Chapter-3 : Regular Languages and Regular Grammars

3.1 Regular Expressions.

3.2 Applications of Regular Expression**:**

3.3 Connection between Regular Expressions and Regular Languages.

3.4 Regular Grammars **.**

3.4.1 Right Linear Grammar

3.4.2 Left Linear Grammar

3.4.3 Regular Grammar and Finite Automata

3.1 Regular Expressions:

The language accepted by finite automata are easily described by simple expressions called **regular** expressions.

The regular expression is the most effective way to represent any language.

Problems:

1) Write the regular expression for the language accepting all combinations of a's over the set ∑={a}

Solution : All combinations of a's means a may be single, double, tripple and so on. There may be the case that a is appearing for zero times, which means a null string. That is we expect the set of $\{\varepsilon, a, aa, aaa, \ldots\}$. So we can give regular expression for this as

$$R = a^*$$

That is kleen closure of a

2) Write the regular expression for the language accepting all combinations of a's except the null string over the set ∑={a}

Solution : The regular expression has to be built for the language

$$L = \{a, aa, aaa, \ldots\}$$

This set indicates that there is no null string. So we can denote r.e. as

$$R = a^+$$

As we know, positive closure indicates the set of strings without a null string.

3) Design the regular expression for the language accepting all the strings containing any number of a's and b's over the set ∑={a,b}

Solution : The regular expression will be

$$\text{r.e.} = (a + b)^*$$

This will give the set as L = {ε, *a*, *aa*, *a*b, b, b*a*, b*a*b, *a*b*a*b, ... any combination of *a* and b} .

The (*a* + b)* means any combination with *a* and b even a null string.

3.2 Applications of Regular Expression:

1. <u>Text editors</u> **:** Text editors are some programs which are used for processing the text. For example UNIX text editor uses the regular expression for substituting the strings, such as

$/bbb*/b/

substitutes a single blank for the first string of two or more blanks found in a given line.

2. <u>Lexical Analyzers:</u> Compiler uses a lexical analyzer to scan the input program and separate out the tokens. For example, Identifier is a category of token in the source language and it can be identified by a regular expression as shown below.

(letter)(letter+digit)* where letter={A,B,…,Z,a,b,…z} and digit ={0,1,…,9}

If anything in the source language matches with this regular expression then it is recognized as an identifier.

3.3 Connection between Regular Expressions and Regular Languages:

Regular Language:

The language accepted by some regular expression is known as a regular language-

.

For every regular language there is a regular expression, and for every regular expression there is a regular language

If r and s are two regular expressions denoting the Languages L1 and L2 respectively ,then

r+s is equivalent to L1 U L2 i.e union.

r.s is equivalent to L1 .L2 i.e concatenation.

r* is equivalent to L1* i.e closure.

Thoerm:

Let r be a regular expression,then there exists a NFA accepts the L(r),consequently L(r) is a regular Language.

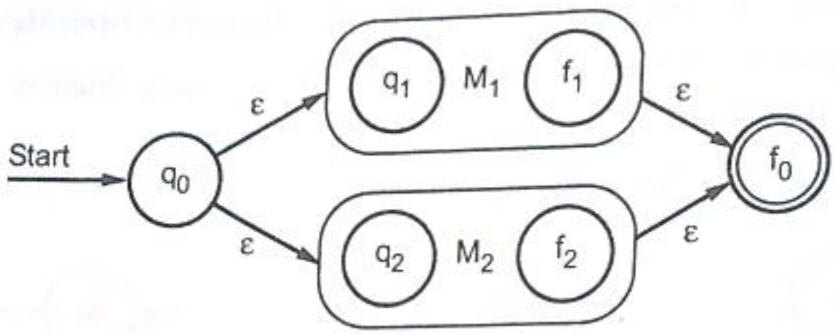
Case 1 : Union case

Let $r = r_1 + r_2$ where r_1 and r_2 be the regular expressions.

There exists two NFA's $M_1 = (Q_1, \Sigma_1, \delta_1, \{f_1\})$

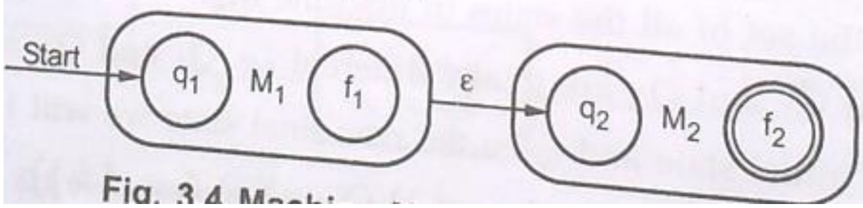
and $M_2 = (Q_2, \Sigma_2, \delta_2, \{f_2\})$

$L(M_1) = L(r_1)$ means the language states by regular expression r_1 is same which is represented by M_1. Similarly $L(M_2) = L(r_2)$.



Case 2 : Concatenation case

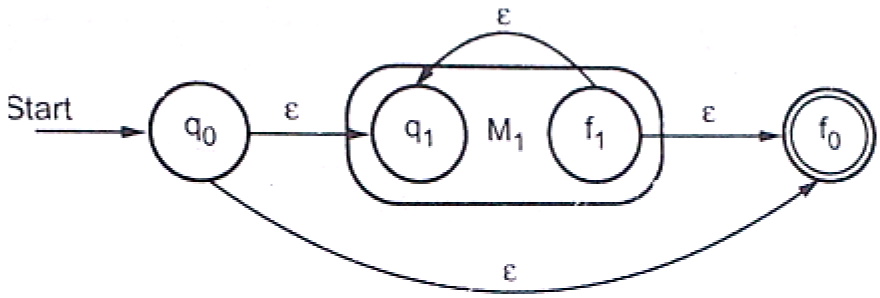
Let, $r = r_1 r_2$ where r_1 and r_2 are two regular expressions. The M_1 and M_2 denotes the two machines such that $L(M_1) = L(r_1)$ and $L(M_2) = L(r_2)$.



Case 3 : Closure case

Let $r = r_1^*$ where r_1 be a regular expression.

The machine M_1 is such that $L(M_1) = L(r_1)$.



Problems:

1) , **Construct NFA for the Regular Expression b+ba*.**

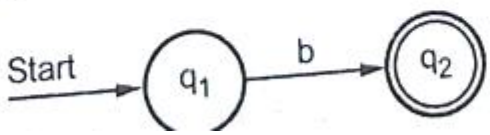
Solution : The regular expression

$r = b + ba^*$ can be broken into r_1 and r_2 as

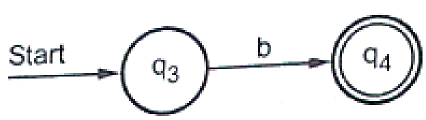
$r_1 = b$

$r_2 = ba^*$

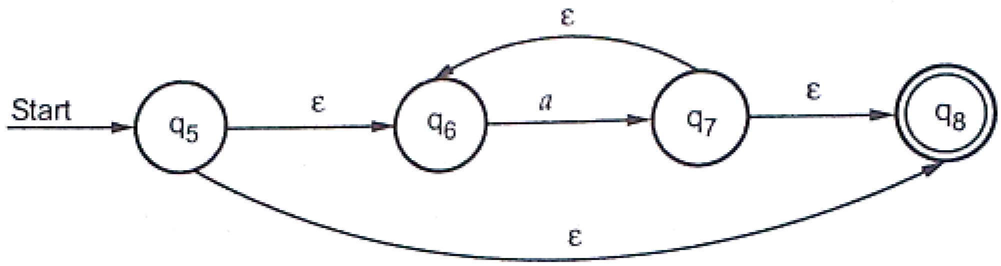
Let us draw the NFA for r_1, which is very simple.



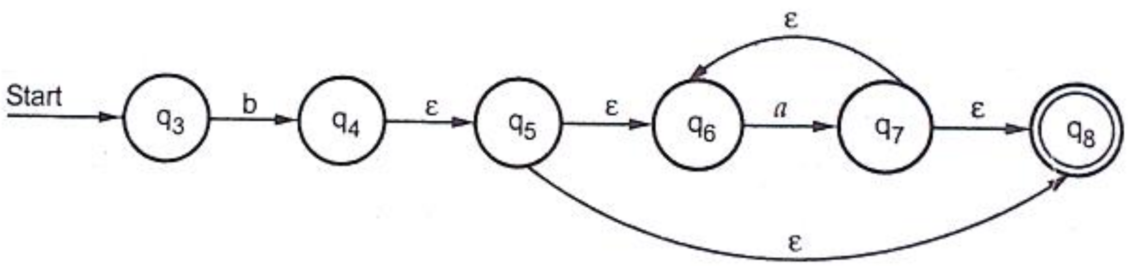
Now, we will go for $r_2 = ba^*$, this can be broken into r_3 and r_4 where $r_3 = b$ and $r_4 = a^*$. Now the case for concatenation will be applied. The NFA will look like this r_3 will be shown in Fig. 3.7.



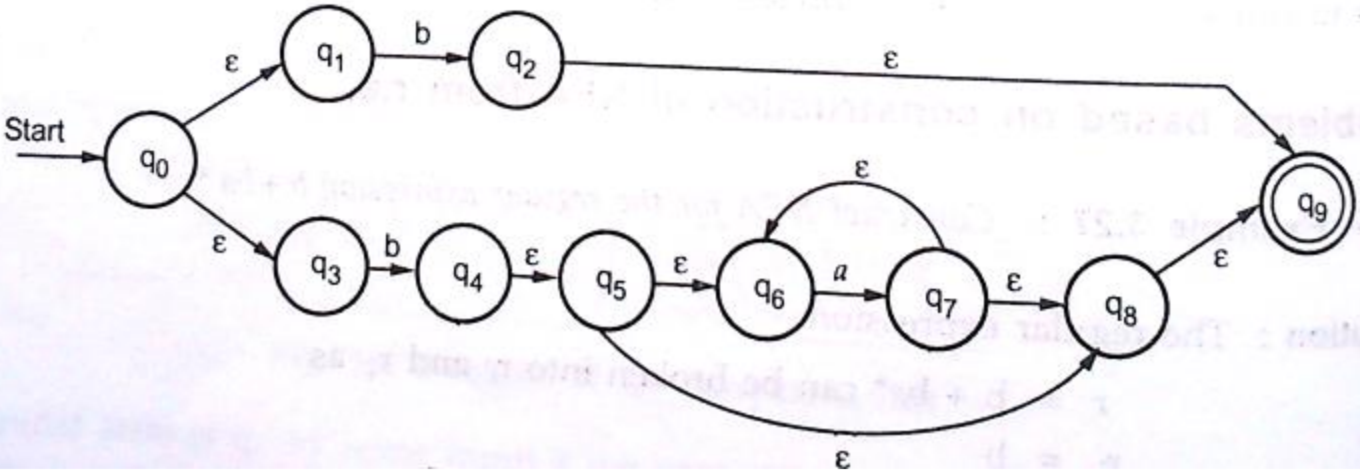
and r_4 will be shown as



The r_2 will be $r_2 = r_3 \cdot r_4$



Now, we will draw NFA for $r = r_1 + r_2$ i.e. $b + ba^*$



2) Construct NFA with € moves for the regular expression (0+1)*.

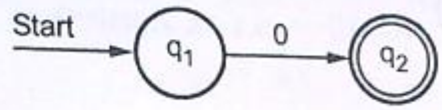
Solution:

$$r_3 = (r_1 + r_2)$$

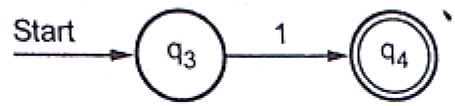
$$r = r_3^*$$

where $r_1 = 0$, $r_2 = 1$

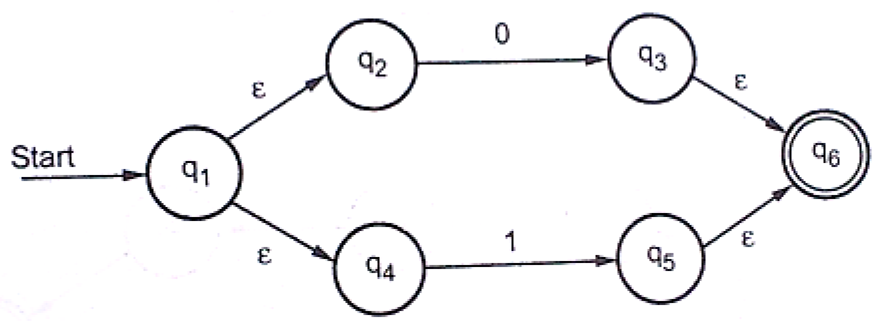
NFA for r_1 will be



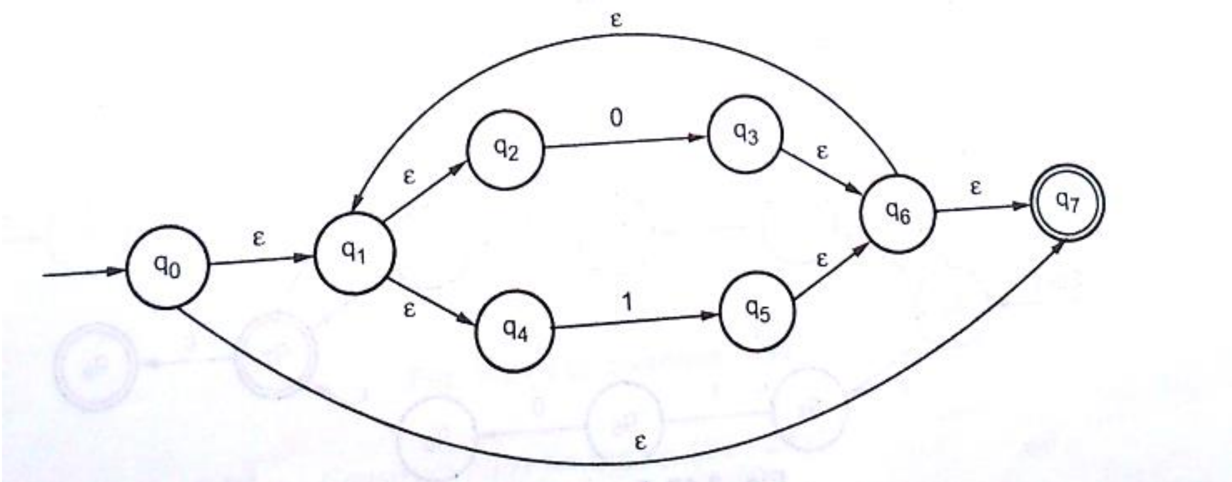
NFA for r_2 will be



NFA for r_3 will be



And finally



3.4 Regular Grammars :

A regular grammar is defined as G = (V, T, P, S) where

V is set of symbols called non terminals

T is a set of symbols called terminals.

P is a set of production rules.

S is a start symbol

The production rules P are of the form.

$$A \rightarrow aB$$

$$A \rightarrow a$$

Where A and B are non-terminal symbols and a is terminal symbol.

For example

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

$V = \{S, A\}$

$T = \{0, 1\}$

S is a start symbol and production rules are as given below -

$S \rightarrow 0S$

$S \rightarrow 1B$

$B \rightarrow \varepsilon$

Right-Linear and Left-Linear Grammar:

3.4.1 Right-Linear Grammar:

If the non terminal symbol appears as a rightmost symbol in each production of regular grammar then it is called right linear grammar.

The right linear grammar is of following form

$A \rightarrow aB$

$A \rightarrow a$

$A \rightarrow \varepsilon$

Where A and B are non-terminal symbols and a is a terminal symbol.

3.4.2 Left-Linear Grammar:

If the non terminal symbol appears as a left most symbol in each production of regular grammar then it is called left linear grammar.

The left- linear grammar is of following form:

$$A \rightarrow Ba$$
$$A \rightarrow a$$
$$A \rightarrow \varepsilon$$

Where A and B are non-terminal symbols and a is a terminal symbol.

A Regular Grammar is one that is either Right-linear or left-linear Grammar.

Example 3.12 The grammar $G_1 = (\{S\}, \{a, b\}, S, P_1)$, with P_1 given as

$$S \rightarrow abS|a$$

is right-linear. The grammar $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$, with productions

$$S \rightarrow S_1ab,$$
$$S_1 \rightarrow S_1ab|S_2,$$
$$S_2 \rightarrow a,$$

is left-linear. Both G_1 and G_2 are regular grammars.

The sequence

$$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababa$$

is a derivation with G_1. From this single instance it is easy to conjecture that $L(G_1)$ is the language denoted by the regular expression $r = (ab)^* a$. In a similar way, we can see that $L(G_2)$ is the regular language $L(aab(ab)^*)$. ■

Example 3.13 The grammar $G = (\{S, A, B\}, \{a, b\}, S, P)$ with productions

$$S \rightarrow A,$$
$$A \rightarrow aB|\lambda,$$
$$B \rightarrow Ab,$$

is not regular. Although every production is either in right-linear or left-linear form, the grammar itself is neither right-linear nor left-linear, and

3.4.3 Regular Grammar and Finite Automata:

Let $M = (\{q_0, q_1, \ldots, q_n\}\ \Sigma, \delta, q_0, F)$ be a DFA. The equivalent grammar G can be constructed from this DFA such that productions should correspond to transitions. The derivations can be terminated by a production rule giving terminals. For such production rule, the transitions terminating at some final state is encountered.

Let,

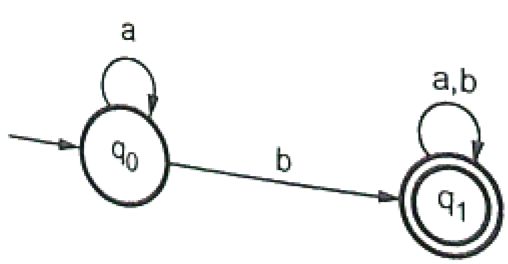
$$G = (\{A_0, A_1, \ldots A_n\}, \Sigma, P, A_0)$$

Where P the set of production rules can be defined by following rules.

1. $A_i \rightarrow a A_j$ is a production rule if $\delta(q_i, a) = q_j$, where $q_j \notin F$
2. $A_i \rightarrow a A_j$ and $A_i \rightarrow a$ are production rules if $\delta(q_i, a) = a_j$ where $q_j \in F$.

Thus the given grammar is accepted by DFA M.

1) Construct Regular Grammar for given Finite Automata.



Solution:

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

$$V = \{A_0, A_1\}$$

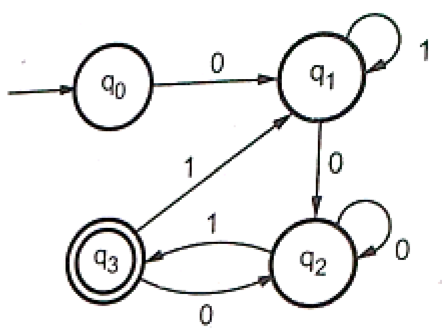
$$T = \{a, b\}$$

$A_0 \rightarrow a A_0 \qquad A_1 \rightarrow a A_1 | a$

$A_0 \rightarrow b A_1 \qquad A_1 \rightarrow b A_1 | b$

$A_0 \rightarrow b$

2) Construct Regular Grammar for the diagram as given below



Solution : The equivalent regular grammar can be denoted by G = (V, T, P, S) wher

$$V = \{A_0, A_1, A_2, A_3\}$$

The production rules can be

$A_0 \rightarrow 0 A_1$

$A_1 \rightarrow 1 A_1$

$A_1 \rightarrow 0 A_2$

$A_2 \rightarrow 0 A_2$

$A_2 \rightarrow 1 A_3$

$A_2 \rightarrow 1$

$A_3 \rightarrow 1 A_1$

$A_3 \rightarrow 0 A_2$