UNIT- II

Regular expressions

Regular Expressions:- A regular expression is a string that describes the whole set of strings according to a certain syntax rule.

(or)

Regular expressions are useful for representing certain set of strings in an algebraic fashion. RE describes the language accepted by finite state automata.

The following are the notations used in regular expressions

→ ϕ, ε and a belonging to Σ, are all regular expressions and are called the primitive regular expressions.

→ if R_1 and R_2 are two regular expressions then union of these represented as $R_1 \cup R_2$ (or) $R_1 + R_2$ also regular expression

→ if R_1 and R_2 are two regular expressions then concatenation of these represented as $R_1 R_2$ is also regular expressions.

→ the kleene closure of a regular expression R is denoted by R^* is also a regular expression.

→ if R is a regular expression then (R) is also Regular expre.

Example:- if $\Sigma = \{a|b\}$, if $R_1 = c$ and $R_2 = \phi$

then $c + \phi$ i.e $R_1 + R_2$ is also a regular expression.

Regular set :- Any set represented by a regular expression is called regular set.

if a, b are the elements of Σ then regular expressions

→ a denotes the set {a}
→ a+b denotes the set {a, b}
→ ab denotes the set {ab}
→ a^* denotes the set {ε, a, aa, aaa - - -}
→ $(a+b)^*$ denotes the set {ε, a, b, aa, bb, ba, aaa - - -}

Regular language :- A language represented by a regular expression defines a regular language. if 'R' is a regular expression, then L(R) denotes the language associated with R.

The language L(R) denoted by any regular expression R is defined by the following rules.

→ ϕ is a regular expression denoting the empty set.
→ ε is a regular expression denoting the {ε}
→ For every $a \in \Sigma$, a is regular expression denoting {a}.

Example :- RE = ab^*a

i.e L(R) = {aa, aba, abba, abbba - - - -}.

The set of all strings of a's and b's that begin and end with 'a'.

→ R.E = a^*b^+

L(R) = {ε, a, b, ab, bb, aa, abb, aabb - - - -}.

Identity rules (or) Algebra of R.E'S :-

Let R, S and T be any arbitary regular expressions Then, the following properties are true:

1) $(R+S)+T = R+(S+T)$
2) $R+R = R$
3) $R+\phi = \phi+R = R$
4) $R+S = S+R$
5) $R\phi = \phi R = \phi$
6) $R.\epsilon = \epsilon R = R$
7) $(RS)T = R(ST)$
8) $R(S+T) = RS+RT$
9) $(S+T)R = SR+TR$
10) $\phi^* = \epsilon^* = \epsilon$
11) $R^*.R^* = R^* = (R^*)^*$
12) $R.R^* = R^*.R = R^* = \epsilon+RR^*$
13) $(R+S)^* = (R^*S^*)^* = (R^*+S^*)^*$
14) $(RS)^* = (R^*S^*)^* = (R^*+S^*)^*$

Manipulations of Regular Expressions :- (Arden's theorem)

Let P and Q be the two regular expressions over the input set Σ. The regular expression R is given as

$$R = Q+RP$$

which has unique solution as $R = QP^*$

Proof :- if 'P' does not contain ε then there exists R such that $R = Q + RP$ —①

we will replace R by Qp^* in eq ①

Consider R.H.S of equation ①

$$= Q + RP$$
$$= Q + Qp^* p$$
$$= Q(\varepsilon + pp^*) \qquad \boxed{\varepsilon + RR^* = R^*}$$
$$= Qp^*$$

thus $R = Qp^*$ is proved.

To prove that $R = Qp^*$ is a unique solution, we will now replace L.H.S of equation ① by $Q+RP$, then it becomes,

$$R = Q + RP$$
$$Q + RP = Q + RP$$

But again R can be replaced with $Q+RP$

$$= Q + (Q+RP)P$$
$$= Q + QP + QRP^2$$

again replace R by $Q+RP$

$$= Q + QP + (Q+RP)P^2$$
$$= Q + QP + QP^2 + RP^3$$

thus if we go on replacing R by $Q+RP$ then we get,

$$Q + RP = Q + QP + QP^2 \cdots QP^i + RP^{i+1}$$
$$= Q(\varepsilon + P + P^2 + \cdots P^i) + RP^{i+1}$$

from equation ① , $R = Q + RP$

$$R = Q + RP$$

$$R = Q(\varepsilon + P + P^2 \cdots P^i) + RP^{i+1} \quad \text{——②}$$

where $i \geq 0$, consider eq ②

$$R = Q(\underbrace{\varepsilon + P + P^2 \cdots P^i}_{P^*}) + RP^{i+1}$$

$$R = QP^* + RP^{i+1}$$

Let w be a string of length i, In RP^{i+1} has no string of less than $i+1$ length. hence w is not set RP^{i+1}

hence R and QP^* represent the same set, hence it is proved that

$R = Q + RP$ has unique solution $R = QP^*$.

Equivalence of two regular expressions :- one important theorem named arden's theorem which helps in checking the equivalence of two regular expressions.

Ex :- prove $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

Sol :- Let us solve L.H.S first

$$(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$$

we will take $(1+00^*1)$ as a common factor

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

$(\varepsilon + RR^*)$ where $R = (0+10^*1)$ ⋆

As we know $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

$(1+00^*1)((0+10^*1)^*)$ out of this Consider

$\Rightarrow 1(\varepsilon+00^*)(0+10^*1)^*$

Apply $\varepsilon+00^* = 0^*$

$= 10^*(0+10^*1)^*$

$= R.H.S$

Hence two regular expressions are equivalent.

2) simplify the following regular expression

$r = \varepsilon + a^*(abb)^*(a^*(abb)^*)^*$

Sol:- The given regular expression is in the following format

$r = \varepsilon + rr^*$

Considering $r = a^*(abb)^*$

This format is simplified according to identity rule,

$\varepsilon + rr^* = \varepsilon + r^*r = r^*$

$r = (a^*(abb)^*)^*$

again this can be simplified using identity rule

$r = (r_1^* r_2^*)^* \quad (\because r_1 = a \; r_2 = abb)$

using $(p+q)^* = (p^*q^*)^*$

$= (p^*+q^*)^*$

$\Rightarrow (p+q)^*$

thus $r = (a+abb)^*$

//

Equivalence between finite automata and Regular Expressions:-

Equivalence of DFA and Regular expressions:-

If L is accepted by a DFA, then L can be expressed by a regular language. In otherwords, for any given DFA, we can obtain a RE and vice-versa.

Following are the two methods of constructing R.E from a given DFA.

→ By solving equations

→ By using transition diagrams

Construction of RE from DFA by solving equations:- Following are the steps required to construct a R.E.

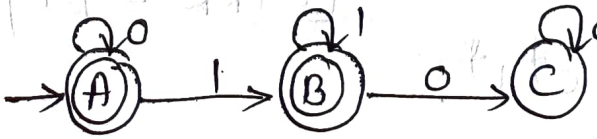
Step1:- For each of the states $q_1, q_2 \cdots q_n$ in DFA, write down the equations by considering all edges, that enter into that state.

Step2:- For the initial state of DFA, the equation is added with ε.

Step3:- Compute the equations for each state.

Step4:- Substitute the results of each state equation into the final state equation of DFA, to get a R.E for DFA.

Example:- Construct R.E from the DFA



Sol:- writing the equations for each state, by considering all edges that enter into that state:

$$A = \varepsilon + A0$$
$$B = A1 + B1$$
$$C = B0 + C(0+1)$$

Since, $R = Q + RP \Rightarrow R = Qp^*$

Consider,

$$A = \varepsilon + A0$$
$$R = Q + RP$$

$R = A \quad Q = \varepsilon \quad p = 0$

$R = Qp^* \Rightarrow \varepsilon \cdot 0^*$

$A = 0^*$

Now substitute A in B

$$B = A1 + B1$$
$$B = 0^*1 + B1$$

$R = Q + RP \Rightarrow Qp^*$

$B = 0^*1.1^*$

Since A and B are final states in the above DFA, the addition of A and B gives,

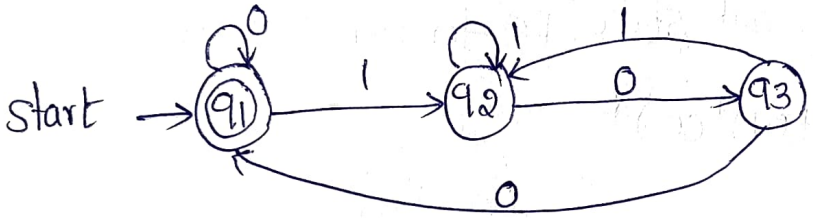
$$\Rightarrow A + B = 0^* + 0^*11^*$$
$$= 0^*(\varepsilon + 11^*)$$
$$= 0^*1^* \quad [\varepsilon + RR^* = R^*]$$

$R.E = 0^*1^*$.

→ Construct a RE corresponding to the DFA represented in below in below table. q_1 is both the initial and final state.

Σ / Q	Present i/p. 0	1
*q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_1	q_2

Sol:- the transition diagram is



Now the equations for each state can be written, by considering all edges that enter into that state,

thus $q_1 = q_10 + q_30 + \epsilon$ (ϵ, only for initial state)

$q_2 = q_21 + q_31 + q_11$

$q_3 = q_20$

substituting q_3 in q_2

$\Rightarrow q_2 = q_11 + q_21 + q_31$

$= q_1 1 + q_2 1 + q_2 01$

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

$= q_1 1 (1+01)^*$ ($R = Q + RP$ is $R = QP^*$)

Now,

$$q_1 = q_1 0 + q_3 0 + \epsilon$$
$$= \epsilon + q_1 0 + q_2 0.0$$
$$= \epsilon + q_1 0 + q_1 1(1+01)^*.00$$
$$= \epsilon + q_1(0 + 1(1+01)^*).00)$$
$$= \epsilon + q_1(0+1(1+01)^*.00)$$

$[R = Q + RP \Rightarrow R = Qp^*]$ $\quad Q = \epsilon$, $R = q_1$, $P = (0+1(1+0)^*.00)$

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

since q_1 is the final state, RE for DFA will be

$$RE = (0+1(1+01)^*00)^*$$

2. Construction of R.E from DFA by using Transition diagram :-

the following are some transition diagrams and their regular exp's.

→ if a transition diagram has no final state, then $RE = \phi$ i.e FA does not accept any string.

$RE = \phi$

→ if the TD has the same initial and final states, then $RE = \epsilon$ i.e FA accepts one of the string as ϵ

$RE = \epsilon$

→ For a transition diagram, with transition to itself

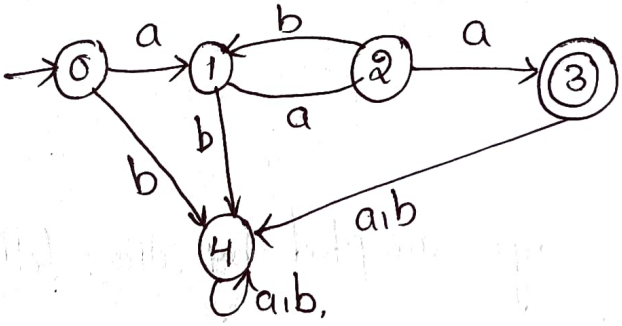
RE = b^* (or) RE = $(a+b)^*$

RE = b

RE = $(a+b)^*$

Example :-

Construct the R.E from the given DFA is.



Sol:-

RE = $a(ab)^* aa$

First from the start state it goes to state 1 by reading one 'a'. then from state 1, it goes through the cycle 1-2-1 any no. of times by reading the sub-string ab and come back to state 1. This is represented by $(ab)^*$. Then from state 1, it goes to state 2 and then to state 3 by reading aa. Thus

RE = $a(ab)^* aa$.

—o—

Language acceptance of FA from transition diagram :-

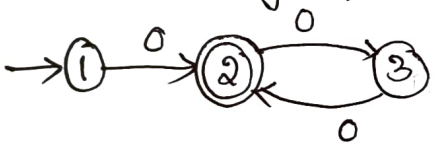
if there is a path in the transition diagram such that, it

- → begins at start state
- → ends at an accepting state.
- → has a sequence of labels $w_1, w_2 \ldots w_n$

In general, the language accepted by FA, first and find the RE from the given FA say 'M', then $L(M) = RE$.

Example :-

what is the language accepted by the following DFA'S ?



Sol :- $RE = 0.(00)^*$

$\Rightarrow L(M) = 0(00)^*$

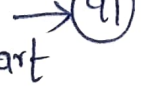
i.e machine which accepts all zero's.

Construction of DFA for a given RE :-

Example :- Construct a DFA for the RE given below.

$RE = \phi$.

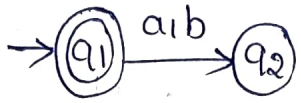
Sol :-



This DFA accepts nothing, since it has no final state.

→ RE = ϵ over Σ = {a,b}

Sol:-



DFA for accepting ϵ (since initial and final states are same)

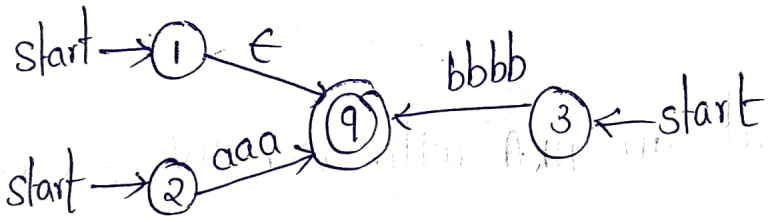
→ RE = (a+b)*

Sol:-

DFA that accepts ϵ or a's or b's.

→ RE = ϵ+aaa+bbbb over Σ = {a,b} accept only words ϵ, aaa, bbbb

Sol:-

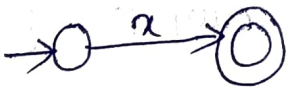


Equivalence of NFA and regular Expressions :-

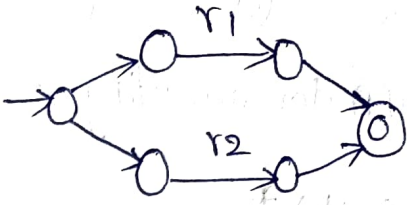
The rules for conversion of regular expressions into finite automata.

	Reg. Exp	meaning	finite automata.
1)	ϵ	null	
2)	ϕ	-	

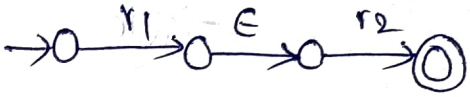
3) x $x \in \Sigma$



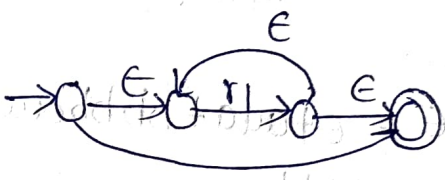
4) $r_1 + r_2$ Parallal alternate path



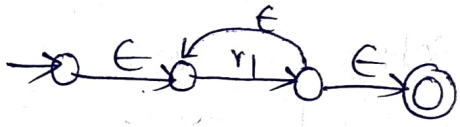
5) $r_1 . r_2$ Series (or) Continuous path



6) r_1^* Kleene closure



7) r_1^+ positive closure



Examply:- Construct an NFA with ϵ-moves for 00^*+1

Sol:- let $r = 00^*+1$

$r = r_1 + r_2$

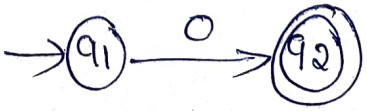
$\Rightarrow r_1 = 00^* \quad r_2 = 1$

$r_1 = r_3 r_4$

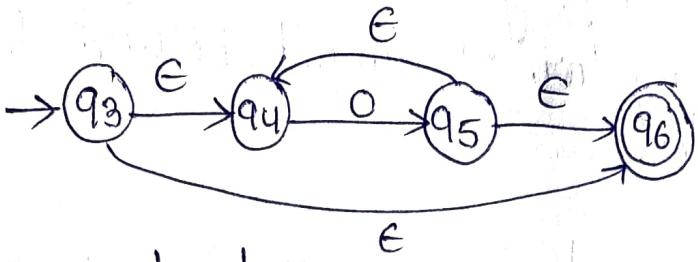
$r_3 = 0 \quad r_4 = 0^*$

Construct a machine for r_3 : $r_3 = 0$

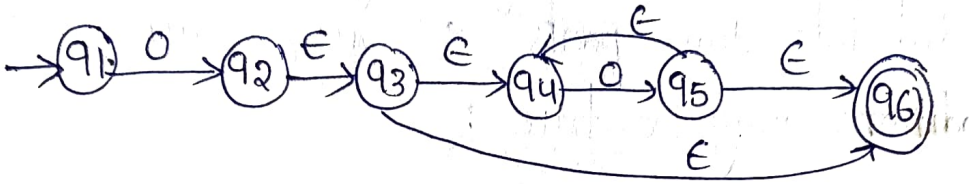
→ To accept 0 :



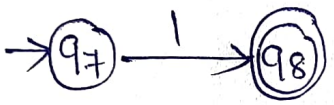
$r_4 = 0^*$



Now construct machine for $r_1 = r_3 . r_4$

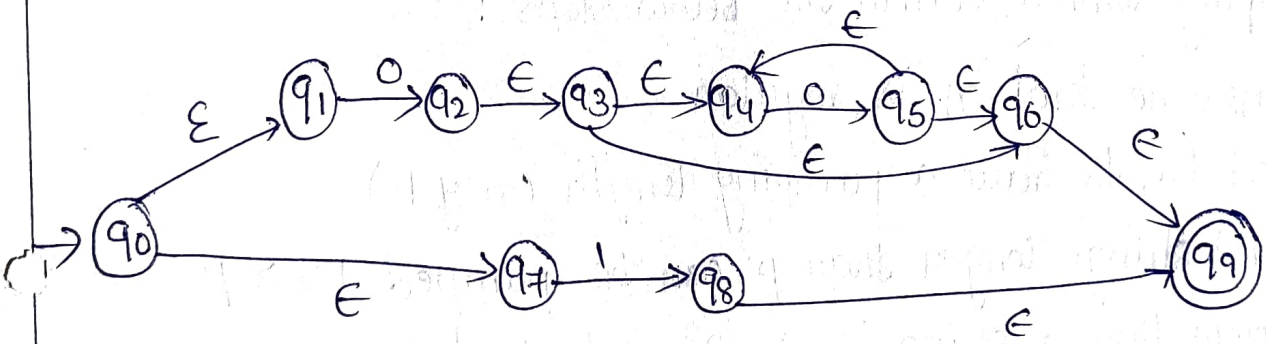


$r_2 = 1$



Now $r = r_1 + r_2$

$r = 00^* + 1$



This is the final NFA for the given regular expression.

Pumping lemma :- pumping lemma is used to prove that a language is not regular. It can not be used to prove that a language is regular.

Statement :- If A is a regular language, then A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be devided into three parts $s = xyz$ such that the following conditions may be true :

i) $xy^iz \in A$ for every $i \geq 0$

ii) $|y| > 0$

iii) $|xy| \leq p$

Proof :- To prove that a language is not regular using pumping lemma, follow the below steps :

→ Assume that A is regular

→ It has to have a pumping length (say p)

→ All strings longer than p can be pumped $|s| \geq p$

→ Now find a string 's' in A such that $|s| \geq p$

→ Devide s into xyz

→ show that $xy^iz \notin A$ for some i

→ Then Consider all ways that can be devided into xyz

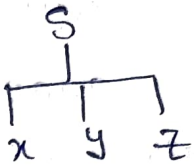
→ Show that none of these can satisfy all 3 pumping conditions at the same time.

→ s cannot be pumped = Contradiction.

Example :- Using pumping lemma prove that the following language $A = \{a^n b^n \mid n \geq 0\}$ is not regular.

Proof :- Assume that A is regular

Pumping length = P



$S = a^P b^P$

Devide S into xyz

then $P = 7$, $S = a^7 b^7$

S = aaaaaaabbbbbbb

Case 1 : The y is in the a part

aa aaaaa bbbbbbb
x y z

Now $xy^i z \Rightarrow xy^2 z$ (where $i = 2$)

$= xyyz$

$=$ aa aaaaa aaaaa bbbbbbb
(y) (y)

Here no. of a's are not equal to no. of b's

$= 11 \neq 7$

Case 2 :- The y is in the b part

aaaaaaa bb bbbbb b
x y z

Now $xy^i z = xy^2 z$ $(i = 2)$

aaaaaaa bb bbbb bbbbb

no. of a's $\neq$ no. of b's

$7 \neq 11$

Case 3:- The y is in the a part and b part

$\underbrace{aaaaaa}_{x}\ \underbrace{aa}_{y}\ \underbrace{bb}_{z}\ bbb$

Now $xy^iz = xy^2z$

aaaaaa aabbaabb bbbbb

Here also no. of a's not equal to no. of b's. hence the rule a^nb^n.

Condition 2 :- $|y| > 0$

$|4| > 0$, is true

condition 3 :- $|xy| \leq p$ where $p = 7$

$|3 \leq 7$ false

$6 \leq 7$ true

$9 \leq 7$ false hence false

So assume A is regular, it is contradiction hence the language A is not regular from pumping lemma.

$xy^iz \notin A$

Show that the language $L = \{a^i a b^{2i} \mid i > 0\}$ is not regular.

Sol:- The set of strings accepted by language L is

$L = \{abb, aabbbb, aaabbbbbb \ldots\}$

Applying pumping lemma for any of the strings above take the string abb

it is of the form xyz, where $|xy| \leq i, |y| \geq 1$

abb
x y z
x=a y=b z=b

To find i such that $uxy^iz \notin L$ (where i=2)

xy^2z = a(bb)b
= abbb

hence xy^2z = abbb $\notin L$

Since abbb is not present in the strings of language.

∴ L is not regular.

Applications of Regular expressions :-

1) Validation :- checking the correctness of inputs i.e. whether the given string compiles with a set of formatted constraints/not.

2) Search and selection :- Identification of a subset of items from a larger set, on the basis of pattern match i.e regular expression.

3) Tokenisation :- Conversion of a string of characters into a sequence of words for later interpretation i.e generation of program called lexical analyzer.

4) Regular expression offers a declarative way to express set of strings.

5) Regular expressions are used to define languages. they define the languages accepted by finite automata exactly.

Closure properties of regular sets / languages :-

Theorem :- Regular languages are closed under union, concatenation, and kleene closure.

Proof :- method 1 :-

Given two regular languages - L_1 and L_2 if R_1 and R_2 are two regular expressions, then

→ R_1+R_2, denotes the union of two regular languages L_1 and L_2, $L_1 \cup L_2$ is also regular.

→ $R_1 \cdot R_2$ denotes the union concatenation of two regular languages L_1 and L_2, $L_1 \cdot L_2$ is also regular.

→ R_1^* denotes the kleene closure of $L_1 = L_1^*$ is regular.

If R_1 and R_2 are two regular expressions with $R_1 = \{ab, c\}$, $R_2 = \{d, ef\}$.

then $R_1 \cup R_2 = \{ab, c\} \cup \{d, ef\}$

$= \{ab, c, d, ef\}$

$R_1 \cdot R_2 = \{ab, c\} . \{d, ef\}$

$= \{abd, abef, cd, cef\}$

If $R_1 = \{ab, c\}$

then $R_1^* = \{\varepsilon, c, ab, abc, abab, cab \ldots\}$

→ if r=a+b, then the language of r is

method 1:-

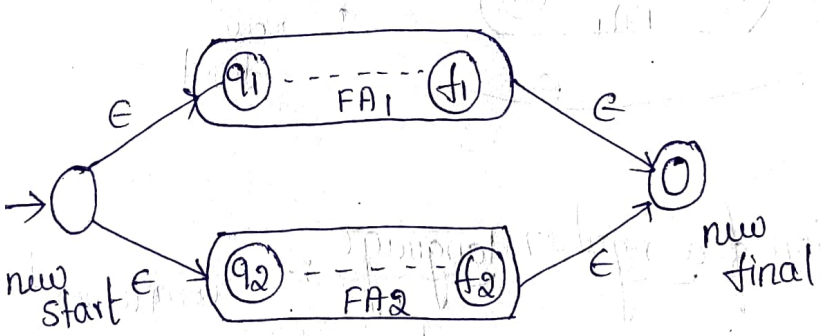
$$L(r) = L(a+b)$$
$$= L(a) \cup L(b)$$
$$= \{a\} \cup \{b\}$$

$L(r) = \{a, b\}$ is also a regular.

method 2:- Assume that FA_1 and FA_2 are two finite automata's accepting languages L_1 and L_2 defined by regular expressions r_1 and r_2 respectively, as.

$$FA_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1), \quad FA_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2)$$

Construction of $L_1 + L_2$:-



→ if r=ab*a, then the language of r is

method 1:-

$$L(r) = L(ab^*a)$$
$$= L(a) \cdot L(b^*) \cdot L(a)$$
$$= \{a\} \cdot \{b(b)\}^* \cdot \{a\}$$
$$= \{a\} \cdot \{\epsilon, b, bb, bbb \cdots\} \cdot \{a\}$$

$L(r) = \{aa, aba, abba \cdots\}$ is also regular.

method 2 :- Construction of $L_1 L_2$

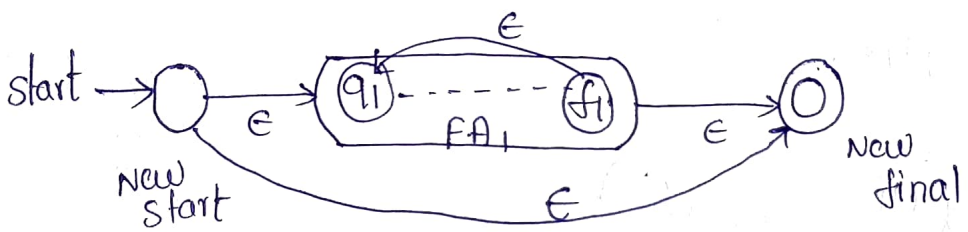
→ if $r = a^*$, then the language of r is

method 1:- $L(r) = L(a^*)$

$= (L(a))^*$

$L(r) = \{\epsilon, a, aa \ldots\}$ is also regular.

method 2:- Construction of L_1^*



Complementation of a regular language :- Complement of a language L is a language, such that,

$$\bar{L} = \Sigma^* - L$$

Ex :- $\Sigma = \{a, b\}$ $L = \{a, b, aa\}$ then $\bar{L}$ as follows

$\bar{L} = \{\epsilon, a, b, aa, bb, aaa, bbb, ba, bab, abbab \ldots\} - \{a, b, aa\}$

$= \{\epsilon, bb, aaa, bbb \ldots\}$

Intersection :- Intersection of two regular languages L_1 and L_2 is defined as $L_1 \cap L_2$. is also regular.

Theorem :- Regular languages are closed under complementation.

proof :- if L is a regular language over the alphabet Σ, then there is a DFA, $D = \{Q, \Sigma, \delta, q_0, F\}$ accepting $L = L(D)$.

To accept $\bar{L} = \Sigma - L$, Complement the final states of D. For this Consider another DFA,

$$D' = (Q, \Sigma, \delta, q_0, F')$$

where $F' = Q - F$ i.e D and D' differ only in final states

Now, if x is a input symbol, then $x \in L(D)'$

$\Rightarrow \delta(q_0, x) \in F'$

$\Rightarrow \delta(q_0, x) \in Q - F$

$\Rightarrow \delta(q_0, x) \notin F$

$\Rightarrow x \notin L$

$\Rightarrow x \in L'$

$\Rightarrow x \in \Sigma^* - L$.

$\therefore L(D') = \bar{L} = \Sigma^* - L$.

Hence $\bar{L}$ is a regular language since it is accepted by DFA.

Intersection of two regular languages :- Intersection of two regular languages L_1 and L_2 is defined as $L_1 \cap L_2$.

Theorem :- Regular languages are closed under intersection.

proof :- Given two regular languages L_1 and L_2

applying demorgan's law,

$$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$$

since L_1, L_2 is ∄ regular.

$\Rightarrow \bar{L_1}, \bar{L_2}$ is a regular (regular sets closed under complementation)

$\Rightarrow \bar{L_1} \cup \bar{L_2}$ is a regular (" " " " union)

$\Rightarrow \overline{\bar{L_1} \cup \bar{L_2}}$ is a regular (" " " " complementation)

$\Rightarrow L_1 \cap L_2$ is regular (by demorgan's law)

Formal Grammar :- A grammar is a notation for defining a language through finite number of rules.

Two main categories of formal grammar are :

→ Generative grammars
→ Analytic grammars.

Generative grammars :- These grammars are set of rules for generation of strings in a language. It describes how to write only those strings in the set.

Analytic grammars :- These grammars are set of rules to determine whether a string is member of the language. It describes how to write only those strings to recognize the strings which are members of a set.

Generative grammars :-

Components of generative grammars :- The generative grammar G1 is consists of the following components :

→ A finite set V of non-terminal symbols

Formally, the generative grammar G1 is quad-tuple.

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

where
→ V is finite set of non terminals
→ T is a finite set of terminals.

→ P is the finite set of productions rules, each of the form, $(T \cup V)^* V (T \cup V)^* \rightarrow (T \cup V)^*$

each production rule maps from one string of symbols to another.

→ S is the start symbol, $S \in V$.

the language of a generative grammar G, denoted by L(G).

Classification of context free grammars In 1956, Noam chomskey classified the generative grammars into types known as the chomskey hierarchy. Each type of generative grammar distinguishes the other types in the form of production rules. The following are the types,

→ unrestricted (or) phrase structure grammar
→ Context-sensitive grammar.
→ Context-free grammar
→ Regular grammar.

unrestricted grammar :- An unrestricted grammar is a formal grammar, on which no restrictions are made on the left and right sides of the grammar's productions.

formally, unrestricted grammar G is a quad-tuple,

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

where V, T, S are same as in the case of generative grammar and P is the finite set of production rules, each of the form,

$$(VUT)^+ \rightarrow (VUT)^*$$

i.e no ϵ on the left hand side of any productions. However ϵ can appear on the right hand side of production.

Ex :- Let $G = (V, T, P, S)$ be unrestricted grammar whose production rules are,

$$P = \{ S \rightarrow AS \mid \epsilon$$
$$A \rightarrow aA \mid a$$
$$aaaA \rightarrow ba \}$$

The derivation for $\omega = bbaa$ is as follows,

$S \Rightarrow AS$ [$S \rightarrow AS$]

$\Rightarrow AAS$ [$A \rightarrow AS$]

$\Rightarrow AAAS$ [$S \rightarrow \epsilon$]

$\Rightarrow AAA$ [$A \rightarrow aA$]

$\Rightarrow aAAA$ [$A \rightarrow aA$]

$\Rightarrow aaAAA$ [$A \rightarrow aA$]

$\Rightarrow aaaAAA$ [$aaaA \rightarrow ba$]

$\Rightarrow baAA$ [$A \rightarrow aA$]

$\Rightarrow baaAA$ [$A \rightarrow aA$]

$\Rightarrow baaaAA$ [$aaaA \rightarrow ba$]

$= bbaA$ [$A \rightarrow a$]

$\Rightarrow bbaa$ //

In chomskey hiirarchy, an unrestricted grammar is called type 0 grammar. The language accepted/generated by this grammar is recursively enumerable language and the recognizers are Turing machines.

$\{ww | w \in \{a,b\}\}$, $\{a^{n^2} b^n | n \geq 0\}$ and $\{a^{n!} | n \geq 0\}$ are some of the languages that are generated by unrestricted grammar.

Context-sensitive grammar :-

A context sensitive grammar is a formal grammar, in which the left hand sides and right side of the productions may be surrounded by a context of terminals and non-terminal symbols.

Formally, a context-sensitive grammar G_1 is a quad-tuple $G_1 = (V, T, P, S)$

where V, T, S are same as generative grammar, and P is the finite set of productions, each of them,

$$(VUT)^+ \rightarrow (VUT)^+$$

i.e no ϵ on the left and right-hand sides of any productions hence this grammar is ϵ-free.

Ex :- Let $G_1 = (V, T, P, S)$ be a context sensitive gramar, whose production rules are,

$P = \{ S \to aBb,\ aB \to bBB,\ bB \to aa,\ B \to b \}$

the derivation for $w = aaabb$ as follows,

$S \Rightarrow \underline{aB}b$
$\Rightarrow b\underline{BB}b$
$\Rightarrow aa\underline{B}b$
$\Rightarrow ab\underline{B}Bb$
$\Rightarrow aaa\underline{B}b$
$\Rightarrow aaabb.$

In chomskey hirarchy, context sensitive grammar is called type1 grammar. the language generated by this grammar is context sensitive language and the recogniser is LBA (linear bounded automata).

The language $\{a^n b^n c^n | n \geq 1\}$ is context-sensitive lang.

3) Context-free grammar :-

A context free grammar is a grammar in which the left hand side of each production rule consists of only a single non terminal symbol. ~~This restriction~~ does not make

formally, a context free grammar G_1 is a quad-tuple

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

where V, T, S are same as generative grammar and P is the finite set of production, as each of the form,

$$A \to (V \cup T)^*$$

Any no. of terminals & non terminals including ε on right side of production

Ex :- Let $G = (V, T, P, S)$ be a context free grammar whose production rules are,

$$P = \{ S \to asa, S \to bSb | a | b | \varepsilon \}$$

The derivation for $w = ababa$ is as follows,

$$S \Rightarrow asa$$
$$\Rightarrow absba$$
$$\Rightarrow ababa$$

In chomskey hierarchy, Context free grammar is called type 2 grammar. The language generated by this grammar is context free language and the corresponding recogniser is PDA (Push down automata).

Ex :- The language $\{a^n b^n | n \geq 1\}$ is context free language.

Regular grammar :- A regular grammar is a formal grammar with the restrictions on both left and right hand sides of the productions.

→ Left hand side of each production rule consists of only a single non-terminal symbol.

→ Right side of production rule may be nothing, (or) a single terminal (or) a single terminal symbol followed by non-terminal symbols, but nothing else.

Formally, a regular grammar G_1 is a quad-tuple,

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

where V, T, S are same as generative grammar and P is the finite set of productions, each of the form,

$A \to t$ (or) $A \to tB$ (or) $A \to \varepsilon$

where $A, B \in V$ and $t \in T$.

Ex:- Let $G = (V, T, P, S)$ be a regular grammar, whose production rules are $P = \{ S \to aS | a \}$

The derivation for $\omega = aaa$ is as follows

$S \Rightarrow aS$

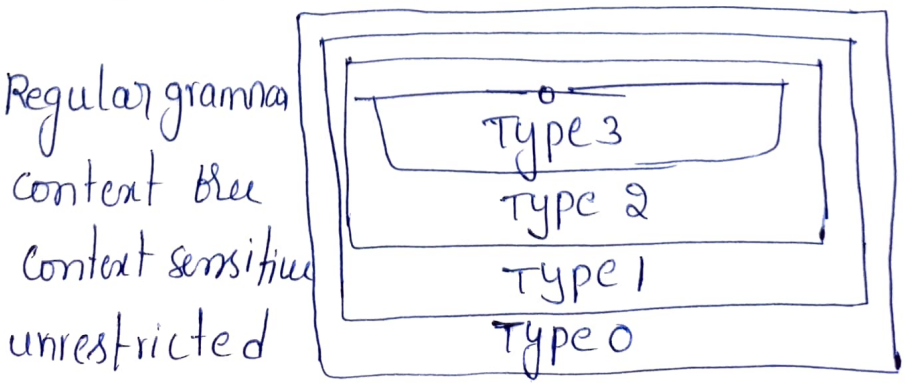
$\Rightarrow aaS$

$\Rightarrow aaa$

In chomsky hierarchy, regular grammar is called a type 3 grammar. the language generated by this grammar is a regular language and the recogniser is FA (finite automata).

the language $\{a^n b^m | m, n \geq 1\}$ is a regular language, and

· Regular grammars are commonly expressed using regular expressions.



Left linear and Right linear grammars :- we discuss some of the Commonly used grammars,

Linear and Nonlinear grammars :-

A grammar with atmost one variable (Non terminal) at the right side of a production, is a linear grammar, otherwise it is non linear grammar.

Example: (linear grammars)

i) $S \to asb$
$S \to \varepsilon$

ii) $S \to Ab$
$A \to aAb$
$A \to \varepsilon$

iii) $S \to A$
$A \to aB \mid \varepsilon$
$B \to Ab$

(Non-linear grammars)

i) $S \to SS$
$S \to \varepsilon$
$S \to asb$
$S \to bSa$

ii) $S \to aA$
$A \to \varepsilon$
$S \to aSAS$

Linear grammars are further classified into 2 types

1. Right linear grammars
2. Left linear grammars

1. Right linear grammar :-

A grammar $G_1 = (V, T, P, S)$ is a right linear, if all productions are of the form

$A \rightarrow xB$

$A \rightarrow x$

where $A, B \in V$ and $x \in T^*$

Ex :- $S \rightarrow abS$

$S \rightarrow a$

Left linear grammar :- A Grammar $G = \{V, T, P, S\}$ is left linear, if all productions are of the form,

$A \rightarrow Bx$

$A \rightarrow x$

where $A, B \in V$ and $x \in T^*$

Ex :- $S \rightarrow Aab$

$A \rightarrow Aab \mid B$

$B \rightarrow a$.

3) Regular Grammar :- A Grammar $G = \{V, T, P, S\}$ is said to be regular, if it is either right linear (or) left linear.

Ex :- $S \rightarrow abS$ $\quad$ $S \rightarrow Aab$

$S \rightarrow a$ $\quad$ $A \rightarrow Aab \mid B$

$B \rightarrow a$

4) non regular grammar :- A grammar $G = \{V, T, P, S\}$ is non-regular, if it is neither right linear nor left linear.

Ex :- $S \rightarrow asb$ $\quad$ $S \rightarrow \epsilon$

$S \rightarrow \epsilon$ $\quad$ $S \rightarrow asb$

$S \rightarrow bSa$

//

5) simple (or) S-grammar :- A grammar $G_1 = \{V, T, P, S\}$ is said to be a simple grammar (or) s-grammar, if all productions are of the form

$$A \to ax$$

where $A \in V$, $a \in T$ and $x \in V^*$ and any pair (A, a) occurs atmost once in P.

Ex :- $S \to as | bss | c$ is simple grammar, since pairs (S, a), (S, b) occur only once in production.

$S \to as | bss | ass | c$ is not simple grammar, since pair (S, a) occurs twice in production as & ass.

6) Recursive grammar :- A Grammar $G = \{V, T, P, S\}$ is recursive, if it contains either a recursive production (or) an indirectly recursive production.

Ex :- $S \to as$ $\quad$ $S \to SS$
~~$S \to SS$~~ $\quad$ $S \to \epsilon$
$\quad$ $\quad$ $S \to asb$.

Derivation Tree :- when deriving the string w from S, if every derivation is considered to be a step in the tree construction, then the graphical display of the derivation of string w results in a tree structure.

This is called a derivation tree (or) parse tree (or) generation tree (or) production tree.

→ Relation between regular grammar and finite automata:-

The following relation exists between regular grammar and finite automata.

1) Finite automata from regular grammar :- For a given right linear grammar G, there exists a language L(G), which is accepted by a finite automata.

2) Regular grammar from finite automata :- For a given finite automata M, there exists a right linear grammar G, such that L(M) = L(G).

3) Regular expression from regular grammar :- For a given regular grammar G, there exists a regular expression that specifies L(G).

1) Finite automata from regular grammar :-

Let G = (V, T, P, S) be a right linear grammar let V = {V0, V1 --} be the variables and productions P(G) be

$$P = \{ V_i \rightarrow a_1 a_2 \cdots a_m V_j$$

(or)

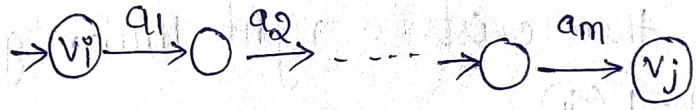
$$V_i \rightarrow a_1 a_2 \cdots a_m \}.$$

We construct the finite automata M = (Q, Σ, δ, q0, F) from these productions, by using the following steps.

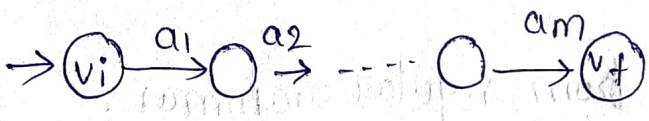
step1 :- start symbol of grammar is the start symbol of M ($q_0 = v_0$).

step2 :- each variable v_i, of G, corresponds to a state in M.

step3 :- for each production of the form $v_i \to a_1 a_2 \ldots a_m v_j$, if the transitions of M are of the form $\delta(v_i, a_1 a_2 \ldots a_m) = v_j$ add the transitions and intermediate states.



step4 :- For each production of the form $v_i \to a_1 a_2 \ldots a_m$ if the transitions of M are of the form $(v_i, a_1 a_2 \ldots a_m) = v_f$ (where v_f is the final states of M), add the transitions and intermediate states.



Example :- Construct the finite automata to accept the language generated by the following grammar.

$S \to aA \mid B$

$A \to aaB$

$B \to bB \mid a$

Sol :- Let G be the right linear grammar and M is the finite automata

step1 :- The start symbol of G is the start symbol for M, i.e $q_0 = \{S\}$

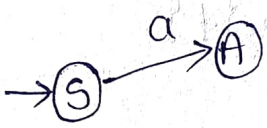
→ (S)

Step2
⇒ Every variable V = {S, A, B} of G, corresponds to state M.

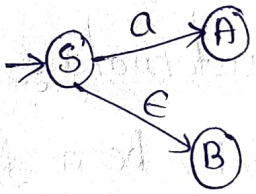
Step3 :-

Compute the transitions of M for all production of the form $V_i \rightarrow a_1 a_2 - a_m V_j$.

a) $S \rightarrow aA$

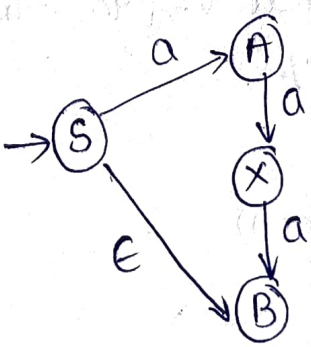


b) $S \rightarrow B$

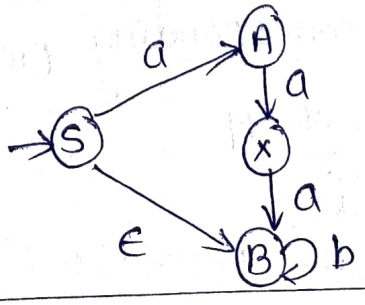


Step4 c) $A \rightarrow aaB$

add transitions and intermediate states (X)

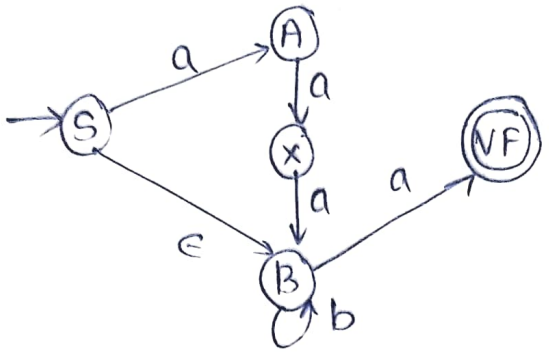


d) $B \rightarrow bB$



Step4 :-
→ Compute for all productions of the form $V_i \to a_1 a_2 \cdots a_m$

$B \to a$



Final M with $L(M) = L(G) = aaab^*a + b^*a$

Thus $M = (Q, \Sigma, \delta, q_0, F)$

where $Q = \{S, A, X, B, V_f\}$

$\Sigma = \{a, b\}$

$q_0 = S$, $F = V_f$.

2) Regular grammar from finite automata :-

Let $M = (Q, \Sigma, \delta S, q_0, F)$ be a finite automata where $Q = \{q_0, q_1 \cdots q_n\}$ is the finite number of states and $\Sigma = \{q_1, q_2 \cdots a_m\}$ is the set of input symbols. construct a regular grammar $G = (V, T, P, S)$, where $V = \{q_0 \cdots q_n\}$, $T = \Sigma$ and $S = q_0$. by using the following steps.

Step1 :- For any transition of the form

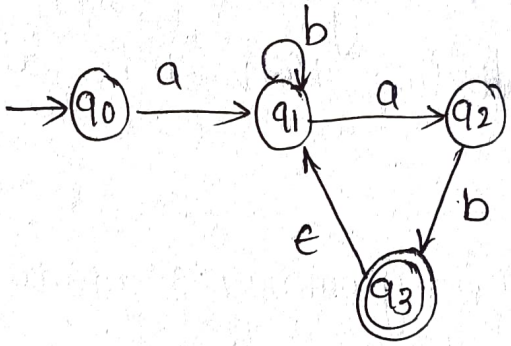
i.e $\delta(q_i, a) = q_j$, the corresponding production is $q_i \to a q_j$

Step2 :- For any final state of M

the production added is $q_f \to \epsilon$

K.Lavanya

Example :- Construct a regular grammar from the following finite automata.



Sol :- For each transition of FA, the corresponding productions are

$q_i \xrightarrow{a} q_j$ productions

$q_0 \xrightarrow{a} q_1$	$q_0 \to aq_1$
$q_1 \xrightarrow{b} q_1$	$q_1 \to bq_1$
$q_1 \xrightarrow{a} q_2$	$q_1 \to aq_2$
$q_2 \xrightarrow{b} q_3$	$q_2 \to bq_3$
$q_3 \xrightarrow{\epsilon} q_1$	$q_3 \to q_1$
$q_3 \Rightarrow q_f$	$q_3 \to \epsilon$

Thus $G_1 = (V, T, P, S)$ is the required grammar, where $V = \{q_0, q_1, q_2, q_3\}$, $T = \{a, b\}$, $S = \{q_0\}$, $F = \{q_3\}$ and

$P = \{$
$q_0 \to aq_1$
$q_1 \to bq_1$
$q_1 \to aq_2$
$q_2 \to bq_3$
$q_3 \to q_1$
$q_3 \to \epsilon$
$\}$

— o —

3) **Regular Expressions from regular grammar :-**

Let $G = (V, T, P, S)$ be a right linear grammar. A regular expression R, that satisfies $L(G)$, can be directly obtained as follows,

→ Replace the '→' symbols in the grammar's productions with '=' symbols to get a set of equations.

→ Convert the equations of the form $A = aA|b$, as $A = a^*b$.

→ solve the set of equations obtained, to obtain the value of variable S. The result is the regular expression, $L(G)$.

Example:- obtain the regular expression, for the grammar given below.

$S \rightarrow 0|B|0$

$B \rightarrow 1B|11$

Sol:-

Replace '→' by '=' in the above productions.

so that, $S = 0|B|0$ — ①

$B = 1B|11$ — ②

Now from $B = 1B|11$

$\Rightarrow B = 1^*11$ [$A = aA|b$ is $A = a^*b$]

substituting for B in equation (1) we get,

$S = 0|B|0$

$\Rightarrow S = 01^*11|0$

(or) $S = (01^*11+0)$ this is the regular expression.

Conversion of LLG to RLG :-

Procedure :-

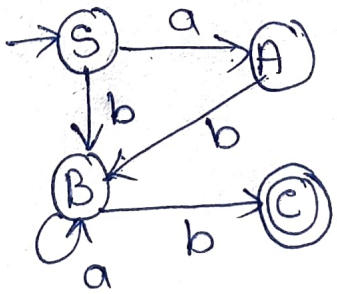
1) Convert the given left linear grammar to FA
2) Interchange the initial and final states of FA
3) Reverse the directions of all the transitions.
4) Construct the regular grammar from this FA.

Example :- Convert the given left linear grammar to right linear grammar.

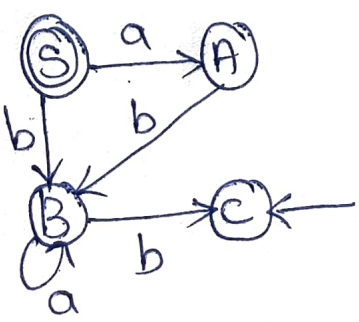
$S \rightarrow Aa \mid Bb$
$A \rightarrow Bb$
$B \rightarrow Ba \mid b$

Sol :- Let G be the left linear grammar.

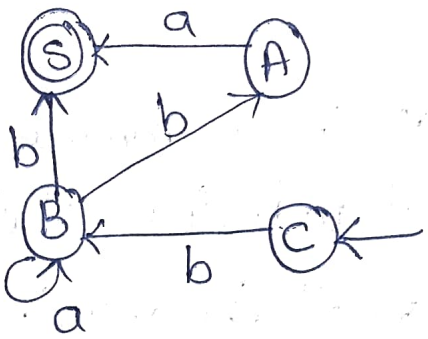
Step1 :- Convert the given left linear grammar to FA.



Step2 :- Interchange the initial and final states of FA



Step3 :- Reverse the directions of all the transitions



Step4 :- Construct regular grammar from this FA

$C \rightarrow bB$
$B \rightarrow aB \mid bA \mid b$
$A \rightarrow a$
$S \rightarrow \varepsilon$

} required RLG

Homework problem :-

$S \rightarrow Ca \mid Aa \mid Bb$
$A \rightarrow Ab \mid Ca \mid Bb \mid a$
$B \rightarrow Bb \mid b$
$C \rightarrow Aa$

K. Jaswanth