilon, 001\}$ then

$L \cdot M = \{001, 001001, 10, 10001, 111, 111001\}$.

③ Kleene(star) closure:-

Let M_1 be NFA, then NFA M_1 is constructed such that $L(M) = (L(M_1))^*$.

Ex:- $L = \{0, 11\}$ then

$L^* = \{\epsilon, 0, 11, 011, \dots\}$

④ Complementation:-

Complementary language for a language $L(M)$ is given by $\Sigma - L(M)$.

Ex:- L_1 = All strings ending in 'a'.

$\bar{L}_1$ = All strings not ending in 'a'.

⑤ Intersection:- $(L_1 \cap L_2)$.

L_1 = All words begin with 'a' → a(ath)*

L_2 = All words end with 'a' → (ath)*a

→ a(ath)*a

→ Algebraic Laws (or) Identities for Regular expressions:-
We give identities for Regular expressions which are useful for simplifying Regular expressions. [The two R.E's P and Q are equivalent (P=Q) if & only if 'P' represents the same set of strings as 'Q' does. For showing this equivalence of R.E's we need to show some identities of R.E's.]

→ $I_1 : \phi + R = R$.

→ $I_2 : \phi R = R\phi = \phi$

→ $I_3 : \epsilon R = R\epsilon = R$

→ $I_4 : \epsilon^* = \epsilon$ and $\phi^* = \epsilon$

→ $I_5 : R + R = R$

→ $I_6 : R^* R^* = R^*$

→ $I_7 : RR^* = R^*R = R^+$

→ $I_8 : (R^*)^* = R^*$

→ $I_9 : \epsilon + RR^* = R^* = \epsilon + R^*R$

→ $I_{10} : (PQ \vee)^* P = P(QP)^*$

→ $I_{11} : (P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

→ $I_{12} : (P+Q)R = PR + QR$ and
$R(P+Q) = RP + RQ$.

→ Arden's Theorem:-
Let P and Q be two regular expressions over Σ. If P does not contain ϵ, then the following equation in R, namely

$R = Q + RP$ has a unique solution given by

$R = QP^*$

→ [Equivalence of two R.E's:- Let us see one important theorem named Arden's Theorem which helps in checking the equivalence of two R.E's]

→ Ex:-1 :-

Prove that $(1+011)^* = \epsilon + 1^*(011)^*(1^*(011)^*)^*$

Sol:-

Let R.H.S :- $\Rightarrow \epsilon + \underbrace{1^*(011)^*}_{R}\underbrace{(1^*(011)^*)^*}_{R^*}$

$\Rightarrow \epsilon + RR^* = R^*$ [by Iq]

$\rightarrow (1^*(011)^*)^*$

$\Rightarrow (P^*Q^*)^* \rightarrow (P+Q)^*$ [By Il]

$= (1+011)^*$

$\therefore$ L.H.S = R.H.S

Hence proved.

→ Ex :-2 :-

Prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

Sol:- L.H.S $\Rightarrow (1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$

$\rightarrow (1+00^*1)(\epsilon + (0+10^*1)^*(0+10^*1))$

$\epsilon + R^* \quad R$

$\rightarrow$ [By Iq $\Rightarrow \epsilon + R^*R = R^*$]

$\rightarrow (1+00^*1)[(0+10^*1)^*]$

$\rightarrow 1(\underbrace{\epsilon + 00^*}_{\epsilon+RR^*=R^*})(0+10^*1)^*$

$\rightarrow 10^*(0+10^*1)^*$ [By Iq]

$\therefore$ L.H.S = R.H.S

Hence proved.

⇒ Arden's method:-

→ Write equation for each state based on incoming edges.

→ Add ϵ to the equation of initial state.

→ Simply the equation using Arden's theorem and find Regular Expression for final state.

Conditions:-

→ FA should not contain ϵ- Transitions.

→ FA should have only one initial state.

Ex-1:- Construct R.E for the given FA

Sol:- Since there is only one state in the FA let us solve for q_0 only as it is a final state.

$$q_0 = q_00 + q_01 + \epsilon$$

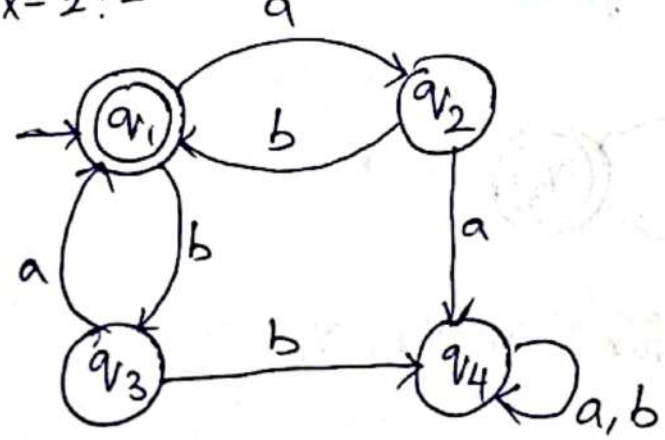
$$q_0 = q_0(0+1) + \epsilon \Rightarrow q_0 = \underset{Q}{\epsilon} + \underset{R}{q_0}\underset{P}{(0+1)}$$

$q_0 = \epsilon \cdot (0+1)^*$ [By Arden's theorem, $R = Q + RP$, $R = QP^*$]

$q_0 = (0+1)^*$

∴ The R.E for the given FA is $R = (0+1)^*$.

→ Ex-2:-



Sol:-
We can apply the above method directly since the graph does not have ϵ-moves and there is only one initial state.
We get the following equations for q_1, q_2, q_3, q_4:-

$$q_1 = q_2 b + q_3 a + \epsilon$$
$$q_2 = q_1 a$$
$$q_3 = q_1 b$$
$$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$$

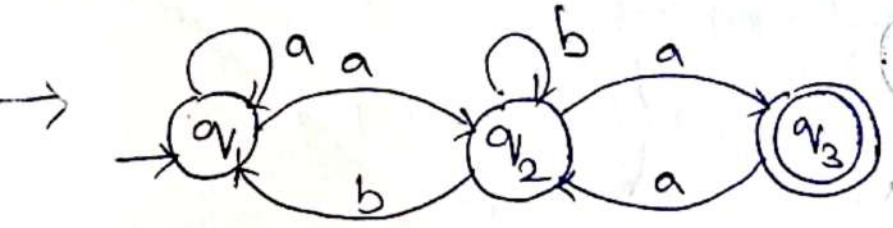
→ As q_1 is the final state and q_1 equation involves only q_2 and q_3. We use only q_2 and q_3 equations.
→ Substituting eq's of q_2 and q_3 in q_1 we get

$$q_1 = q_1 ab + q_1 ba + \epsilon$$
$$q_1 = \epsilon + q_1(ab + ba)$$
$$\downarrow R \quad \downarrow Q \quad \downarrow R \quad \downarrow P$$

By applying Arden's theorem, we get for $R = Q + RP$ as $R = QP^*$.

$$q_1 = \epsilon \cdot (ab+ba)^*$$
$$\therefore q_1 = (ab+ba)^* \quad \text{i.e } R = (ab+ba)^*.$$

∴ The R.E accepted by given FA is $(ab+ba)^*$.



Sol:-
We get following eq's for q_1, q_2, q_3

$$q_1 = q_1 a + q_2 b + \epsilon$$
$$q_2 = q_1 a + q_2 b + q_3 a$$
$$q_3 = q_2 a$$

By substituting q_3 in the q_2 - Eq'n $\quad R = Q + RP$, $R = QP^*$

$$q_2 = q_1a + q_2b + q_2aa$$

$$\underset{R}{q_2} = \underset{Q}{q_1a} + \underset{R}{q_2}\underbrace{(b+aa)}_{P} \Rightarrow \therefore \boxed{q_2 = q_1a(b+aa)^*}$$

Substituting q_2 in q_1 - Eq'n

$$q_1 = q_1a + q_2b + \epsilon$$

$$q_1 = q_1a + q_1a(b+aa)^*b + \epsilon$$

$$\underset{R}{q_1} = \underset{R}{q_1}\underbrace{(a + a(b+aa)^*b)}_{P} + \underset{Q}{\epsilon}$$

$$\rightarrow \boxed{q_1 = \epsilon(a+a(b+aa)^*b)^*}$$

$$\rightarrow \boxed{q_2 = (a+a(b+aa)^*b)^*a(b+aa)^*} \quad [\because q_2 = q_1a(b+aa)^*]$$

$$\rightarrow \boxed{q_3 = (a+a(b+aa)^*b)^*a(b+aa)^*a}$$

Since q_3 is a final state, the set of strings recognized by the graph is given by

$$(a+a(b+aa)^*b)^*a(b+aa^*)a$$

→ Ex-3:-

Sol:- We can apply Arden's Theorem

$$q_1 = q_10 + \epsilon$$

$$q_2 = q_11 + q_21$$

$$q_3 = q_20 + q_30 + q_31 \rightarrow$$

$$\underset{R}{q_1} = \underset{R}{q_1}\underset{P}{0} + \underset{Q}{\epsilon} \Rightarrow q_1 = \epsilon 0^* = 0^* \rightarrow \boxed{q_1 = 0^*}$$

$$q_2 = q_11 + q_21 \Rightarrow \underset{R}{q_2} = \underbrace{0^*1}_{Q} + \underset{R}{q_2}\underset{P}{1}$$

$$\boxed{q_2 = 0^*11^*}$$

Note:- As the final states are q_1 and q_2, we need not solve for q_3:

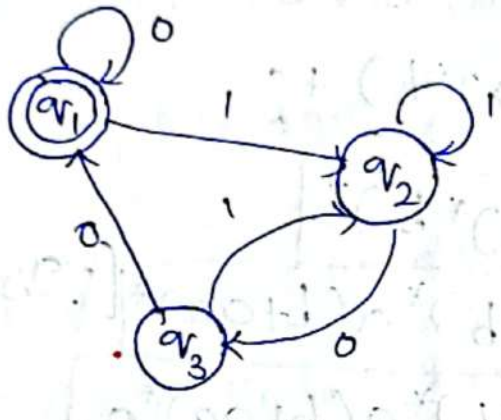
$q_1 + q_2 = 0^* + 0^*1 \cdot 1^*$

$= 0^*(\epsilon + 11^*) \rightarrow \epsilon + RR^* = R^* \rightarrow$ Eq.

$\therefore R \cdot \epsilon = 0^* 1^*$

∴ The string represented by transition graph are $0^* 1^*$

→ Ex-4:-



Sol:- $q_1 = q_1 0 + q_3 0 + \epsilon$

$q_2 = q_1 1 + q_2 1 + q_3 1$

$q_3 = q_2 0$ ⟶ substitute q_3 in q_2

$\Rightarrow q_2 = q_1 1 + q_2 1 + q_2 01$

$q_2 = q_1 1 + q_2 (1 + 01)$

$R \quad Q \quad R \quad P$

$\boxed{q_2 = q_1 1 (1 + 01)^*}$

$\Rightarrow q_3 = q_2 0$

$\boxed{q_3 = q_1 1 (1 + 01)^* 0}$

$\Rightarrow q_1 = q_1 0 + q_3 0 + \epsilon$

$q_1 = q_1 0 + q_1 1 (1 + 01)^* 00 + \epsilon$

$q_1 = q_1 (0 + 1(1 + 01)^* 00) + \epsilon$

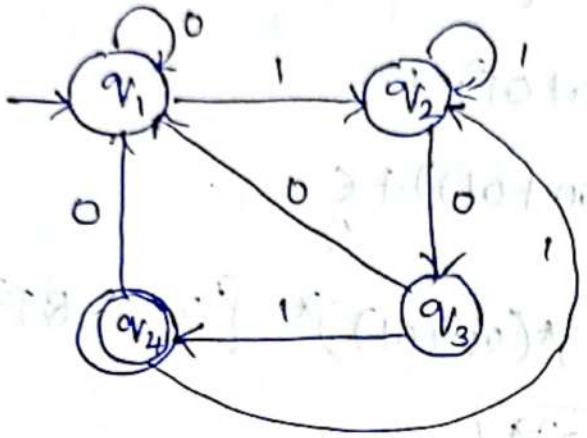
$R \quad R \quad P \quad Q$

$q_1 = \epsilon(0+1(1+01)^*00)^*$

$\boxed{q_1 = (0+1(1+01)^*00)^*}$

As q_1 is the only final state, the R.E corresponding to the given diagram is $(0+1(1+01)^*00)^*$.

→ Ex-5:-



Sol:-

$q_1 = q_1 0 + q_3 0 + q_4 0 + \epsilon$

$q_2 = q_1 1 + q_2 1 + q_4 1$

$q_3 = q_2 0$

$q_4 = q_3 1$

Now, $q_4 = q_3 1$ [Substitute q_3 in q_4]

$\boxed{q_4 = (q_2 0)1 = q_2 01}$

$q_2 = q_1 1 + q_2 1 + q_4 1$

$= q_1 1 + q_2 1 + q_2 01 1$ (Substitute q_4 in q_2)

$q_2 = \underset{Q}{q_1 1} + \underset{R}{q_2}\underset{P}{(1+011)}$

$\boxed{q_2 = q_1 1 (1+011)^*}$

$q_1 = q_1 0 + q_3 0 + q_4 0 + \epsilon$ (Substitute q_3 & q_4 in q_1)

$= q_2 00 + q_2 01 1 0 + \epsilon$

$q_1 = q_1 0 + q_2 1 (1+011)^* 00 + q_2 10 + \epsilon$

Ex-6:-

$\Rightarrow q_1 = q_1 0 + q_2 00 + q_2 01 + \epsilon$

$q_1 = q_1 0 + q_2 (00+01) + \epsilon$ [substitute q_2 in q_1]

$q_1 = q_1 0 + q_1 1 (1+011)^* (00+01) + \epsilon$

$q_1 = q_1 (0+1(1+011)^*(00+01)) + \epsilon$

R R P Q

$q_1 = \epsilon . (0+1(1+011)^*(00+01))^*$ [$\because R = QP^*$]

$\boxed{q_1 = (0+1(1+011)^*(00+01))^*}$

$\Rightarrow q_3 = (q_2 0)$

$q_3 = q_1 1 (1+011)^*$

$\boxed{q_3 = (0+1(1+011)^*(00+01))^* 1 (1+011)^*}$

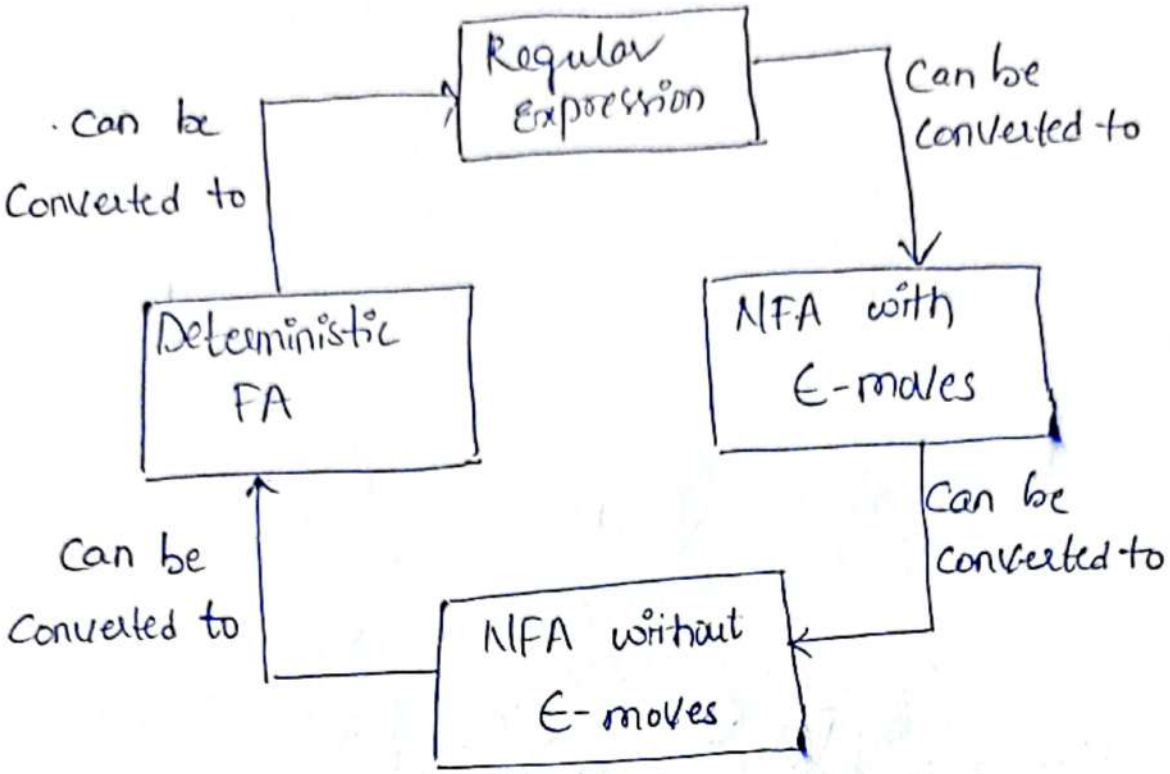
$\Rightarrow q_4 = q_3 1$ [substitute q_3 in q_4]

$\boxed{q_4 = (0+1(1+011)^*(00+01))^* 1 (1+011)^* 1}$

As q_4 is the final state, the expression generated is the R.E for the given finite Automata.

→ Relationship between FA and RE:-

There is a close relationship b/n FA and RE, we can show this relation in below figure:-



The above figure shows that it is convenient to convert the R.E to NFA with ∈ moves. Let us see the theorem based on this Conversion.

→ Conversion of R.E's to FA (or) Constructing FA for a given R.E's :- 1. Subset Method
2. Direct Method.

Theorem:- If 'R' be a regular expression then there exists a NFA with ∈-moves, which accepts L(R).

Proof:- First we will discuss the construction of NFA (M) with ∈-moves for R.E 'R', then we prove that L(M) = L(R).

Let 'R' be the Regular expression over the alphabet Σ.

→ Construction of NFA with ∈-moves

→ Case 1:-

(i) $R = \phi$ $\quad Q \quad \Sigma \quad \delta \quad q_0 \quad F$

NFA $M = (\{s, f\}, \{\}, \delta, s, \{f\})$ → (s) (f)

(No path from inital state to reach the final state f)

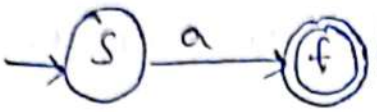
ii) $R = \epsilon$

NFA $M = (\{s\}, \{\epsilon\}, \delta, s, \{s\})$

(The initial state & final state is same).

iii) $R = a$, $a \in \Sigma$

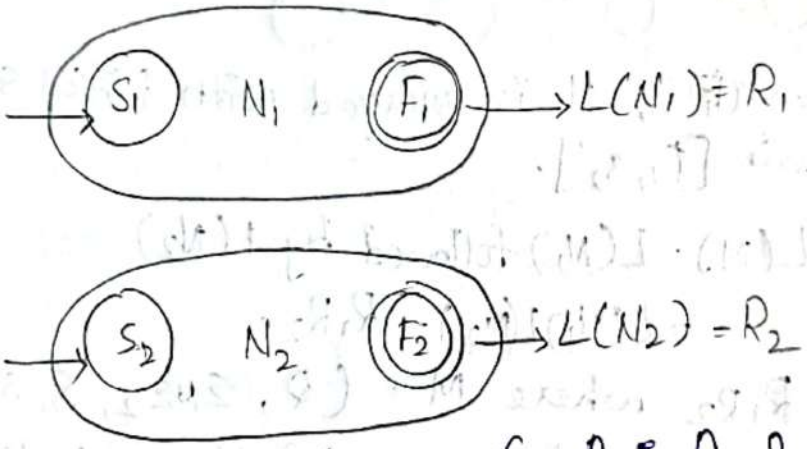
NFA $M = (\{s, f\}, \{a\}, \delta, s, \{f\})$



(One path is there from initial state s to reach the final state f with label a)

⇒ Case 2:- for $R \geq 1$

Let R_1 and R_2 be the two R.E's over Σ_1, Σ_2 and N_1 & N_2 are two NFA R_1 & R_2 respectively as shown in the figure.

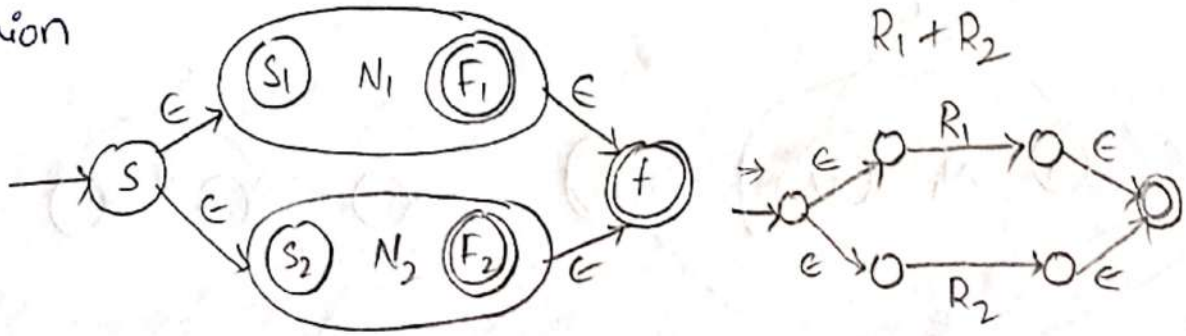


The above diagram is the NFA for R.E R_1 & R_2

→ Rule 1:- For Constructing NFA, M for $R = R_1 + R_2$ (or) $R_1 \cup R_2$

Let s and f are the starting state and final state respectively of M. Transition diagram of M is

Rule: Union



$L(M) = \in L(N_1) \in$ (or) $\in L(N_2) \in$

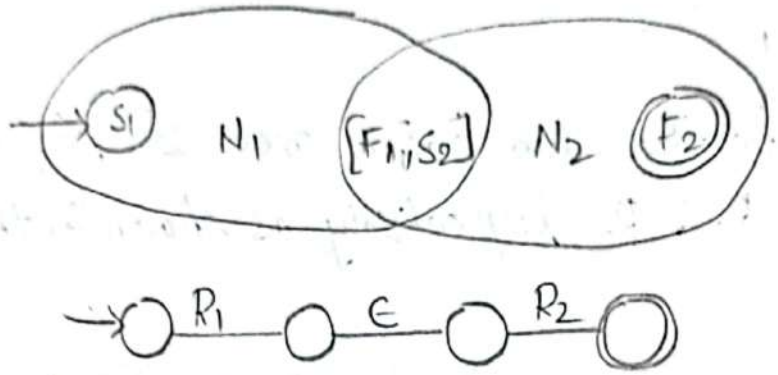
$= L(N_1)$ (or) $L(N_2) = R_1$ or R_2

So, $R = R_1 + R_2$

$M = (Q, \Sigma_1 \cup \Sigma_2, \delta, s, \{f\})$, where Q contains all the states of N_1 & N_2.

→ Rule 2:- Concatenation.

For Regular expression $R = R_1 R_2$, NFA M is shown as



The final state(F_1) of N_1 is merged with initial state(S_2) of N_2 into one state $[F_1, S_2]$.

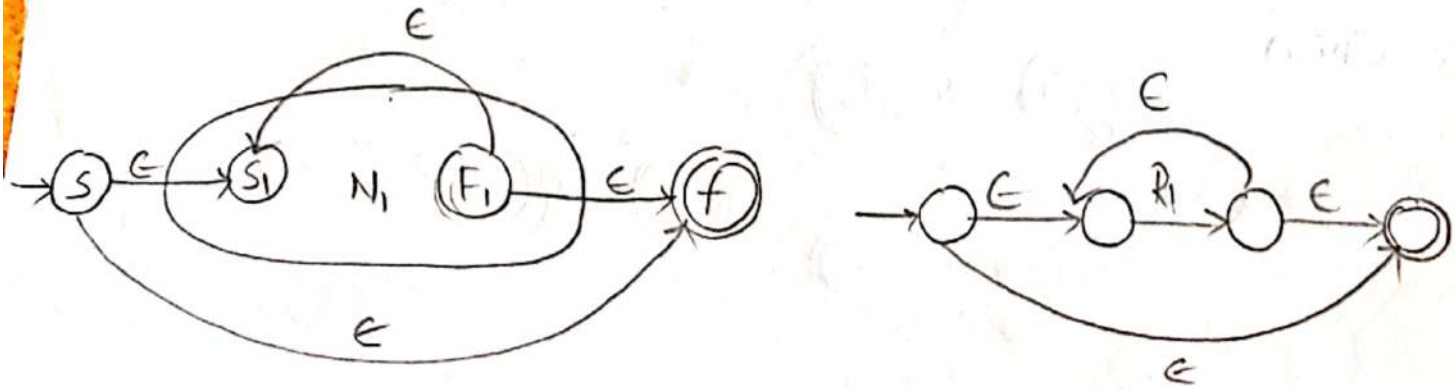
$L(M) = L(N_1)$ followed by $L(N_2)$

$= L(N_1) L(N_2) = R_1 R_2$.

So, $R = R_1 R_2$ where $M = (Q, \Sigma_1 \cup \Sigma_2, \delta, s_1, \{F_2\})$ and Q contains all the states of N_1 & N_2 such that final state of N_1 is merged with initial state of N_2.

→ Rule 3:- Kleene closure

For Regular expression $R = R_1^*$, NFA M is shown as



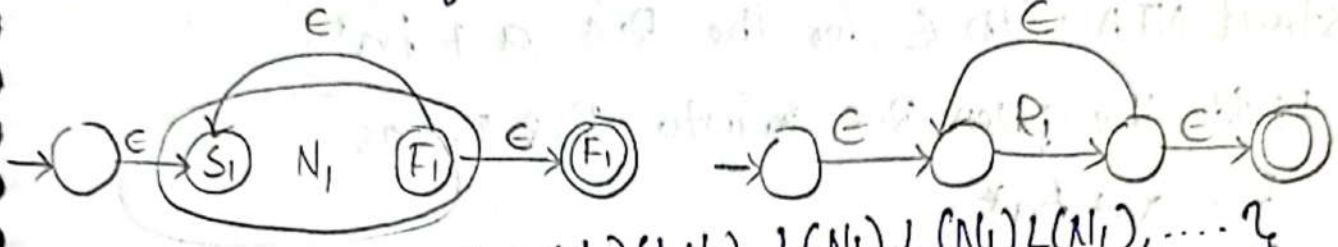
$L(M) = \{ \in, L(N_1), L(N_1)L(N_1), L(N_1)L(N_1)(LN_1), \cdots \}$

$= L(N_1)^* = R_1^*$

$\therefore M = (\{s, f\} \cup Q_1, \Sigma_1, \delta, s, \{f\})$, where Q_1 is the set of states of N_1.

→ Rule 4: positive closure:-

For construction of $R = R_1^+$, M is shown as



$L(M) = \{ L(N_1), L(N_1)(LN_1), L(N_1)L(N_1)L(N_1), \cdots \}$

$= L(N_1)^+ = R_1^+$

$M = (\{s, f\} \cup Q_1, \Sigma_1, \delta, s, \{f\})$, where Q_1 is set of states of N_1.

→ ∈-NFA for R_1^+ :-

This structure is for R^+ which means there must be atleast one R_1 in the expression. It is preceded by epsilon and also succeeded by one. There is epsilon feedback from v_2 to v_1 so that there can be more than one R_1 in the expression.

→ ∈-NFA for R_1^* :-

This structure is for R^* which means there can be any no of R_1 in the expression and ∈ also. The previous structure is just modified a bit so that even if there is no input symbol i.e if the i/p symbol is null, then also the expression is valid.

→ ∈-NFA for $R_1 + R_2$:-

This structure accepts either R_1 or R_2 as i/p. So there are two paths, both of which lead to the final state.

→ ∈-NFA for R_1R_2 :- For concatenation, R_1 must be followed by R_2, only then it can reach the final state. So, the ∈-moves is taken b/n R_1 & R_2.

→ ~~∈construct~~

→ Subset method:-

Step 1:- Construct NFA with ∈

Step 2:- Construct NFA with ∈ to NFA without ∈

Step 3:- Convert NFA to DFA.

→ Ex-1:-

Construct NFA with ∈ for the R.E $a + ba^*$

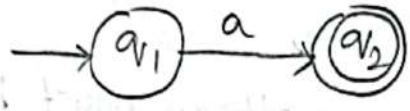
Sol:- Divide the given R.E into R_1 & R_2 as

$R = a + ba^*$

$R_1 = a$

$R_2 = ba^*$

Let us draw the NFA for R_1 as

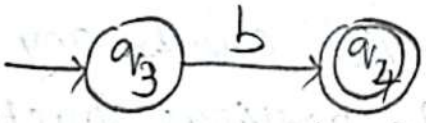


Now, we will divide R_2 into R_3 & R_4 as

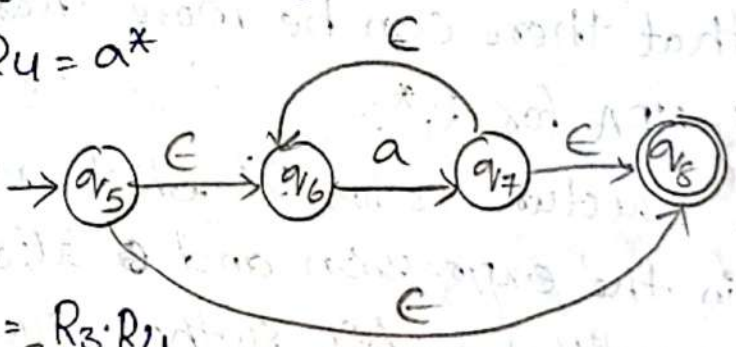
$R_3 = b$ & $R_4 = ba^*$

NFA for R_3 & R_4 are drawn as

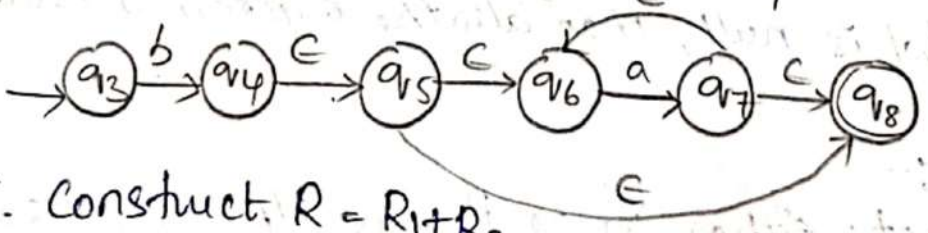
$R_3 = b$



$R_4 = a^*$

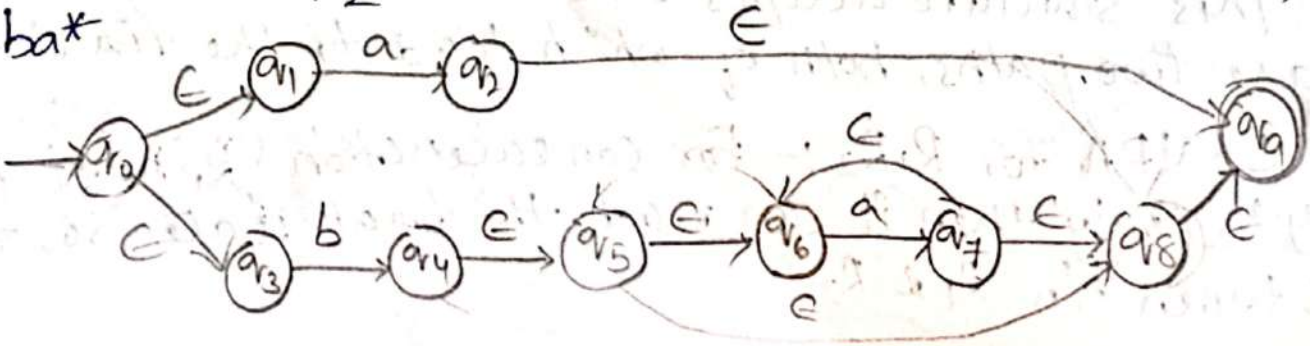


→ Now Construct R_2 i.e. $R_2 = R_3 \cdot R_4$



∴ Construct $R = R_1 + R_2$

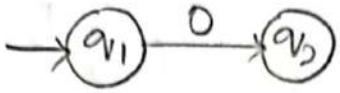
$R = a + ba^*$



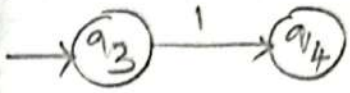
(2) → Construct NFA For the R.E $(0+1)^*$

Sol:- $R = (R_1 + R_2)^*$ i.e $R = (0+1)^*$

→ Construct $R_1 = 0$

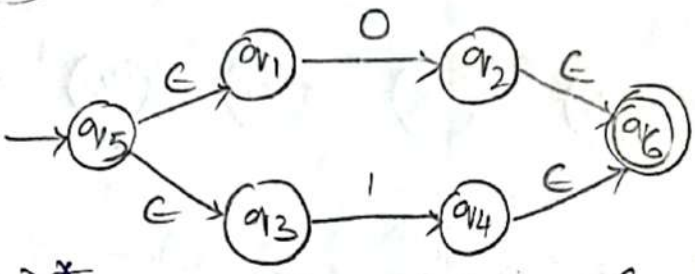


→ construct $R_2 = 1$

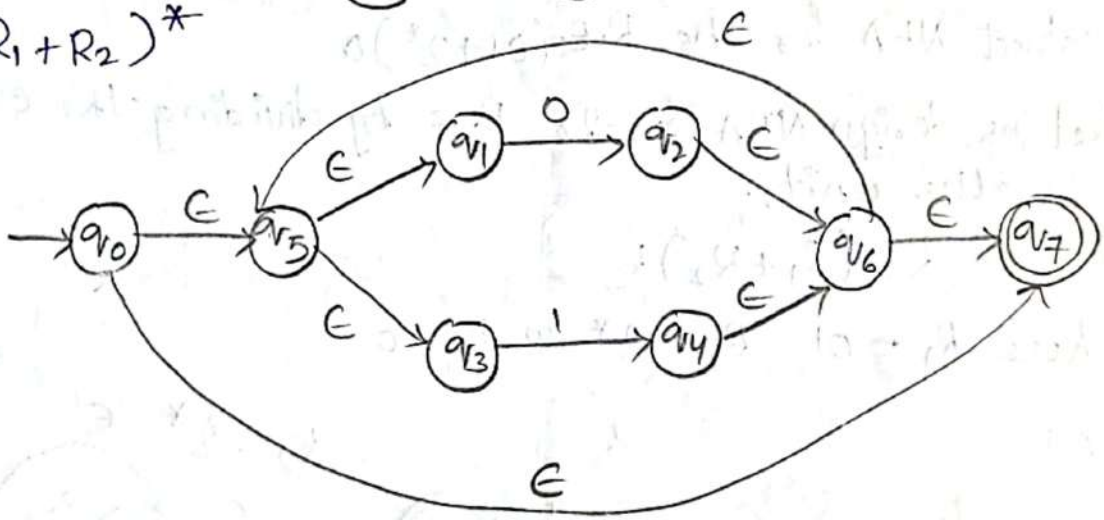


→ Now construct R i.e $R = (R_1 + R_2)^*$

$R_1 + R_2$



$R = (R_1 + R_2)^*$

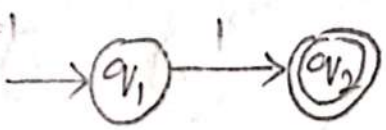


(3) → Construct NFA for the language having odd number of 1's over the set $\Sigma = \{1\}$.

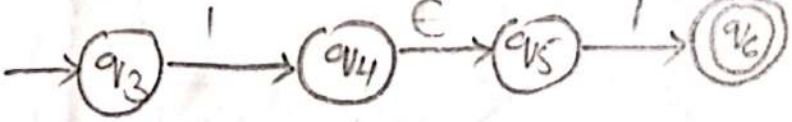
Sol:- The R.E is $1(11)^*$

Let $R_1 = 1$, $R_2 = 11$, $R_3 = (R_2)^*$

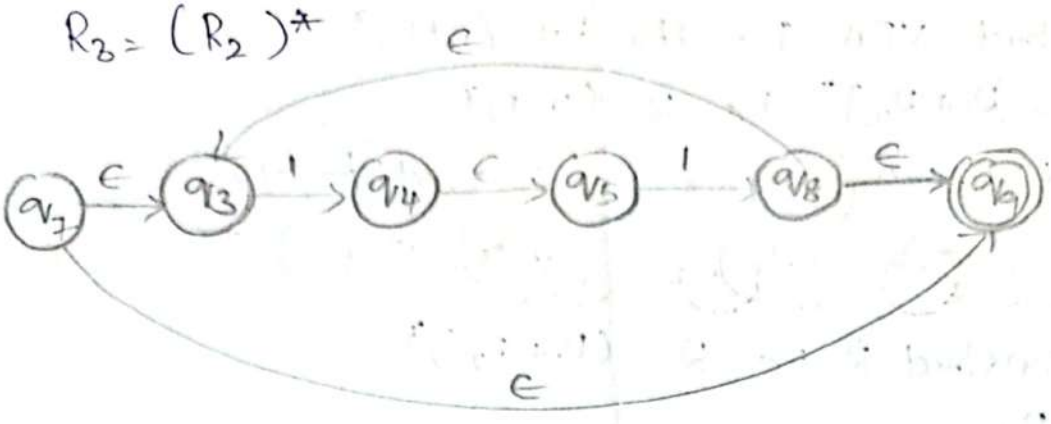
$R_1 = 1$



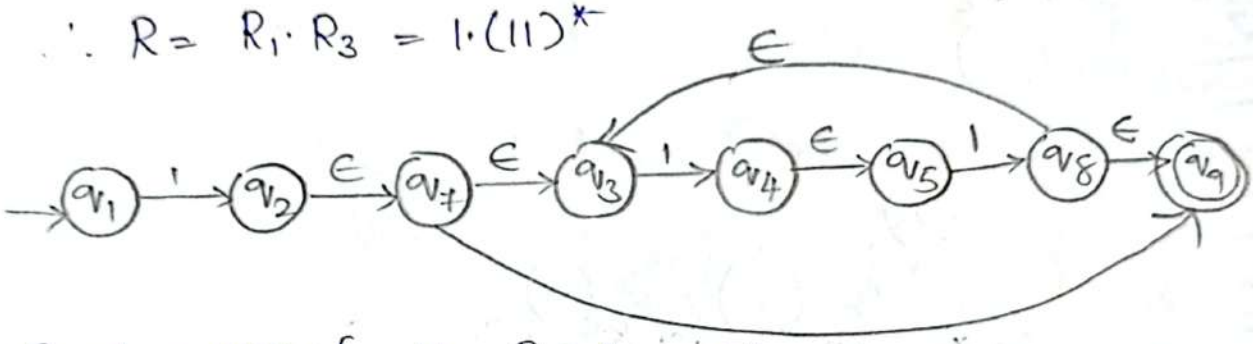
$R_2 = 11$



$R_3 = (R_2)^*$



$\therefore R = R_1 \cdot R_3 = 1 \cdot (11)^*$



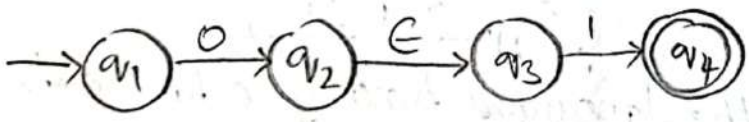
④ Construct NFA for the R.E: $(01+2^*)0$

Sol:- Let us design NFA for the R.E by dividing the expression into smaller units.

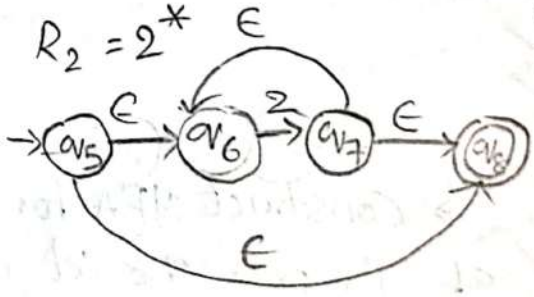
$$R = (R_1 + R_2) R_3$$

where $R_1 = 01$, $R_2 = 2^*$ & $R_3 = 0$

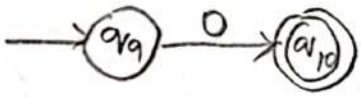
$R_1 = 01$



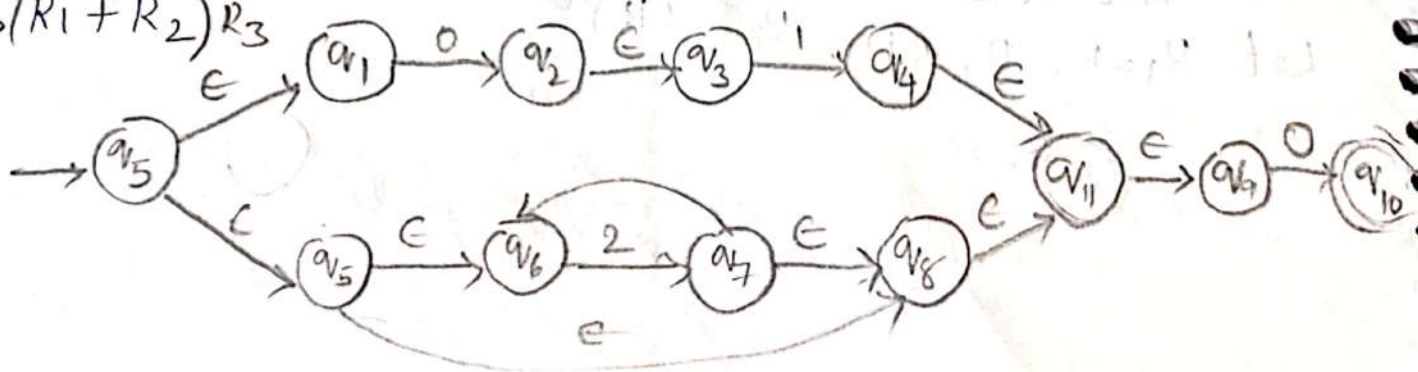
$R_2 = 2^*$



$R_3 = 0$



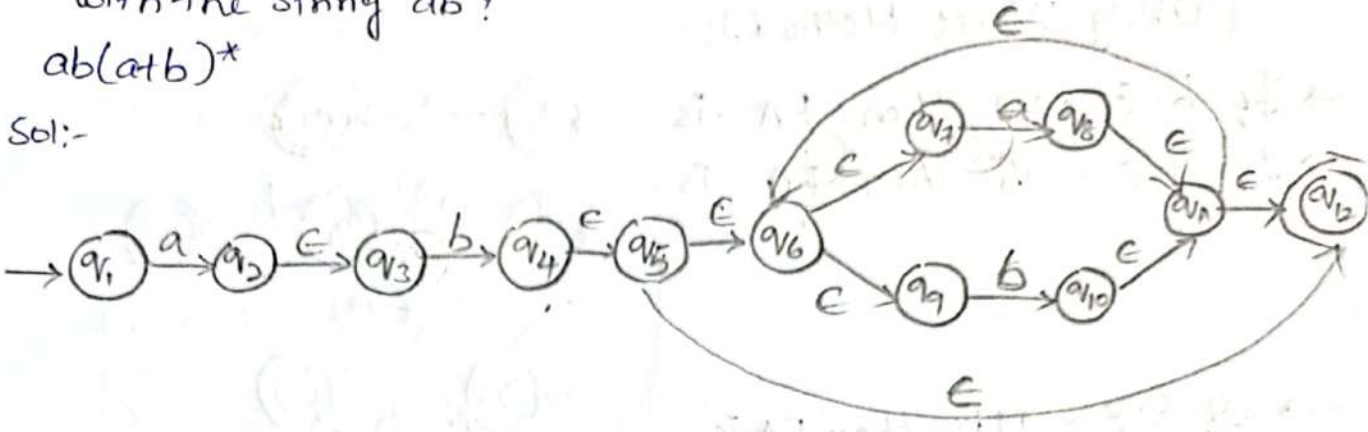
$\rightarrow (R_1 + R_2) R_3$



⑤ → Obtain NFA which accepts strings of a's & b's starting with the string ab?

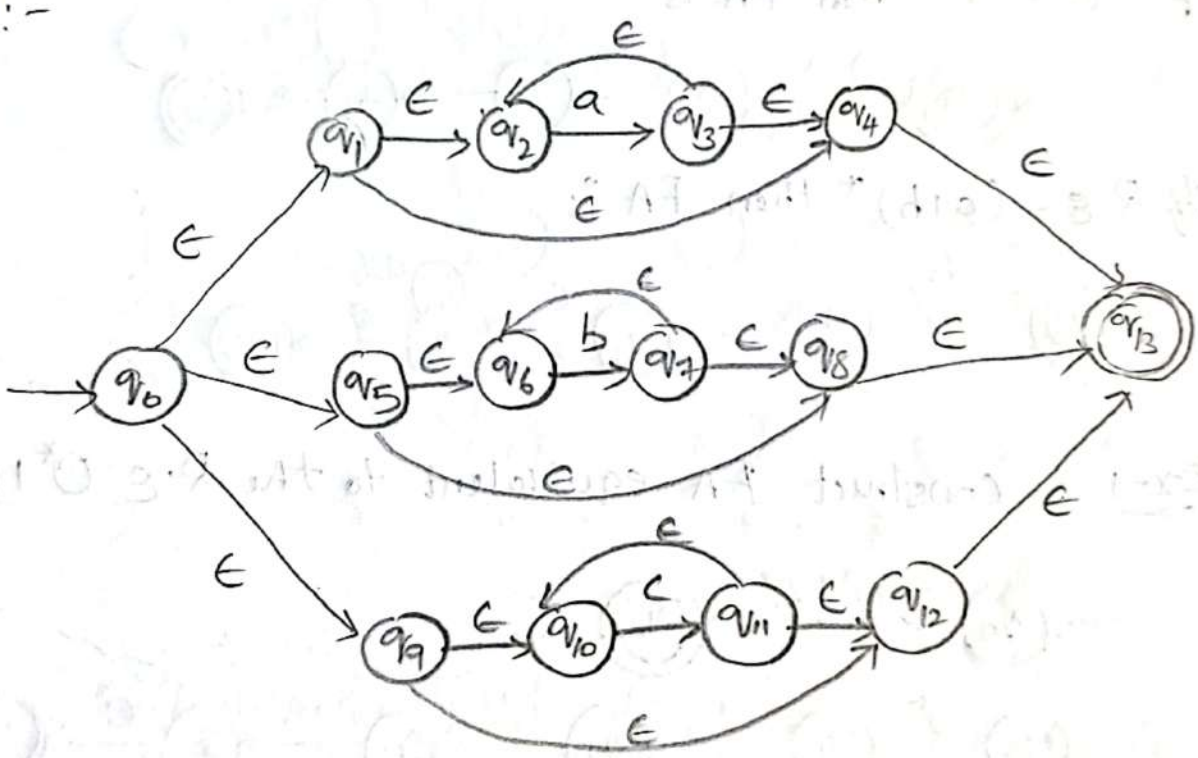
ab(a+b)*

Sol:-



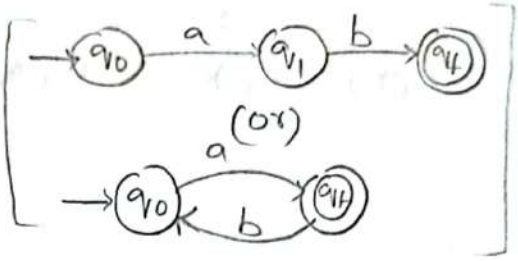
→ Obtain an NFA for the R.E $a^*+b^*+c^*$

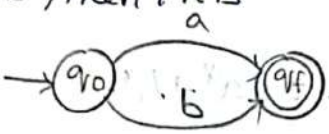
Sol:-

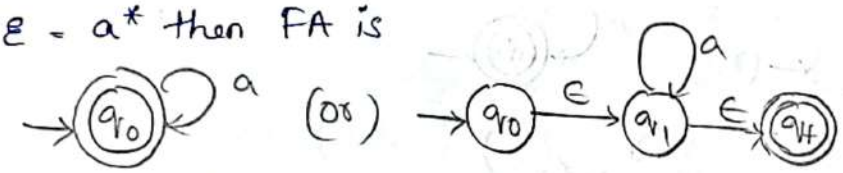


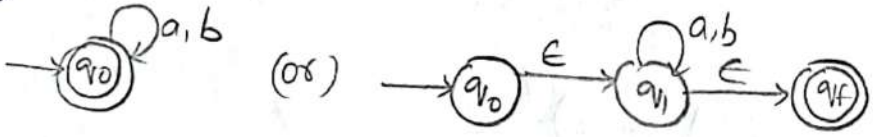
→ Conversion of Regular expression to Finite Automata (Using Direct Method):-

→ If R.E = a, then FA is 

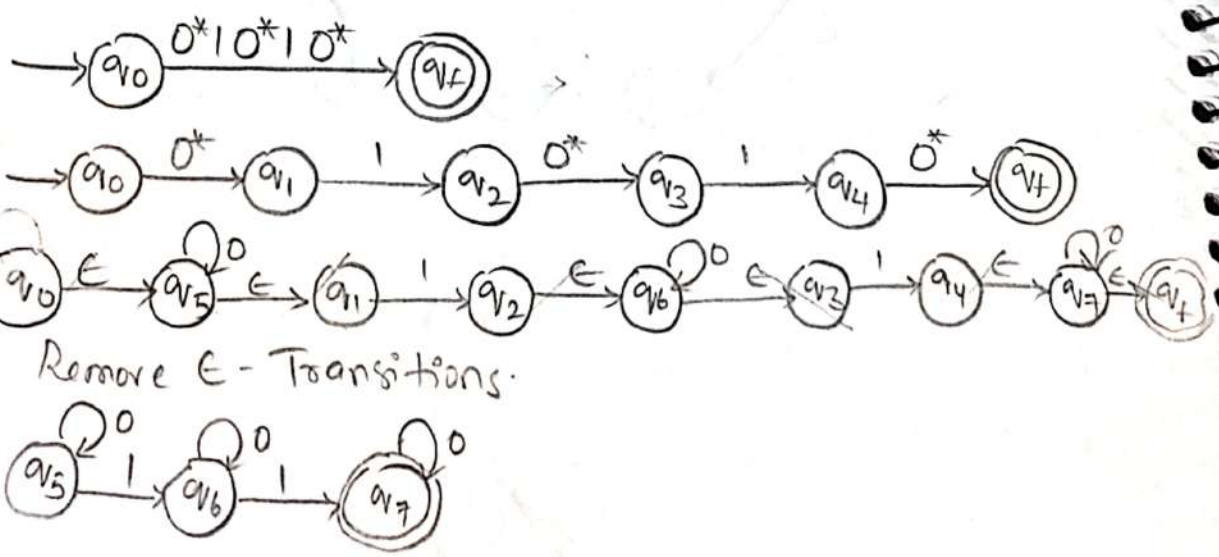
→ If R.E = ab, then FA is 

→ If R.E = a+b, then FA is 

→ If R.E = a* then FA is 

→ If R.E = (a+b)* then FA is 

→ Ex-1 :- Construct FA equivalent to the R.E 0*10*10*

Sol:- 

→ Remove ε - Transitions.

→ Another

0^*10^* 

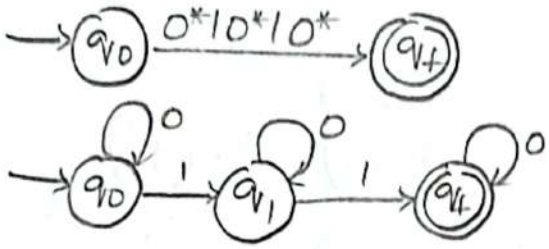
→ Ex-2:-
Construct
Sol:-

⇒

Another

→ Another method:-

$0^*10^*10^*$

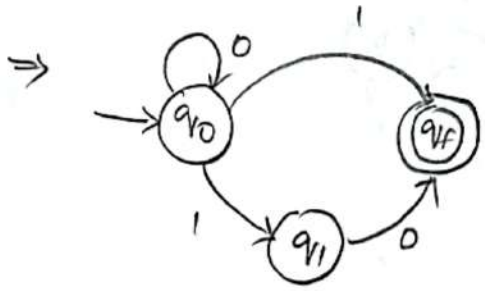


→ Ex-2:-

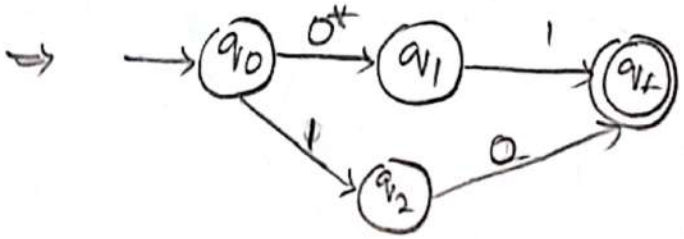
Construct FA Equivalent to given R.E - 0^*1+10.

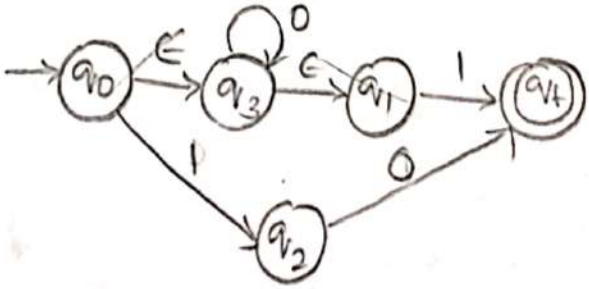
Sol:-

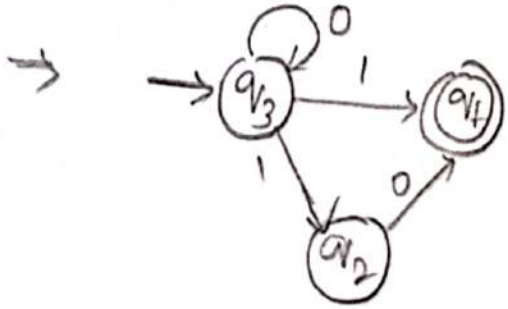
0^*1+10.



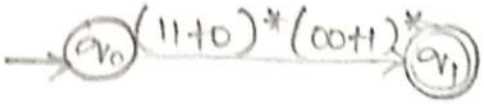
Another Method:-

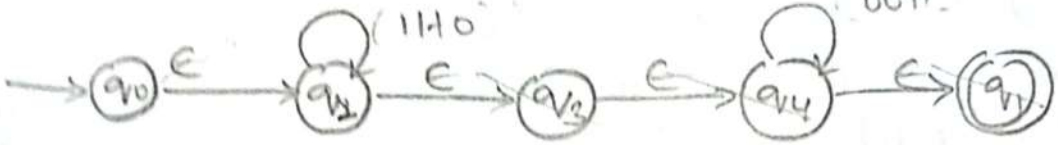


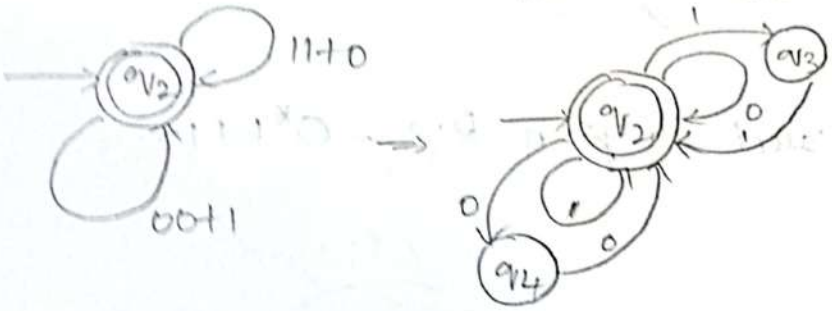




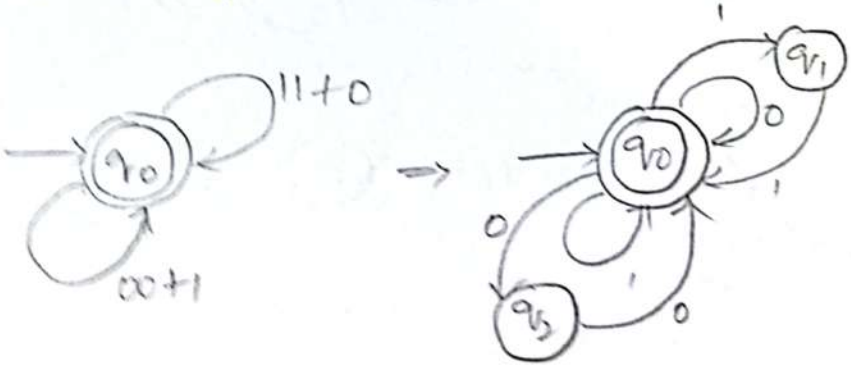
→ (11+10)* (00+1)*







Another Method:-



→ Pumping Lemma for Regular Expression:-

The language accepted by FA are described by R.E's. So, to prove a language is accepted by FA, it is sufficient to prove the R.E of that language is accepted by FA. The languages which are accepted by FA are called Regular languages. Here it means that the FA accepts only the words of this language and does not accept any word outside it.

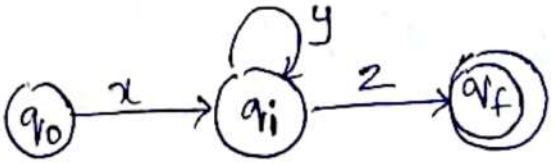
In this section, we will discuss how to prove the certain language is not regular language. Pumping lemma is useful tool to prove that a certain language is not regular language.

→ Pumping lemma is useful because

1. It gives a method for pumping (generating) many substrings from a given string. In other words, its provides a method to break a given i/p string into several substrings.
2. It gives a neccessary conditions to prove a set of strings is not regular.

→ Theorem:-

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a FA having n states, M recognizes the language L. Let $w \in L$ and $|w| \geq M$. If $M \geq n$ then there exists x, y, z such that $w = xyz$, $y \neq \epsilon$ and $xy^iz \in L$ for each $i \geq 0$, where n is no. of states in FA.



Application of pumping Lemma:-

Following steps:-

① Assume that language 'L' is regular. Let 'n' be the no. of states in the corresponding FA.

② Choose a string 'w' such that $|w| \geq n$. Use pumping lemma to write $w = xyz$ with $|xy| \leq n$ & $|y| > 0$

③ Find suitable integer 'i' such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Ex:- Prove that $\{a^n b^n \mid n \geq 0\}$ is not regular.

Sol:-

Language $L = \{\epsilon, ab, aabb, aaabbb, \ldots\}$.

Take $n=6$ and $w = aaabbb$

$\therefore |w| = 6 \geq n$ i.e $6 = 6$ [condition satisfied]

Now break the string $w = aaabbb$ into x, y, z.

$w = \underbrace{aa}_{x}\underbrace{ab}_{y}\underbrace{bb}_{z}$

$\therefore x = aa, y = ab, z = bb$

Now check the conditions $|xy| \leq n$ & $|y| > 0$.

$|xy| = |aaab| = 4 \leq 6$ ✓

$|y| = |ab| = 2 > 0$ ✓

Above 2 conditions satisfied.

Now check the string for xy^iz

For $i=0$, $xy^0z = aa \in bb \Rightarrow aabb \in L$ - Regular

For $i=1$, $xy^1z = aaabbb \Rightarrow aaabbb \in L$ - Regular

For $i=2$, $xy^2z = aa(ab)^2bb \Rightarrow aaababbb \notin L$

→ Not Regular

$\therefore$ Given language is not Regular.

→ Equivalence of FA:-
The two FA are said to be equivalent if both the automata accepts the same set of strings over input set Σ.
→ When two FA are equivalent then there is some string w over Σ on acceptance of that string one finite automata reaches final state and other FA also reaches to the final state.
→ If two FA are not equivalent then there is some string 'w' over Σ on acceptance of that string, one FA reaches final state and other FA reaches to the non-final state.
→ To test the equivalence of two FA we use a method called comparision method.

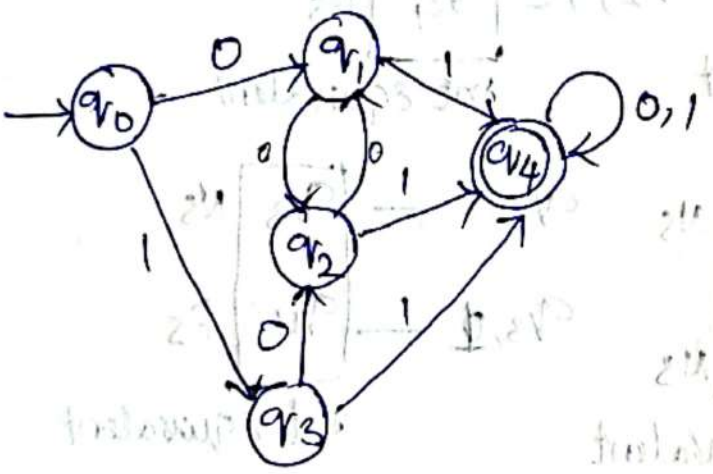
→ Minimization of FA:-

→ Minimization of FA is useful for making compilers execute faster.

→ It removes identical operations by merging 2 (or) more states into a single equivalence state.

→ Minimization of FA involves
- Reducing the no. of states from a given FA.
- Finding which 2 states are equivalent
- Representing those 2 states by one representative state
- Removing unreachable states.
- Joining all states in each class into one state of new automaton.

Ex:-



Sol:-

	0	1
→q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
(q_4)	q_4	q_4

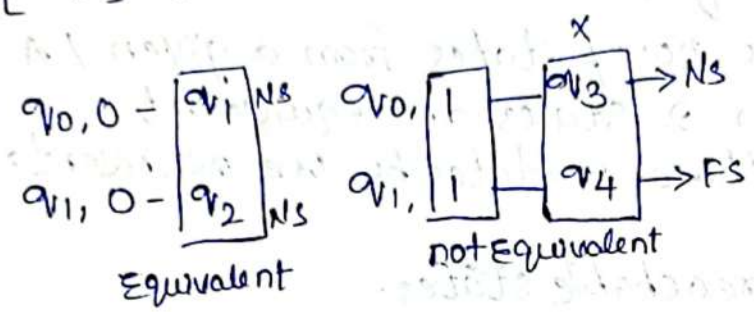
→ 0-Equivalence:-

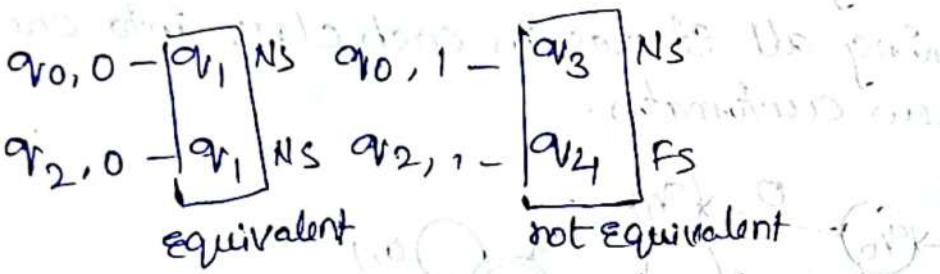
$\{q_0, q_1, q_2, q_3\}$: $\{q_4\}$

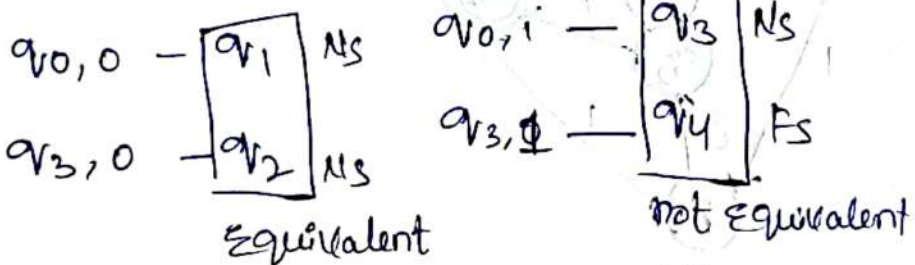
Non-final states Final states

→ 1-Equivalence.

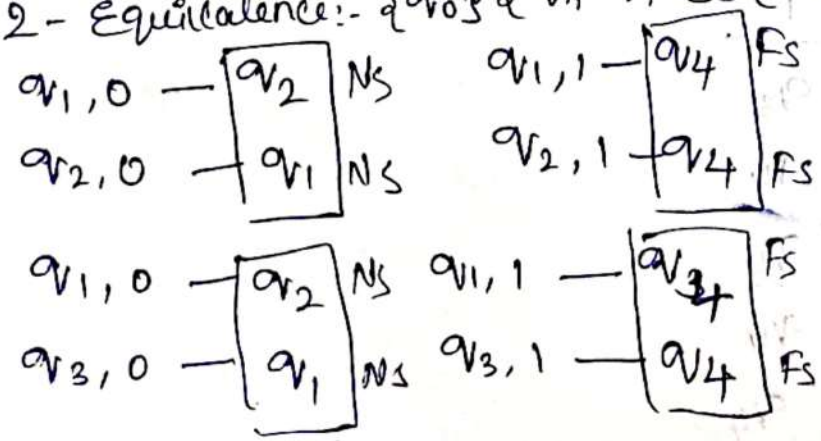
$\{q_0\}$ $\{q_1, q_2, q_3\}$ $\{q_4\}$

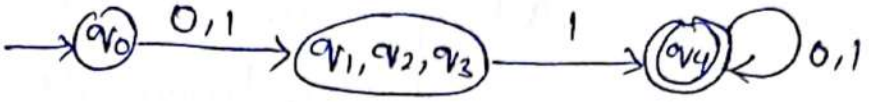




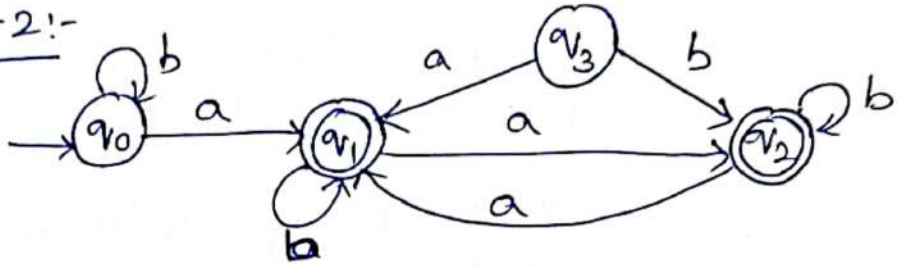


→ 2-Equivalence:- $\{q_0\}\{q_1, q_2, q_3\}\{q_4\}$





→ Ex-2:-



UNIT-III :- CONTEXT FREE GRAMMAR & AUTOMATA

→ Context Free Grammars: Definition of Context free Grammars, Derivations using a Grammar, LeftMost and RightMost Derivations, the language of grammar, Sentential Forms, Parse Trees, Applications of Context free grammars, Ambiguity in grammars and languages.

→ Inorder to study languages we need some rules to be followed. Rules are nothing but grammar.

Grammar (Generator)

↓

Language

↓

Automata (Acceptor).

→ Grammar consists of 4 components

$G = \{V, T, P, S\}$

V → A finite non-empty set of elements called as Variables | Non-Terminals

T → A finite non-empty set of elements called as Terminals | symbols.

P → A special Variable called the start symbol.

P → (production rules) A finite set whose elements are $\alpha \rightarrow \beta$ productions, where α & β are strings on VUT and α has atleast one symbol from V.

→ Elements of P are called as productions or production rules or recuring rules.

→ According to chomsky Heirarchy, grammar is divided into 4 types as follows:

→ 1. Type 0: Unrestricted grammar / generates Recursive enumerable language / accepted by Turing machine.

→ Grammar production is in the form of $\alpha \rightarrow \beta$

where $\alpha \in (VUT)^* V (VUT)^*$

$\beta \in (VUT)^*$

Rule:- α Should have atleast one Variable.
β has no restrictions.

Ex:- $A \longrightarrow \in$, $AaB \longrightarrow Ab$.

Variables $\longrightarrow$ A, B

Terminals $\longrightarrow \in, a, b$.

→ Type 1:- Context Sensitive Grammar / Generates context sensitive language / accepted by Linear Bound Automata. (Some restrictions)

→ Grammar production is in the form of $\alpha \longrightarrow \beta$

where $\alpha \in (VUT)^+$ } $\alpha, \beta \neq \in$

$\beta \in (VUT)^*$

Rule:- $|\alpha| \leq |\beta|$

Ex:- $Aa \Leftrightarrow BaA$, $Aa \Leftrightarrow Bb$

$2 \leq 3$ $\quad 2 = 2$.

→ Type 2:- Context Free Grammar / Generates Context Free Language / accepted by Push Down Automata.

→ Grammar production is in the form of $\alpha \longrightarrow \beta$.

where $\alpha \in V$

$\beta \in (VUT)^*$

Rule:- $|\alpha| = 1$, No Restriction for β.

Ex:- $A \longrightarrow \in$, $B \longrightarrow Ab$, $B \longrightarrow b$

→ Type 3:- Regular Grammar / Generates Regular language / accepted by Finite Automaton. It is the most restricted form of Grammar.

→ Grammar production is in the form of $\alpha \longrightarrow \beta$ where $\alpha \in V$ and $\beta \in TV$ or VT

Type 3 grammar is given in the form of

→ Left-Linear Grammar $V \rightarrow VT \mid T$

→ Right-Linear Grammar $V \rightarrow TV \mid T$

Rule:- $|\alpha| = 1$, $|\beta| = 2$

Ex:- $A \rightarrow Ba$ (LRG)

$A \rightarrow aB$ (RRG).

→ Context:-

The context of a word, sentence or text consists of the words, sentences or text before and after it which help to make its meaning clear.

→ Context free grammars:-

The "context free" means the set of production rules is defined in such a way that the replacement of non-Terminals symbols in anyone of the production rules does not depend upon the presence of terminal symbols before or after it.

→ A Grammar $G = (V, T, P, S)$ is said to be CFG if its productions are of the form:- $\alpha \rightarrow \beta$ where $\alpha \in V$ and $\beta \in (V \cup T)^*$

→ The Right hand side (β) of a CFG is not restricted and it may be null or combination of Variables & Terminals. The possible length of β ranges from 0 to ∞ i.e. $0 \leq |\beta| \leq \infty$

Example:-①:-

consider the grammar $G = (V, T, P, S)$ having productions: $S \rightarrow aSa \mid bSb \mid \epsilon$. Check the production and find the language generated

Sol:-

Let P_1: $S \rightarrow aSa$ (RHS is Terminal Variable Terminal)

P_2: $S \rightarrow bSa$ (RHS is " " ")

P_3: $S \rightarrow \in$ (RHS is null string)

Since all the productions are of the form $V \rightarrow (VUT)^*$

hence the grammar is CFG.

→ Language Generated:-

$S \rightarrow aSa$ or bSb

$S \rightarrow a^nSa^n$ or b^mSb^m (using m or n step derivation).

$S \rightarrow a^nb^mSb^ma^n$ (or) $b^ma^nSa^nb^m$ (substituting 'S' value).

$S \rightarrow a^nb^mb^ma^n$ (or) $b^ma^na^nb^m$ (substituting $S \rightarrow \in$).

So, $L(G) = \{ww^R : w \in (a+b)^*\}$.

→ Example-2:-

Let $G = \{V, T, P, S\}$ where $V = \{S, C\}$, $T = \{a, b\}$ having productions $S \rightarrow aca$, $C \rightarrow aca|b$, 'S' is the Start Symbol. } Find the language generated.

Sol:-

Let p_1: $S \rightarrow aca$

P_2: $C \rightarrow aca$

P_3: $C \rightarrow b$.

→ Language generated:-

$S \rightarrow aca$

$S \rightarrow aacaa$ (By applying $C \rightarrow aca$)

$S \rightarrow aaacaaa$ (By applying $C \rightarrow aca$)

⋮

$S \rightarrow a^nca^n$ (By applying 'c' n-times)

$S \rightarrow a^nba^n$ (By applying $C \rightarrow b$)

∴ The language generated $L(G) = \{a^n b a^n \mid n \geq 1\}$

→ Construct CFG for given language.

Ex-1:- $L = \{ww^R \mid w \in 0,1\}$

Sol:-

w - given string

w^R - Reverse string

$L = \{00, 11, 0000, 1111, 1001, 0110, \cdots\}$

$S \rightarrow 00 \mid 11 \mid 0S0 \mid 1S1$

$G = (V, T, P, S)$

$V = \{S\}$

$T = \{0,1\}$

$S = \{S\}$

P: $\{S \rightarrow 00 \mid 11 \mid 0S0 \mid 1S1\}$

Generate 10110l

$S \rightarrow 1S1$

$S \rightarrow 10S01$

$S \rightarrow 10110l$

Generate 11001l

$S \rightarrow 1S1$

$S \rightarrow 11S11$

$S \rightarrow 11001l$

Ex-2:- $L = \{w \mid w \in$ equal no of a's & equal no of b's$\}$

Sol:- $L = \{\epsilon, ab, aabb, abab, baba, aaabbb, \cdots\}$

P: $S \rightarrow \epsilon \mid asbs \mid bsas$, $S \rightarrow as \mid bs \mid a \mid a$

Ex-3:- $L = \{a^n b^n \mid n \geq 1\}$

sol:- P: $S \rightarrow ab \mid asb$

Ex-4:-

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

sol:- P: $S \rightarrow \epsilon \mid ab \mid asb$

Ex-5:- $L = \{$ one 'a' & any no of b's$\}$

$L = \{ab^n \mid n \geq 0\}$.

Sol:- $L = \{a, ab, abb, abbb, \cdots\}$

P: $S \rightarrow aA$

$A \rightarrow bA \mid \epsilon$

Ex-6:- $L = \{a^n b^n c^m d^m \mid m, n \geq 0\}$

Sol:- $L = \{\epsilon, ab, cd, aabb, abcd, ccdd, \cdots\}$

(ab: $m=0, n=0$; $aabb$: $n=2, m=0$; $abcd$: $n=1, m=1$; $ccdd$: $n=0, m=2$)

P: $S \rightarrow AB$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow CBd \mid \epsilon$

Ex-7:- $L = \{a^n b^m c^{2m} \mid m, n \geq 0\}$

Sol: $L = \{\epsilon, a, bcc, abcc, \cdots\}$

P: $S \rightarrow AB$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

$A \rightarrow aA$

$B \rightarrow bBcc$.

→ Derivations using a grammar:-

→ We apply productions of a CFG to infer that certain strings are in the language of a certain variable. There are two approaches to this inference. The more conventional approach is to use the rules from body to head. That is, we take strings known to be in the language of each of the variables of the body, concatenate them, in the proper order, with any terminals appearing in the body and infer that the resulting string is in the language of the variable in the head. We shall refer to this procedure as recursive inference.

→ Each production consists of
a) A variable is being defined by production. This variable is called the head of the production.
b) The production symbol →
c) A string of zero or more terminals & variables. This string is called body of the production.

P: S → aA
(S: Head, aA: Body)

→ There is another approach to defining the language of grammar, in which we use the productions from head to body. We expand the start symbol using one of its productions. (i.e using a production whose start symbol is head). We further expand the resulting string by replacing one of the variables by the body of one of its productions & so on, until we derive a string consisting entirely of terminals. The language of the grammar is all strings of terminals that we can obtain in this way. This use of grammars is called Derivation.

→ Leftmost and Rightmost Derivations:-

Leftmost derivation:-

→ If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$ then a derivation $S \underset{L}{\overset{*}{\Rightarrow}} w$ is called leftmost derivation if and only if all steps involved in derivation have leftmost replacement only.

→ Rightmost derivation:-

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$ then a derivation $S \underset{R}{\overset{*}{\Rightarrow}} w$ is called rightmost derivation if and only if all steps involved in derivation have rightmost replacement only.

→ Example 1: Consider the grammar $S \rightarrow S+S \mid S*S \mid a \mid b$ Find leftmost and rightmost derivations for string $w = a*a+b$.

Variables – S
Terminals – +, *, a, b.

Sol:- $S \rightarrow S+S$, $S \rightarrow S*S$, $S \rightarrow a$, $S \rightarrow b$

Leftmost derivation for $w = a*a+b$

$S \underset{L}{\Rightarrow} S*S$ (Using $S \rightarrow S*S$)

$\underset{L}{\Rightarrow} a*S$ (Leftmost Variable S, apply $S \rightarrow a$)

$\underset{L}{\Rightarrow} a*S+S$ (Leftmost Variable S, apply $S \rightarrow S+S$)

$\underset{L}{\Rightarrow} a*a+S$ (Leftmost Variable S, apply $S \rightarrow a$)

$\underset{L}{\Rightarrow} a*a+b$ (Leftmost Variable S, apply $S \rightarrow b$)

Rightmost derivation for $w = a*a+b$

$S \underset{R}{\Rightarrow} S*S$ (Using $S \rightarrow S*S$)

$\underset{R}{\Rightarrow} S*S+S$ (Rightmost Variable S, apply $S \rightarrow S+S$)

$\underset{R}{\Rightarrow} S*S+b$ (Rightmost Variable S, apply $S \rightarrow b$)

$\underset{R}{\Rightarrow} S*a+b$ (Rightmost Variable S, apply $S \rightarrow a$)

$\underset{R}{\Rightarrow} a*a+b$ (Rightmost Variables S, apply $S \rightarrow a$)

→ Ex-2:- Consider a CFG, $S \to bA|aB$, $A \to as|aAA|a$, $B \to bs|aBB|b$. Find the leftmost and rightmost derivations for $w = aaabbabbba$.

Sol:-

Leftmost derivation for $w = aaabbabbba$

$S \Rightarrow aB$ (Using $S \to aB$)

$\Rightarrow aaBB$ (Using $B \to aBB$)

$\Rightarrow aaaBBBB$ (Using $B \to aBB$)

$\Rightarrow aaabBB$ (Using $B \to b$)

$\Rightarrow aaabbB$ (Using $B \to b$)

$\Rightarrow aaabbaBB$ (Using $B \to aBB$)

$\Rightarrow aaabbabB$ (Using $B \to b$)

$\Rightarrow aaabbabbs$ (Using $B \to bs$)

$\Rightarrow aaabbabbbA$ (Using $B \to bA$)

$\Rightarrow aaabbabbba$ (Using $A \to a$)

Rightmost Derivation:-

$S \Rightarrow aB$ (Using $S \to aB$)

$\Rightarrow aaBB$ (Using $B \to aBB$)

$\Rightarrow aaBbs$ (Using $B \to bs$)

$\Rightarrow aaBbbA$ (Using $S \to bA$)

$\Rightarrow aaBbba$ (Using $A \to a$)

$\Rightarrow aaaBBbba$ (Using $B \to aBB$)

$\Rightarrow aaabBbba$ (Using $B \to b$)

$\Rightarrow aaabsbbba$ (Using $B \to bs$)

$\Rightarrow aaabbAbbba$ (Using $S \to bA$)

$\Rightarrow aaabbabbba$ (Using $A \to a$).

→ Derivation Tree :-

Derivation Tree is a graphical representation for the derivation of given production rules for a given CFG. It is a simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation is also known as parse tree.

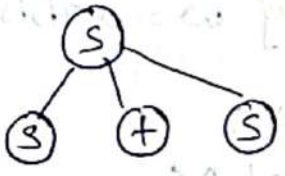
→ Let $G_1 = (V, T, P, S)$ is a CFG. Each production of G_1 is represented with a tree satisfying following conditions:

1. If $A \rightarrow \alpha_1 \alpha_2 \alpha_3 \cdots \alpha_n$ is a production in G_1 then A becomes the parent node labelled $\alpha_1, \alpha_2, \alpha_3, \cdots \alpha_n$ and
2. The collection of children from left to right yields $\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n \rightarrow \alpha_1 \alpha_2 \alpha_3 \cdots \alpha_n$

Ex:- Consider a CFG $S \rightarrow S+S \mid S*S \mid a \mid b$ and Construct derivation trees for all productions.

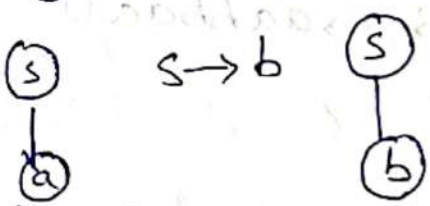
Sol:-

For the production $S \rightarrow S+S$



For the production $S \rightarrow S*S$

For the production $S \rightarrow a$ $S \rightarrow b$



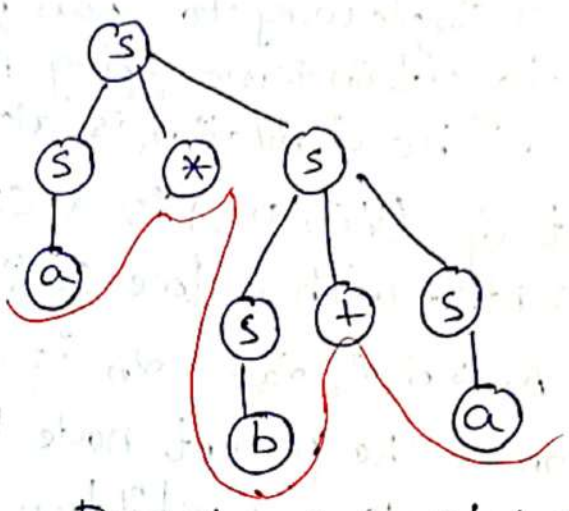
→ The parse tree, contains the following properties for $w \in L(G_1)$:-

1. The root node is labelled as S (the starting symbol).
2. The all internal vertices (or nodes) are labelled with variables.
3. The leaves or terminal nodes are labeled with ϵ or terminal symbols.
4. If $A \rightarrow \alpha_1 \alpha_2 \alpha_3 \cdots \alpha_n$ is a production in G_1 then 'A' becomes the parent node labelled $\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n$.
5. The collection of leaves from left to right yields string 'w'.

→ Example 1:- Consider the grammar $S \rightarrow S+S \mid S*S \mid a \mid b$ construct the derivation tree for string $w = a*b+a$.

Sol:- Leftmost Derivation for $w = a*b+a$

$$S \underset{L}{\Rightarrow} S*S$$
$$\underset{L}{\Rightarrow} a*S$$
$$\underset{L}{\Rightarrow} a*S+S$$
$$\underset{L}{\Rightarrow} a*b+S$$
$$\underset{L}{\Rightarrow} a*b+a$$

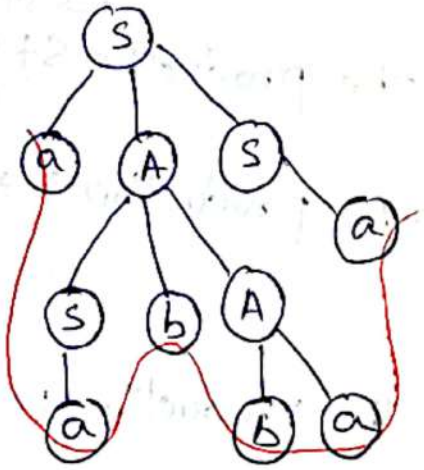


Parse tree for $a*b+a$

→ Ex-2:- Consider the grammar having productions $S \rightarrow aAS \mid a$, $A \rightarrow SbA \mid SS \mid ba$. Construct derivation tree for the string $w = aabbaa$.

Sol:-

$S \rightarrow aAS$
$S \rightarrow aSbAS$
$S \rightarrow aabAS$
$S \rightarrow aabbas$
$S \rightarrow aabbaa$



Parse tree for aabbaa.

→ Ex-3:- $S \rightarrow 0B \mid 1A$, $A \rightarrow 0 \mid 0S \mid 1AA$, $B \rightarrow 1 \mid 1S \mid 0BB$.

String $w = 00110101$.

→Ex-3:- $S \rightarrow 0B \mid 1A$, $A \rightarrow 0 \mid 0S \mid 1AA$, $B \rightarrow 1 \mid 1S \mid 0BB$

String $w = 00110101$

Sol:-

$S \rightarrow 0B$

$\Rightarrow 00BB$

$\Rightarrow 0011S$

$\Rightarrow 00110B$

$\Rightarrow 001101S$

$\Rightarrow 00110101$

→ Ambiguity in context-free Grammars:-

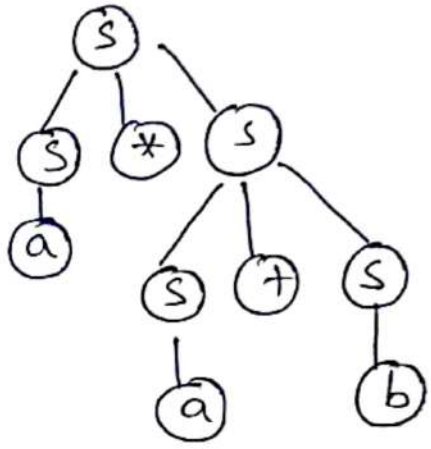
A grammar G is ambiguous if there exists some string $w \in L(G)$ for which there are 2 or more distinct derivation trees or there are 2 or more distinct leftmost derivations.

Ex:- Consider the CFG $S \rightarrow S+S \mid S*S \mid a \mid b$ and string $w = a*a+b$ and derivations as follows:

Sol:-

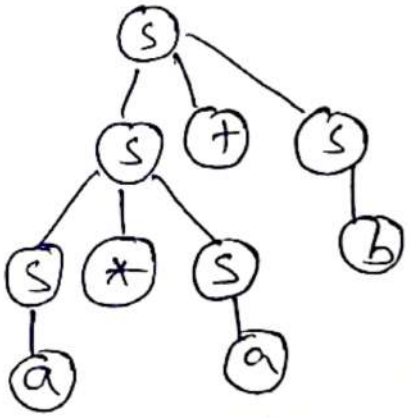
First Leftmost derivation for $w = a*a+b$

$S \Rightarrow \underline{S}*S$ (Apply $S \rightarrow S*S$)
$\Rightarrow a*\underline{S}$ ($S \rightarrow a$)
$\Rightarrow a*\underline{S}+S$ ($S \rightarrow S+S$)
$\Rightarrow a*a+\underline{S}$ ($S \rightarrow a$)
$\Rightarrow a*a+b$ ($S \rightarrow b$)



Second Leftmost derivation for $w = a*a+b$

$S \Rightarrow \underline{S}+S$ (Apply $S \rightarrow S+S$)
$\Rightarrow \underline{S}*S+S$ (Apply $S \rightarrow S*S$)
$\Rightarrow a*S+S$ ($S \rightarrow a$)
$\Rightarrow a*a+S$ ($S \rightarrow a$)
$\Rightarrow a*a+b$ ($S \rightarrow b$)



Since, there are two distinct, leftmost derivations (two parse trees) for string w, hence w is ambiguous and there is ambiguity in grammar G.

→ Ex-2:-

If G is the grammar, $S \rightarrow SbS \mid a, S \mid T$ G is ambiguous for the string $w = abababa$.

Sol:-

First Leftmost Derivation for string $w = abababa$

$S \Rightarrow \underline{S}bS$ (Apply $S \to SbS$)

$\Rightarrow ab\underline{S}$ (Apply $S \to a$)

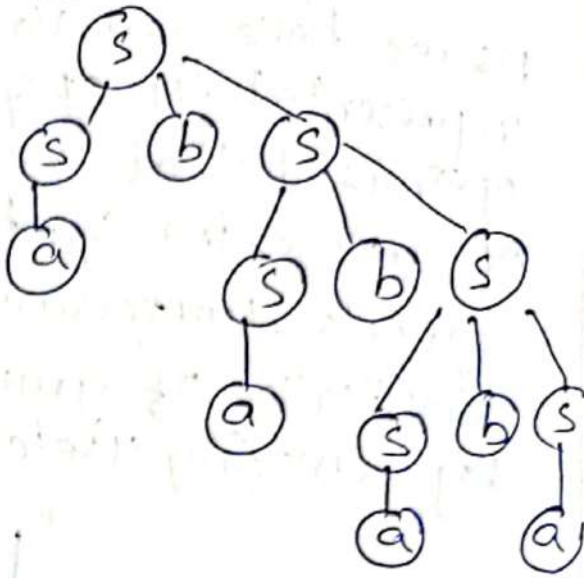
$\Rightarrow abSbS$ ($S \to SbS$)

$\Rightarrow ababS$ ($S \to a$)

$\Rightarrow ababSbS$ ($S \to SbS$)

$\Rightarrow abababS$ ($S \to a$)

$\Rightarrow abababa$ ($S \to a$).



Second Leftmost Derivation for string $w = abababa$

$S \to \underline{S}bS$ (Apply $S \to SbS$)

$\Rightarrow \underline{S}bSbS$ ($S \to SbS$)

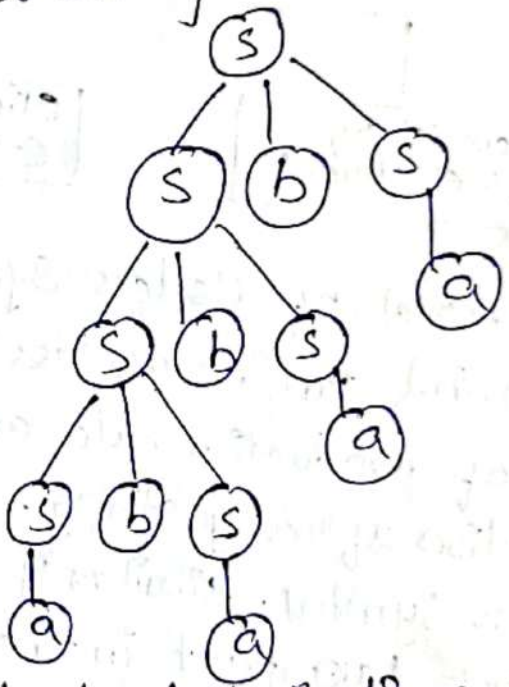
$\Rightarrow \underline{S}bSbSbS$ ($S \to SbS$)

$\Rightarrow ab\underline{S}bSbS$ ($S \to a$)

$\Rightarrow ababSbS$ ($S \to a$)

$\Rightarrow abababS$ ($S \to a$)

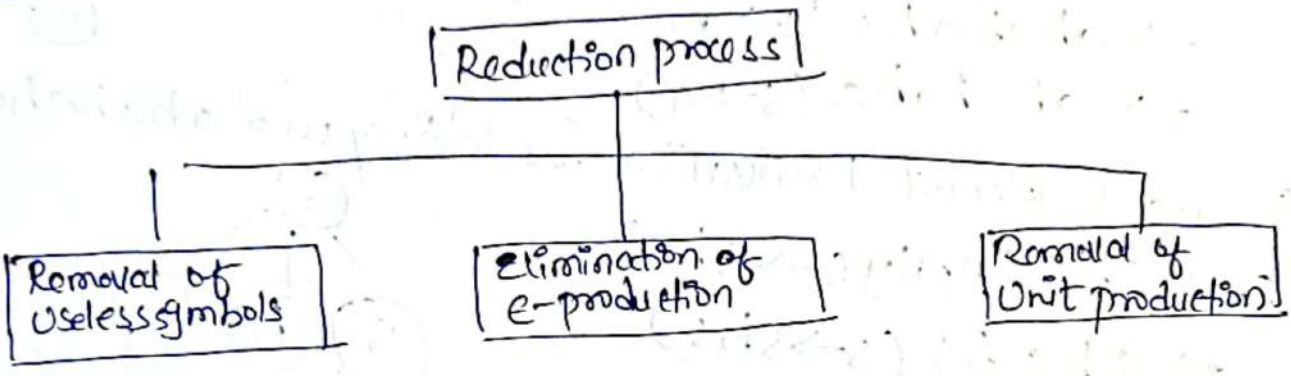
$\Rightarrow abababa$ ($S \to a$)



Since, there are 2 distinct, leftmost derivations.

Hence proved given grammar is ambiguous for the string $w = abababa$.

→ Simplication or Minimization of CFG:-
As we have seen, various languages can be efficiently represented by CFG. All the grammar are not always optimized that means the grammar may consist of some extra symbols (non-terminals). Having extra symbols, unnecessarily increase the length of grammar. Simplication of grammar means reduction of grammar by removing useless symbols.



→ Removal of Useless Symbols:-
A symbol can be useless if it does not appear on the RHS of production rule and does not take part in the derivation of any string. That symbol is known as a Useless Symbol. Similarly, a variable can be useless if it does not take part in the derivation of any string. That Variable is known as a useless variable.

Ex:- $T \rightarrow aaB \mid abA \mid aaT$
$A \rightarrow aA$
$B \rightarrow ab \mid b$
$C \rightarrow abad$

→ In the above example, the variable 'C' will never occur in the derivation of any string. So, the production $C \rightarrow ad$ is useless.
→ Production $A \rightarrow aA$ is also useless because there is no way to terminate it. If it never terminates then it can never produce a string.

∴ production rules after eliminating useless symbols are

$T \rightarrow aaB \mid aaT$

$B \rightarrow ab \mid b$.

→ Elimination of ∈-production:-

The productions of type $S \rightarrow \epsilon$ are called ∈-productions. These type of productions can only be removed from those grammars that do not generate ∈.

Ex:- Remove the production from the following CFG by preserving the meaning of it.

$S \rightarrow XYX$

$X \rightarrow 0X \mid \epsilon$

$Y \rightarrow 1Y \mid \epsilon$

Sol:- Now, while removing ∈-production, we are deleting rule $X \rightarrow \epsilon$ & $Y \rightarrow \epsilon$. To preserve the meaning of CFG we are actually placing ∈ at the right-hand side whenever X and Y have appeared.

Let us take $S \rightarrow XYX$

→ If the first X at RHS is ∈ then

$S \rightarrow YX$ → If second X at RHS is ∈ then

$S \rightarrow XY$.

→ If $Y \rightarrow \epsilon$ then.

$S \rightarrow XX$

→ If X & Y are ∈ then

$S \rightarrow X$

→ If both X are replaced by ∈ then

$S \rightarrow Y$

∴ production rules for

$S \rightarrow XYX \mid YX \mid XY \mid XX \mid X \mid Y$

→ Now Let us consider $X \rightarrow 0X$

If we place ϵ at R.H.S for X then

$X \rightarrow 0$

∴ production rules for X are

$X \rightarrow 0X \mid 0$

→ Similarly $Y \rightarrow 1Y \mid 1$.

∴ We can rewrite CFG as

$S \rightarrow XYX \mid YX \mid XY \mid XX \mid X \mid Y$

$X \rightarrow 0X \mid 0$

$Y \rightarrow 1Y \mid 1$

→ Removing Unit productions:-

The unit productions are the productions in which one non-terminal gives another non-terminal.

Ex:-

$S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

$S \rightarrow C$ is a unit production. But while removing $S \rightarrow C$. we have to consider what C gives. So, we can add a rule to S.

∴ $S \rightarrow 0A \mid 1B \mid 01$.

→ Similarly $B \rightarrow A$ is also a unit production, so we can modify as

$B \rightarrow 1 \mid 0S \mid 00$.

∴ Final CFG is

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid 0S \mid 00$

$C \rightarrow 01$

→ Normal forms for CFG:-
In a CFG, the RHS of a production can be any string of variables & terminals. When the productions in G satisfy certain restrictions, then G is said to be in a "normal form". Among several normal forms, we study two of them - CNF (Chomsky Normal form) & the Greibach Normal form (GNF).

→ Chomsky Normal form (CNF):-
In chomsky Normal form (CNF), we have restrictions on the length of R.H.S and the nature of symbols in the R.H.S of productions.

→ Definition:-
A CFG 'G' is in CNF if every production is of the form
$A \rightarrow a$, $A \rightarrow BC$ and $S \rightarrow \epsilon$.

→ The start symbol can have ϵ-production.

$S \rightarrow \epsilon$

→ The Non-terminal generating single Terminal.

$A \rightarrow a$
$A \rightarrow abx$

→ The Non-Terminal generating two Non-terminals

$A \rightarrow BC$
$A \rightarrow BCDx$
$A \rightarrow aBx$

→ Reduction to chomsky Normal form:-
We develop a method of constructing a grammar in CNF equivalent to a given CFG.
- Let us consider an example. Let 'G' be $S \rightarrow ABC \mid aC$, $A \rightarrow a$, $B \rightarrow b$, $C \rightarrow c$. Except $S \rightarrow ABC \mid aC$, all the productions are in the form required for CNF.
- The terminal 'a' in $S \rightarrow aC$ can be replaced by a new variable 'D'. By adding a new variable production $D \rightarrow a$, the effect of applying $S \rightarrow aC$ can be achieved by $S \rightarrow DC$ and $D \rightarrow a$.
$S \rightarrow ABC$ is not in the required form and hence this

Production can be replaced by $S \rightarrow AE$ & $E \rightarrow BC$. Thus an equivalent grammar is $S \rightarrow AE \mid DC$, $E \rightarrow BC$, $A \rightarrow a$, $B \rightarrow b$, $C \rightarrow c$, $D \rightarrow a$.

→ Steps to be followed in conversion of CFG to CNF:-

-Step-1:- Remove start symbol in R.H.S with another Non-Terminal and create new production.

$S \rightarrow ASB$

$S_1 \rightarrow S$
$S \rightarrow ASB$

→ Start symbol - S_1

-Step-2: Remove all null productions, Unit productions and useless symbols (Simplification of CFG).

-Step-3: Replace terminals on RHS with non-Terminals and create new production.

$A \rightarrow aB \Rightarrow$ $A \rightarrow XB$, $X \rightarrow a$

-Step-4: Reduce the number of Non-Terminals on RHS by creating new production.

$S \rightarrow ABC \Rightarrow$ $S \rightarrow XC$, $X \rightarrow AB$.

Ex-1:- Convert CFG to CNF.

$S \rightarrow ASB \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \epsilon$

Sol:-

Step-1:-

$S \rightarrow ASB \Rightarrow$ $S_1 \rightarrow S$, $S \rightarrow ASB \mid aB$

Step-2:-

Eliminate ϵ-productions

$B \rightarrow \epsilon \Rightarrow$ $S \rightarrow ASB \mid aB \mid As \mid a$, $A \rightarrow B \mid S \mid \epsilon$, $B \rightarrow b$

$A \to \epsilon$: eliminate $A \to \epsilon$:

$S \to ASB \mid aB \mid As \mid a \mid SB \mid S$

$A \to B \mid S$

$B \to b$

→ Eliminate Unit productions

$S \to S$
$S_1 \to S$

$S_1 \to S$
$S \to ASB \mid aB \mid As \mid a \mid SB$
$A \to B \mid S$
$B \to b$

$S_1 \to S$

$S_1 \to ASB \mid aB \mid As \mid a \mid SB$
$S \to ASB \mid aB \mid As \mid a \mid SB$
$A \to B \mid S$
$B \to b$

$A \to B$

$S_1 \to ASB \mid aB \mid As \mid a \mid SB$
$S \to ASB \mid aB \mid As \mid a \mid SB$
$A \to b \mid S$
$B \to b$

$A \to S$

$S_1 \to ASB \mid aB \mid As \mid a \mid AB$
$S \to ASB \mid aB \mid As \mid a \mid SB$
$A \to b \mid ASB \mid aB \mid As \mid a \mid SB$
$B \to b$

→ Step-3:-
Replace Terminal with Non-Terminal

$S_1 \to ASB \mid XB \mid As \mid a \mid SB$
$S \to ASB \mid XB \mid As \mid a \mid SB$
$A \to b \mid ASB \mid XB \mid As \mid a \mid SB$
$B \to b$
$X \to a$

Step-4:- Reduce no of Non-Terminals.

$S_1 \to YB \mid XB \mid As \mid a \mid SB$
$S \to YB \mid XB \mid As \mid a \mid SB$
$A \to b \mid YB \mid XB \mid As \mid a \mid SB$
$B \to b$
$X \to a$
$Y \to As$

→ Ex-2:-
Reduce the following grammar G to CNF.

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

→ Ex-3:-
$S \rightarrow aAbB$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b.$

→ Elimination Left Recursion:-
A grammar $G = (V, T, P, s)$ is left recursive if it has production in the form $A \rightarrow A\alpha \mid \beta$.
The above can eliminate left recursion by replacing a pair of production with

$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \epsilon$

Ex:- Eliminate Left Recursion from the grammar
$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid (id)$

Sol:- comparing $E \rightarrow E+T \mid T$ with $A \rightarrow A\alpha \mid \beta$

E	→	E	+T	|	T
A	→	A	α	|	β

∴ $A = E, \alpha = +T, \beta = T$
∴ $A \rightarrow A\alpha/\beta$ is changed to $A \rightarrow \beta A'$ and $A' \rightarrow \alpha A' \mid \epsilon$

∴ $A \rightarrow \beta A'$ means $E \rightarrow TE'$, $A' \rightarrow \alpha A' \mid \epsilon$ means $E' \rightarrow +TE' \mid \epsilon$

Comparing $T \rightarrow T*F \mid F$ with $A \rightarrow A\alpha \mid \beta$.

T	→	T	*F	|	F
A	→	A	α	|	β

∴ $A = T, \alpha = *F, \beta = F$
∴ $A \rightarrow \beta A'$ means $T \rightarrow FT'$
$A \rightarrow \alpha A' \mid \epsilon$ means $T' \rightarrow *FT' \mid \epsilon$

production $F \rightarrow (E) \mid id$ does not have left recursion.
∴ production rules are

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow (E) \mid id.$

→ Greibach Normal Form:-
A Context Free Grammar is in Greibach Normal Form if all the production rules satisfy the following conditions:-

- A start symbol generating ε.

  $S \rightarrow \epsilon$

- A non-terminal generating a terminal.

  $A \rightarrow a$

- A non-terminal generating a terminal which is followed by any no. of non-terminals.

  $S \rightarrow aASB$.

→ Steps for converting CFG into GNF:-

- step-1: Convert the grammar into CNF, if the given grammar is in CFG.
- step-2: [Replace all Variables of the CFG as $A_1, A_2, \ldots A_n$. $A_1 \rightarrow$ Start Symbol] [Rename Non-Terminals with Numeric variables in ascending order in which they appeared.]
- step-3: $A_i \rightarrow A_j\beta$
  (i) $i<j$
  (ii) $i>j$
  iii) $i=j$

case (i) :- If $i<j$ for the production rule $A_i \rightarrow A_j\beta$ leave the production as it is.

case (ii):- If $i>j$ for the production rule $A_i \rightarrow A_j\beta$ Use substitution method.

case (iii):- If $i=j$ for the production rule $A_i \rightarrow A_j\beta$ Remove Left Recursion.

step-4:- Make the productions following GNF rules.

→ Examples for Left Recursion:-

① $S \rightarrow SOS1S \mid 01$

Sol:

$S \rightarrow \underset{A}{S}\underset{A}{O}\underset{\alpha}{S1S} \mid \underset{\beta}{01}$

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

$\therefore S \rightarrow 01S'$

$S' \rightarrow OS1SS' \mid \epsilon$

② $S \rightarrow (L) \mid x$

$L \rightarrow L,S \mid S$

Sol:-

$S \rightarrow (L) \mid x$ - no left recursion

$L \rightarrow \underset{A}{L},\underset{\alpha}{S} \mid \underset{\beta}{S}$

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

$\therefore L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \epsilon$

→ Examples for Greibach Normal form:-

① $S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$

Sol:-

Step-1:- The given grammar is in chomsky NF.

Step-2:- Replace $\{S \rightarrow A_1, C \rightarrow A_2, A \rightarrow A_3, B \rightarrow A_4\}$

$\therefore$ production rules are

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step-3:-

Case(i):- $i<j$, for $A_1 \rightarrow A_2A_3 \mid A_4A_4$

$1<2,\ 1<4$

Leave the production as it is.

case(ii) $i>j$, for $A_4 \rightarrow b/A_1A_4$

$4 > 1$

substitute A_1 in $A_4 \Rightarrow A_1 \rightarrow A_2A_3/A_4A_4$

$\therefore A_4 \rightarrow b/A_2A_3A_4/A_4A_4A_4 \longrightarrow (1)$.

In (1) eqn, $i>j$ i.e $4>2$.

Substitute A_2 in A_4.

$A_4 \rightarrow b/bA_3A_4/A_4A_4A_4 \longrightarrow (2)$.

In eqn (2), $i=j$ i.e $A_4 = A_4$.

So, elimination left recursion.

$A \rightarrow A\alpha/\beta$
$A \rightarrow \beta A'$
$A' \rightarrow \alpha A'/\epsilon$

$A_4 \rightarrow bZ/bA_3A_4Z \longrightarrow (3)$

$Z \rightarrow A_4A_4Z/\epsilon \longrightarrow (4)$

From eqn (4), eliminate Null production.

$A_4 \rightarrow bZ/bA_3A_4Z/b/bA_3A_4 \longrightarrow (5)$

$Z \rightarrow A_4A_4Z/A_4A_4 \longrightarrow (6)$.

eqn (5) in GNF but eqn (6) is not in GNF.

Substitute A_4 in Z

$Z \rightarrow bZA_4Z/bA_3A_4ZA_4Z/bA_4Z/bA_3A_4A_4Z/$
$bZA_4/bA_3A_4ZA_4/bA_4/bA_3A_4A_4 \longrightarrow (7)$.

$\therefore$ eqn (4) & eqn (7) are in GNF form.

→ $\therefore$ Production rules after converting from CFG to GNF.

$A_1 \rightarrow A_2A_3/A_4A_4$, substitute A_2 & A_4.

$A_1 \rightarrow bA_3/bZA_4/bA_3A_4ZA_4/bA_4/bA_3A_4A_4$

$A_4 \rightarrow bZ/bA_3A_4Z/b/bA_3A_4$

$Z \rightarrow bZA_4Z/bA_3A_4ZA_4Z/bA_4Z/bA_3A_4A_4Z/bZA_4/$
$bA_3A_4ZA_4/bA_4/bA_3A_4A_4$

$A_2 \rightarrow b$, $A_3 \rightarrow a$.

Ex-2 :- Convert the grammar to GNF

(2) $S \rightarrow AB$
$A \rightarrow BS \mid b$
$B \rightarrow SA \mid a$

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

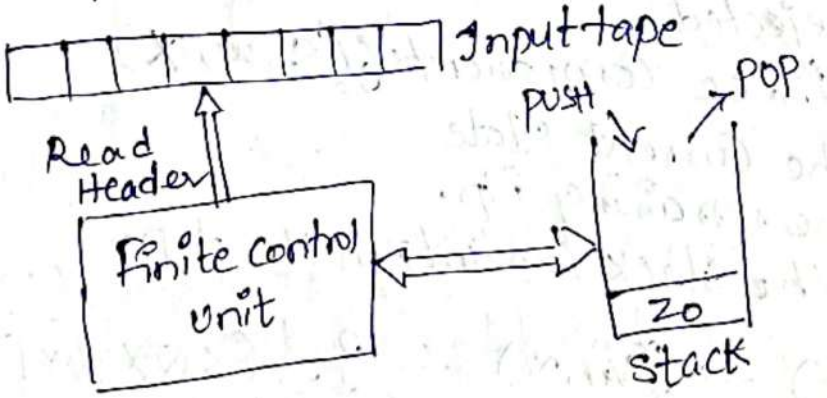
→ Push Daon Automata:-

PDA is a way to implement a context free Grammar in the same way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but PDA can remember an infinite amount of information.

→ PDA is simply an NFA augmented with an "External Stack Memory". The addition of stack is used to provide a last-in-first-out memory management capability to PDA.

A PDA can push an element onto the top of the stack and pop off an element from the top of the stack. To read an element into the stack, the top elements must be popped off and are lost.

→ PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA. PDA also accepts a class of language which even cannot be accepted by FA. Thus PDA is much more superior than FA.



→ PDA Components:-

(1) Input Tape:-
The i/p tape is divided into many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.

(2) Finite Control:-
The finite control has some pointer which points the current symbol which is to be read.

→ Stack:-
The stack is a structure in which we can push and remove the items from one end only. It has an infinite size. In PDA, stack is used to store the items temporarily.

→ PDA can be defined as collection of 7 components:

PDA = { Q, Σ, q0, F, Z0, Γ, δ } → FA = {Q, Σ, δ, q0, F}

Q = Finite set of states
Σ = input alphabet
q0 = Initial state
F = set of final states
Z0 = Bottom / initial stack symbol which is in Γ
Gamma Γ = stack alphabet which can be pushed popped from stack.
δ = Transition function which is used for moving from current state to next state Q × (Σ ∪ {ε}) × Γ to final subsets of Q × F*

→ Instantaneous Description (ID):-
ID is an informal notation of how a PDA computes an i/p string and make a decision that string is accepted or rejected.
An ID has three components δ(qi, w, α).
- q describes the current state
- w describes the remaining i/p.
- α describes the stack contents on the top;

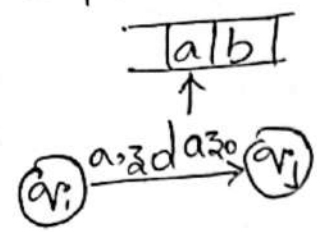
δ(q, w, x) → {(qn, α), ...}. δ: Q × Σ × Γ → Q × Γ*

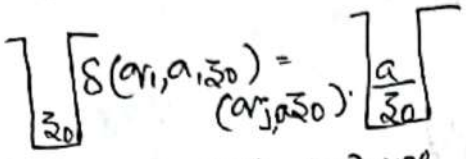
→ qn → New State
→ α → Top of stack to be replaced by x (or) string of stack symbols.

→ Turnstile Notation:-
⊢ Sign describes the turnstile notation and represents one more.
⊢* sign describes a sequence of moves.

→ Basic operations on stack:-

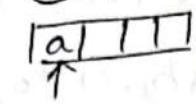
① PUSH

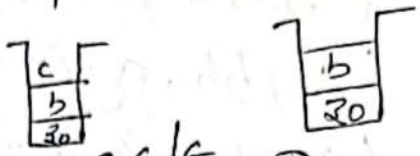




By seeing $(q_j, a z_0)$ we can recognize PUSH operation is done.

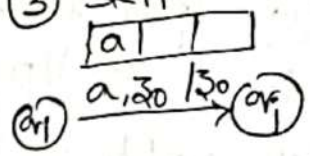
② POP





By seeing (q_j, ϵ) we recognize pop operation is done.

③ SKIP.



By seeing (q_j, z_0) we recognize SKIP operation is done.

→ Design PDA for ~~the~~ accepting the language $L = \{a^n b^n / n \geq 1\}$.

Sol:- $L = \{a^n b^n / n \geq 1\}$

$M = \{Q, \Sigma, q_0, F, z_0, \Gamma, \delta\}$

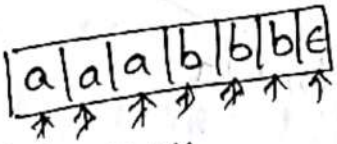
Step-1:- Initially push all a's onto the stack.

Step-2:- Whenever 'b' occurs change the state & pop 'a' from the stack.

Step-3:- Repeat step-2 until stack is empty.

So, $L = \{ab, aabb, aaabbb, \ldots\}$.

Transition function:-



$\delta(q_0, a, z_0) = (q_0, a z_0)$ – PUSH

(input, stack symbol)

$\delta(q_0, a, a) = (q_0, aa)$ – PUSH

$\delta(q_0, a, a) = (q_0, aa)$ – PUSH

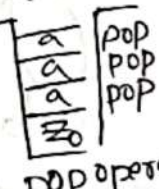
$\delta(q_0, b, a) = (q_1, \epsilon)$ – POP

$\delta(q_1, b, a) = (q_1, \epsilon)$ – POP

$\delta(q_1, b, a) = (q_1, \epsilon)$ – POP

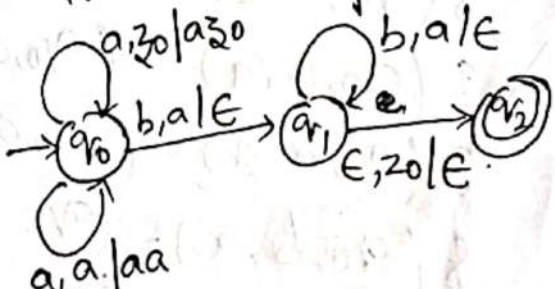
$\delta(q_1, \epsilon, z_0) = \delta(q_2, \epsilon)$

PUSH operation



pop operation

Transition Diagram:-



From the Transition function, lets check whether push & pop operations are done correctly or not.

Lets take i/p string - aabb:

$(q_0, aabb, z_0) \vdash (q_0, abb, az_0)$ PUSH

$\vdash (q_0, bb, aaz_0)$ PUSH → pop 'a'

$\vdash (q_1, b, aaz_0)$ POP

$\vdash (q_1, \epsilon, z_0)$ → pop 'z0'

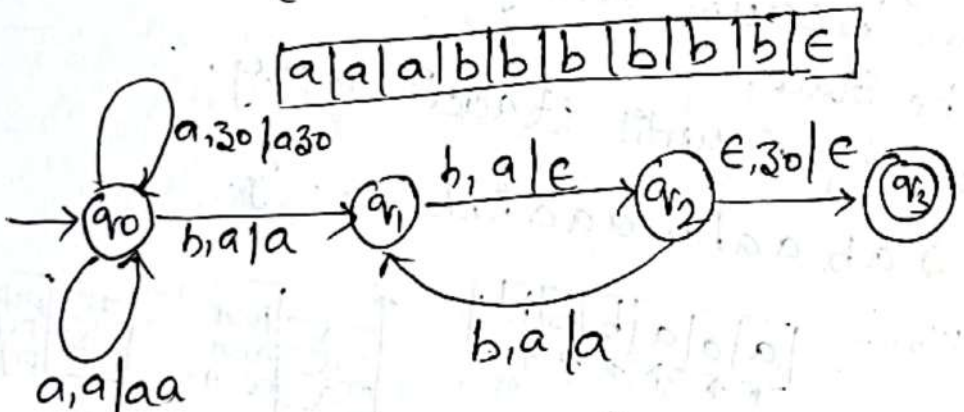
$\vdash (q_2, \epsilon)$

As, it reached final the given string and language is accepted by PDA.

$M = (\{q_0, q_1, q_2\}, \{a, b\}, q_0, \{q_2\}, z_0, \{z_0, a\})$

→ Design PDA for accepting the language $L = \{a^n b^{2n} / n \geq 1\}$

Sol:- $L = \{abb, aabbbb, aaabbbbbb, \ldots\}$.



Transition function:-

$\delta(q_0, a, z_0) = (q_0, az_0)$ PUSH

$\delta(q_0, a, a) = (q_0, aa)$ PUSH

$\delta(q_0, a, a) = (q_0, aa)$ PUSH

$\delta(q_0, b, a) = (q_1, a)$ SKIP

$\delta(q_1, b, a) = (q_2, \epsilon)$ POP

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

$\delta(q_1, b, a) = (q_2, \epsilon)$ POP

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

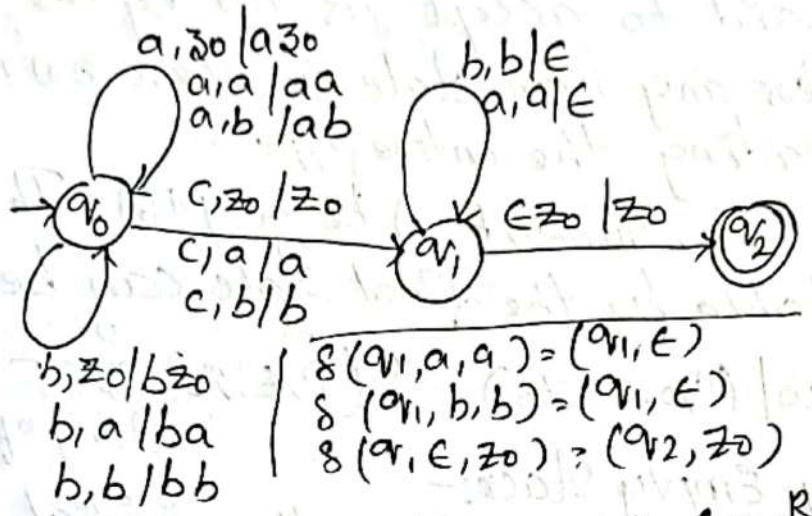
$\delta(q_1, b, a) = (q_2, \epsilon)$ POP

$\delta(q_1, \epsilon, z_0) = (q_3, \epsilon)$ POP

→ Construct a PDA for the language $L = \{|w| = \text{even}\}$, $\Sigma = \{a\}$.

→ Construct a PDA for the language $L = \{a^n b^m c^n / n \geq 1, m \geq 1\}$

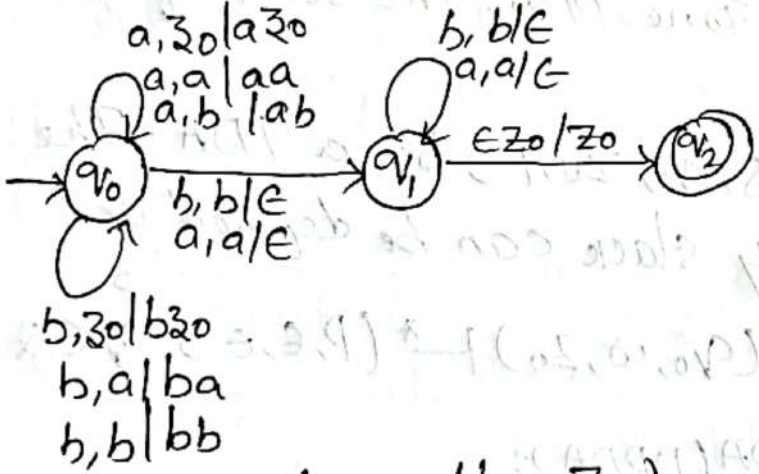
→ Construct a PDA for the language $L = \{wcw^R / w \in (a+b)^*\}$



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, c, Z_0) = (q_1, Z_0)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$

$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

→ Design PDA for language $L = \{ww^R / w \in (a+b)^*\}$



→ string = $\delta(q_0, abba, Z_0)$

$\delta(q_0, abba, Z_0) \vdash \delta(q_0, \underline{b}ba, aZ_0)$

$\vdash \delta(q_0, \underline{b}a, baZ_0)$

Not centre | centre.

$\times (q_0, a, baZ_0)$

$(q_0, \epsilon, abbaZ_0)$

$\vdash (q_1, a, aZ_0)$

$\vdash (q_1, \epsilon, Z_0)$.

$\vdash (q_2, \epsilon)$

→ PDA accepted Languages:-

A language can be accepted by PDA using 2 approaches.

1. Acceptance by Final state:-

The PDA is said to accept its i/p by the final state if it enters any final state in zero or more moves after reading the entire i/p.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language acceptable by the final state can be

$$L(PDA) = \{w \mid (q_0, w, Z_0) \vdash^* (P, \epsilon, \vec{\gamma}), q_p \in F\}.$$

↳ string of stack symbols.

2. Acceptance by Empty Stack:-

On reading the i/p string from the initial configuration for some PDA, the stack of PDA gets empty.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language acceptable by empty stack can be defined as

$$L(PDA) = \{w \mid (q_0, w, Z_0) \vdash^* (P, \epsilon, \epsilon), q_p \in Q\}$$

→ Deterministic PDA (DPDA):-

DPDA is just like DFA which has at most one choice to move for certain i/p. A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if it satisfies both the conditions given as follows:-

1. For any $q \in Q$, $a \in (\Sigma \cup \{\epsilon\})$ & $Z_0 \in \Gamma$, $\delta(q, a, Z_0)$ has at most one choice of move.
2. For any $q \in Q$, $Z_0 \in \Gamma$, if (q, ϵ, Z_0) is defined i.e if $(q, \epsilon, Z_0) \neq \phi$ then $\delta(q, a, Z_0) = \phi$ for all $a \in \Sigma$.

→ Construction of PDA equivalent to a CFG:-

Rule-1:- $\delta(q, \in, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in } P\}$

Rule-2:- $\delta(q, a, a) = \{(q, \in) \mid \text{for every terminal } a\}$.

→ Ex-1:-

Construct PDA for the given CFG

$S \to aAA$

$A \to as \mid bs \mid a$

Sol:-

Terminals: $\{a, b\}$

Stack alphabets $= \{S, A, a, b\}$.

Variables $= \{S, A\}$.

$M = \{\underset{Q}{\{q\}}, \underset{\Sigma}{\{a, b\}}, \underset{\delta}{\delta}, \underset{\Gamma}{\{S, A, a, b\}}, \underset{q_0}{q}, \underset{Z_0}{S}, \underset{F}{\phi}\}$.

where δ is given by

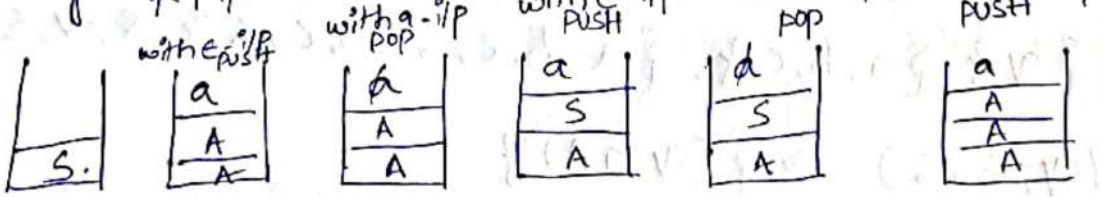
P1: $\delta(q, \in, S) = \{(q, aAA)\}$

P2: $\delta(q, \in, A) = \{(q, as), (q, bs), (q, a)\}$

P3: $\delta(q, a, a) = \{(q, \in)\}$

P4: $\delta(q, b, b) = \{(q, \in)\}$

String: aaabbb.



String aaabbb not accepted.

→ Ex-2:-

Construct PDA for

$S \rightarrow 0A$

$A \rightarrow 0AB \mid 1$

$B \rightarrow 1$

Sol:-

Terminals:- $\{0, 1\}$, stack alphabets :- $\{0, 1, S, A, B\}$

$M = (\{q\}, \{0,1\}, \{0,1,S,A,B\}, \delta, q, S, \phi)$

where δ is given by

P1: $\delta(q, \epsilon, S) \rightarrow \{(q, 0A)\}$

P2: $\delta(q, \epsilon, A) \rightarrow \{(q, 0AB), (q, 1)\}$

P3: $\delta(q, \epsilon, B) \rightarrow \{(q, 1)\}$

P4: $\delta(q, 0, 0) \rightarrow \{(q, \epsilon)\}$

P5: $\delta(q, 1, 1) \rightarrow \{(q, \epsilon)\}$

→ Ex-3:-

Construct PDA for

$S \rightarrow aA$

$A \rightarrow aABC \mid bB \mid a$

$B \rightarrow b$

$C \rightarrow c$

Sol:- Terminals: $\{a, b, c\}$, stack alphabets : $\{a, b, c, A, B, S\}$

$M = (\{q\}, \{a, b, c\}, \{a, b, c, A, B, C, S\}, \delta, q, S, \phi)$

P1: $\delta(q, \epsilon, S) \rightarrow \{(q, aA)\}$

P2: $\delta(q, \epsilon, A) \rightarrow \{(q, aABC), (q, bB), (q, a)\}$

P3: $\delta(q, \epsilon, B) \rightarrow \{(q, b)\}$

P4: $\delta(q, \epsilon, C) \rightarrow \{(q, c)\}$

P5: $\delta(q, a, a) \rightarrow \{(q, \epsilon)\}$

P6: $\delta(q, b, b) \rightarrow \{(q, \epsilon)\}$

P7: $\delta(q, c, c) \rightarrow \{(q, \epsilon)\}$

Ex-4:-

Construct a PDA 'M' equillalent to the CFG, Test whether 010^4 in N(M)

$S \rightarrow OBB$

$B \rightarrow OS \mid 1S \mid 0$.

Sol:-

Terminals: $\{0,1\}$

Stack alphabets: $\{0,1,S,B\}$.

$M = \{\{q\}, \{0,1\}, \{0,1,S,B\}, \delta, q, S, \phi\}$

where δ is given by

P1: $\delta(q,\epsilon,S) \rightarrow \{(q, OBB)\}$

P2: $\delta(q,\epsilon,B) \rightarrow \{(q,0S), (q,1S), (q,0)\}$.

P3: $\delta(q,0,0) \rightarrow \{(q,\epsilon)\}$

P4: $\delta(q,1,1) \rightarrow \{(q,\epsilon)\}$

Given string is $010^4 \rightarrow 010000$

$(q, 010000, S) \vdash (q, 010000, OBB)$ by P1

$\vdash (q, 10000, BB)$ by P3

$\vdash (q, 10000, 1SB)$ by P2

$\vdash (q, 0000, SB)$ by P4

$\vdash (q, 0000, OBBB)$ by P2

$\vdash (q, 000, SBB)$ by P3

$\vdash (q, 000, OBB)$ by P2

$\vdash (q, 00, BB)$ by P3

$\vdash (q, 00, 0B)$ by P2

$\vdash (q, 0, B)$ by P3

$\vdash (q, 0, 0)$ by P2

$\vdash (q, \epsilon)$ by P3.

$\therefore$ Given string is in N(M).

→ Construction of CFL equivalent to the given PDA:-

If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA there exists a CFG 'G' such that $L(G) = N(M)$.

$L(G)$ is CFL.

we define $G = (V, T, P, S)$ where

$V = \{S\} \cup \{[q, Z, q'] / q, q' \in Q, Z \in \Gamma\}$

here $S \rightarrow$ start symbol for G

$q, q' \rightarrow$ states

$Z \rightarrow$ Push down symbol

The productions in 'P' are induced by moves of PDA as follows:-

1. S-productions are given by $S \rightarrow [q_0, Z_0, q]$ for every q in Q.
2. Each move erasing a push Down symbols given by $(q', \epsilon) \in \delta(q, a, Z)$ induces the production $[q, Z, q'] \rightarrow a$.
3. Each more not erasing a push Down symbol given by $(q_1, Z_1, Z_2, \cdots, Z_m) \in \delta(q, a, Z)$ induces many productions of the form

$[q, Z, q'] \rightarrow a[q_1, Z_1, q_2][q_2, Z_2, q_3] \cdots [q_m, Z_m, q_{m+1}]$

where each of the states $q_1, q_2, \cdots q_m$ can be any state in Q.

Ex:- Construct CFG which accepts N(M) where $M = (\{q_0, q_1\}, \{a, b\}, \{z_0, z\}, \delta, q_0, z_0, \phi)$ is given by

$\delta(q_0, b, z_0) = \{(q_0, zz_0)\}$

$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

$\delta(q_0, b, z) = \{(q_0, zz)\}$

$\delta(q_0, a, z) = \{(q_0, z)\}$

$\delta(q_1, b, z) = \{(q_1, \epsilon)\}$

$\delta(q_1, a, z_0) = \{(q_0, z_0)\}$

Sol:-

$G_1 = \{V, T, P, S\}$.

→ V consists of S, $[q_0, z_0, q_0]$, $[q_0, z_0, q_1]$, $[q_1, z_0, q_0]$, $[q_1, z_0 q_1]$, $[q_0, z, q_0]$, $[q_0, z, q_1]$, $[q_1, z, q_0]$, $[q_1, z, q_1]$.

productions for start symbol

P1: $S \rightarrow [q_0, z_0, q_0]$

P2: $S \rightarrow [q_0, z_0, q_1]$

⇒ For $\delta(q_0, b, z_0) = \{(q_0, zz_0)\}$ productions are 2^2

yields by rule(3)

P3: $[q_0, z_0, q_0] \rightarrow b[q_0, z, q_0]\ [q_0, z_0, q_0]$

P4: $[q_0, z_0, q_0] \rightarrow b[q_0, z, q_1]\ [q_1, z_0, q_0]$

P5: $[q_0, z_0, q_1] \rightarrow b[q_0, z, q_0]\ [q_0, z_0, q_1]$

P6: $[q_0, z_0, q_1] \rightarrow b[q_0, z, q_1]\ [q_1, z_0, q_1]$

⇒ for $\delta(q_0, \epsilon, z_0) = \{q_0, \epsilon\}$ yields by rule(2)

P7: $[q_0, z_0, q_0] \rightarrow \epsilon$

For $\delta(q_0, b, Z) = \{(q_0, ZZ)\}$ yields by Rule(3), 2^2.

P8: $[q_0, Z, q_0] \rightarrow b[q_0, Z, q_0]\ [q_0, Z, q_0]$

P9: $[q_0, Z, q_0] \rightarrow b\ [q_0, Z, q_1]\ [q_1, Z, q_0]$

P10: $[q_0, Z, q_1] \rightarrow b\ [q_0, Z, q_0]\ [q_0, Z, q_1]$

P11: $[q_0, Z, q_1] \rightarrow b\ [q_0, Z, q_1]\ [q_1, Z, q_1]$

$\Rightarrow$ For $\delta(q_0, a, Z) = \{(q_1, Z)\}$ yields by Rule(3), 2^1

P12: $[q_0, Z, q_0] \rightarrow a[q_1, Z, q_0]$

P13: $[q_0, Z, q_1] \rightarrow a\ [q_1, Z, q_1]$

$\Rightarrow$ For $\delta(q_1, b, Z) = \{(q, \epsilon)\}$ yields by Rule(2),

P14: $[q_1, Z, q] \rightarrow b$.

$\Rightarrow$ For $\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$ yields by Rule(2), 2^1

P15: $[q_1, Z_0, q_0] \rightarrow a\ [q_0, Z_0, q_0]$ 

P16: $[q_1, Z_0, q_1] \rightarrow a\ [q_0, Z_0, q_1]$ 

$\rightarrow$ Ex-2:-

Give the grammar for the language N(M) where $M = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \phi)$ where δ is given by

$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$, $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$

$\delta(q_0, 0, X) = \{(q_0, XX)\}$ $\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$

$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$ $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

Sol:-

$G_1 = (V, T, P, S)$

$V = S, [q_0, x, q_0], [q_0, x, q_1], [q_1, x, q_0], [q_1, x, q_1], [q_0, z_0, q_0], [q_0, z_0, q_0], [q_1, z_0, q_0], [q_1, z_0, q_1]$.

→ Productions for start symbol

P1: $S \rightarrow [q_0, z_0, q_0]$

$S \rightarrow [q_0, z_0, q_1]$

for $\delta(q_0, 0, z_0) = \{q_0, xz_0\}$.

P3: $[q_0, z_0, q_0] \rightarrow 0\,[q_0, x, q_0]\,[q_0, z_0, q_0]$

P4: $[q_0, z_0, q_0] \rightarrow 0\,[q_0, x, q_1]\,[q_1, z_0, q_0]$

P5: $[q_0, z_0, q_1] \rightarrow 0\,[q_0, x, q_0]\,[q_0, z_0, q_1]$

P6: $[q_0, z_0, q_1] \rightarrow 0\,[q_0, x, q_1]\,[q_1, z_0, q_1]$

for $\delta(q_0, 0, x) = \{q_0, xx\}$

P7: $[q_0, x, q_0] \rightarrow 0\,[q_0, x, q_0]\,[q_0, x, q_0]$

P8: $[q_0, x, q_0] \rightarrow 0\,[q_0, x, q_1]\,[q_1, x, q_0]$

P9: $[q_0, x, q_1] \rightarrow 0\,[q_0, x, q_0]\,[q_0, x, q_1]$

P10: $[q_1, x, q_1] \rightarrow 0\,[q_0, x, q_1]\,[q_1, x, q_1]$

for $\delta(q_0, 1, x) = \{q_1, \epsilon\}$

P11: $[q_0, x, q_1] \rightarrow 1$

for $\delta(q_1, 1, x) = \{q_1, \epsilon\}$

P12: $[q_1, x, q] \rightarrow 1$

for $\delta(q_1, \epsilon, x) = \{q_1, \epsilon\}$

P13: $[q_1, x, q_1] \rightarrow \epsilon$

for $\delta(q_1, \epsilon, z_0) = \{q_1, \epsilon\}$

P14: $[q_1, z_0, q_1] \rightarrow \epsilon$