TM = Turing machine

LBA = Linear Bounded Automaton

PDA = Pushdown Automation

FA = Finite Automaton

□□□

CHAPTER 3

Regular Set and Regular Grammar

University Prescribed Syllabus

Regular Sets and Regular Grammar : Regular Grammar, Regular Expressions, Finite automata and Regular Expressions, Pumping Lemma and its Applications, Closure Properties, Regular Sets and Regular Grammar.

Syllabus Topic : Regular Grammar

3.1 Regular Grammar

- A regular grammar is a mathematical object,
- Grammar having four tuples, G = (N, Σ, P, S), where
  N is a nonempty, finite set of non-terminal symbols,
- Σ is a finite set of terminal symbols , or alphabet, symbols,
- P is a set of grammar rules,
- S is the start symbol. Where S ∈ N (S belong to N)
- Every regular grammar describes a regular language. A **regular grammar** is formal grammar that is right-regular or left-regular.
- A **right regular grammar** is also called **right linear grammar** such that all the production rules in P are in following forms:

  $A \rightarrow a$

  $A \rightarrow aB$

  $A \rightarrow \varepsilon$.

where A and B are non-terminals in N and a is in Σ and ε denotes the empty string

- **Example :**

  $S \rightarrow abS$, $S \rightarrow b$

  In this example, S is nonterminal and a,b are terminals

- A left regular grammar is also called left linear grammar , such that all the production rules in P are in following forms:

  $A \rightarrow a$

  $A \rightarrow Ba$

  $A \rightarrow \varepsilon$.

  where A and B are non-terminals in N and a is in Σ and ε denotes the empty string [note :empty string means string of length is 0]

- For Example :

  $S \rightarrow Sbb$, $S \rightarrow b$

- In this example, S is non terminal and b is terminal.

Syllabus Topic : Regular Expressions

3.2 Regular Expression

- Regular expression is useful for representing the set of strings. Regular expression describes the languages accepted by finite automata.

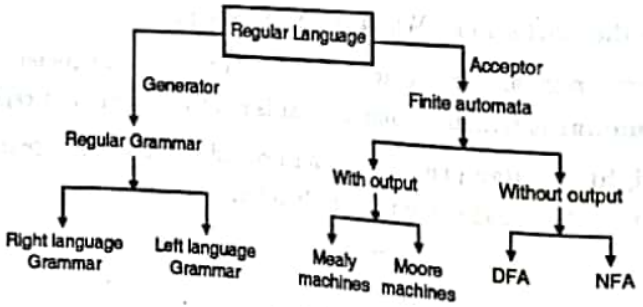


Fig. 3.2.1

- Regular expression contains **three** operations :
  1. **union (+)**
  2. **concatenation (.)**
  3. **Iteration on kleene closure (*)**

3.2.1 Primitive Expression

☞ **Write a primitive expression ?**

- ϕ and ε are an empty set. These are both regular expression .

  ϕ = regular expression represents { }

  ε = regular expression represents {ε}

  $a \in \Sigma$ (a belongs to Σ) is also regular expression.

  $a \in \Sigma$ = represents {a}.

- These regular expressions are called as **primitive expression.**
- If we take any primitive expressions r_1 and r_2 and apply all operations.

1. **union operation**

   $r_1 + r_2$

2. **concatenation operation**

   $r_1 \cdot r_2$

3. **kleene closure operation**

   r_1^*

3.2.2 Rules for Regular Expression

☞ **Write a rules for regular expression ?**

- Regular expressions are used for representing certain sets of strings in an algebraic fashion.
- **Rules** for Regular expression.

1. Any **terminal symbol** i.e. symbols belongs to Σ including empty (ε) and Null (ϕ) are regular expression.

e.g. a, b, ... A, Q – are regular expression

2. The **union of two regular expressions** is also regular expression.

   Let consider R_1 and R_2 are regular expression, then $R_1 \cup R_2$ or $(R_1 + R_2)$ are regular expression.

3. The **concatenation of two regular** expression is also regular expression.

   Let consider R1 and R2 are regular expression

   $R_1 \cdot R_2$ ---- are R.E.

   Concatenation denoted by (·)

4. The **iteration or closure of regular expression** is also a regular expression.

   $R \rightarrow R^*$

   e.g. $a^* = \varepsilon, a, aa, aaa.....$

5. The RE over Σ are precisely those obtained recursively by application of above rules once or several times.

For example :

Describe the following sets as regular expression.

1. {0,1,2}

   R = 0 + 1 + 2 [+ symbol denotes or]

2. {ε, ab}

   R = ε ab [here we have not used + symbol]

3. {abb, a, b, bba}

   R = abb + a + b + bba

4. { ε, 0, 00, 000}

   R = 0* closure of 0

5. {1,11,111,1111...}

   R = 1+

3.2.3 Regular Set

What is a Regular set ?

Any set represented by regular expression is called **regular set.**

For example,

- a, b $\in \Sigma$ then,
  - (i) a denotes the set {a}
  - (ii) a + b denotes set {a, b}
  - (iii) ab denotes {a, b}
  - (iv) a* denotes set {ε, a, aa, aaa ---}
  - (v) (a + b)* denotes {a b}*
- The set is represented by R.
- Language is denoted by L(R)
- These are three basic operations
  - (i) union (+)
  - (ii) concatenation (·)
  - (iii) iteration or closure (*)

For Example, describe the following sets by regular expression.

1. L_1 = set of all stirngs of 0's and 1's ending in 00.

   $\Rightarrow L_1$ is obtained by concatenary any string over {0,1} and 00.

   {0,1} is represented by 0 + 1

   $$L_1 = (0 + 1)^* 00$$

2. L_2 = set of all strings of 0's and is beginning with 0 and ending with 1.

   $\Rightarrow L_2$ is obtained by concatenating 0, anything over {0,1} and 1.

   $$L_2 = 0 (0 + 1)^* 1$$

3. L_3 = {ε, 11, 111, 1111, ...}

   $\Rightarrow L_3$ is either ε or string of even number of 1's. i.e. string is in of $(11)^n$ where $n \geq 0$.

$L_2 = (11)^*$

4. Lets consider $\Sigma = (a, b)$

L_1 = Set of all strings which are having length **exactly two**. Find regular expression.

$\Rightarrow L_1 = \{aa, ab, ba, bb\}$

$\therefore$ R.E. $= aa + ab + ba + bb = a(a + b) + b(a + b)$

$= (a + b)(a + b)$

5. Find regular expression, which represents set of all string which are having length **at least two.**

$\Rightarrow L_1 = \{aa, ab, ba, bb, aaa.....\}$

Language is infinite language.

$(a + b) \cdot (a + b)(a + b)^*$

6. Find regular expression which represents set of all strings which are having length **at most 2** (at most means length is 0, 1, 2).

$\Rightarrow L_1 = \{\varepsilon, a, b, aa, ab, ba, bb\}$

$\therefore (a + b + \varepsilon) \quad (a + b + \varepsilon)$

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

Even length strings

$L = \{\varepsilon, aa, ab, ba, bb, ...\}$

(∵length is 0, 2, 4, 6....)

R.E. $= ((a + b)(a + b))^*$

8. **Odd length** strings

(length is 1,3,....)

$\Rightarrow ((a + b)(a + b))^* a + b$

9. Length is divisible by '3'

Language contains length 0,3,9,12...

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

10. Number of a's have to be **exactly 2**

$\Rightarrow b^*ab^*ab^*$

11. **a's at least 2**

$\Rightarrow b^*ab^*a\,(a + b)^*$

12. **a's at most 2**

$\Rightarrow b^*\,(\varepsilon + a)\,b^*\,(\varepsilon + a)b^*$

13. Number of a's are **even**

$\Rightarrow (b^*ab^*ab^*)^* + b^*$ or $(b^*a\,b^*ab^*)\cdot b^*$

14. **Starts** with a's

$\Rightarrow a(a + b)^*$

15. **Ends** with a's

$\Rightarrow (a + b)^*a$

16. **Containing** a's

$\Rightarrow (a + b)^*\,a\,(a + b)^*$

17. **Starting** and **ending** with **different symbols**

$\Rightarrow a\,(a + b)^*\,b$ or $b\,(a + b)^*\,a$

18. Starting and ending with **same symbols**

L is $\{\varepsilon, a, b, aa, bb.....\}$

$\Rightarrow a\,(a + b)^*\,a + b\,(a + b)^*\,b + a + b + \varepsilon$

19. Not a's come together (two aa's **should not come together**)

$\Rightarrow$ L is $\{\varepsilon, b, bb, bbb, a, ab, aba\ abab, ababa, ba, bab, baba, babab.....\}$

$(b + ab)^*\,(\varepsilon + a)$

3.2.4 Identities of Regular Expression

Write a identities of regular expression

1. $\phi + R = R$ .... ϕ = Empty set union of ϕ and R where R is Regular expression.
2. $\phi R + R\phi = \phi$
   ϕ concatenation R Union R concatenation $\phi = \phi$
3. $\varepsilon R = R\varepsilon = R$.
4. $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$
5. $R + R = R$
   RE union RE = RE
6. $R^* R^* = R^*$
   Concatenation of closure of two regular expressions = closure of Regular expression.
7. $RR^* = R^*R$
8. $(R^*)^* = R^*$
9. $\varepsilon + RR^* = \varepsilon + R^*R = R^*$
10. $(PQ)^*P = P(QP)^*$
11. $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
12. $(P + Q) R = PR + QR$ and
    $R(P + Q) = RP + RQ$

3.2.5 Arden's Theorem

If P and Q are two regular expressions over Σ and if P does not contain ε, then the following equation in R given by $R = Q + RP$ has a unique solution i.e $R = QP^*$

Proof

$$R = Q + RP \quad \ldots(3.2.1)$$

Replace $R = QP^*$,

$$= Q + QP^* P$$

$$= Q(\varepsilon + P^* P) \quad \ldots\text{Identity } [\varepsilon + R^*R = R^*]$$
$$= QP^*$$

Prove that $R = QP^*$ is a solution.

∴To prove that this is a unique solution.

$$R = Q + RP \quad [\text{Replace R with } Q + RP]$$
$$= Q + [Q + RP] P$$
$$= Q + QP + RP^2 \quad [\text{R is a replaced by } Q + RP]$$
$$= Q + QP + [Q + RP] P^2$$
$$= Q + QP + QP^2 + RP^3$$
$$\vdots$$
$$= Q + QP + QP^2 + \ldots + QP^n + RP^{n+1}$$

Replace $R = QP^*$

$$= Q + QP + QP^2 + \ldots + QP^n + QP^* P^{n+1}$$
$$= Q [\varepsilon + P + P^2 + \ldots + P^n + P^* P^{n+1}]$$
$$R = QP^*$$

This is a unique solution.

Example proofs using Identities of Regular expression

Example 3.2.1 : Prove that $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1)$ is equal to $0^*1(0 + 1^* 1)^*$

Solution :

$$\text{LHS} = (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1)$$
$$= (1 + 00^*1)[\varepsilon + (0 + 10^*1)^* (0 + 10^*1)] \quad \ldots \varepsilon + R^* R = R^*$$
$$= (1 + 00^*1)(0 + 10^*1)^*$$
$$= (\varepsilon \cdot 1 + 00^*1)(0 + 10^*1)^* \quad \ldots \varepsilon \cdot R = R$$
$$= (\varepsilon + 00^*) 1 (0 + 10^*1)^*$$
$$= 0^*1 (0 + 10^* 1)^*$$
$$= \text{R.H.S}$$

Example 3.2.2 : Give an Regular expression for representing the set L of strings in which every 0 is immediately followed by at least two 1's.

Solution. :

String does not contain only zero, string contains 0 preceded by 1 and followed by 11 ;

so $L = (1 + 011)^*$

Example 3.2.3 : Prove that regular expression,
$R = \varepsilon + 1^* (011)^* (1^*(011)^*)^*$
also describes the same set strings in Example 3.1.2.

Solution. :

$$R = \varepsilon + P^1P_1^*$$

where $P_1 = 1^* (011)^*$

$R = P_1^*$ ...using identity $\varepsilon + RR^* = R^*$

$= (1^*(011)^*)^*$

Lets consider $P_2 = 1$ and $P_3 = 011$

$= \left(P_2^*P_3^*\right)^*$

$= (P_2 + P_3)^*$ by using identity $\left(P^* + Q^*\right)^* = \left(P^* + Q^*\right)^*$

$R = (1 + 011)^*$

Syllabus Topic : Finite Automata and Regular Expression

3.3 Finite Automata and Regular Expression

In this topic, we will study the representation of Regular expression.

3.3.1 Transition System Containing ε Moves

Write a steps for converting transition system with ε moves into transition system without ε moves ? Explain it with example

- ε-transition or ε moves which are associated with null symbol.
- Transition system can be generalized by permitting ε moves.
- These transition can occur when no input is applied.

- We can convert transition system with ε moves into transition system without ε moves.

Steps for converting transition system with ε moves into transition system without ε moves

1. First find out all ε transitions from each state from Q, that is called as ε-closure (q_i) when $q_i \in Q$.

Note : Every state on ε goes to itself.

2. The δ' transition can be obtained. The δ' transition means an ε-closure on δ moves.
3. Step 2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ε moves can be built.

Note : ε* closure- All states that can be reached from a particular state only by seeing ε b symbol.

Example 3.3.1 : Converting ε NFA to its equivalent NFA.

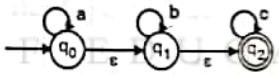


Fig. P. 3.3.1(a)

Solution :

This is for q_0 state

For a input

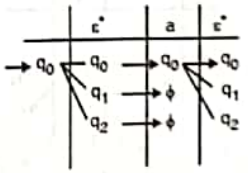


Fig. P. 3.3.1(b)

For b input

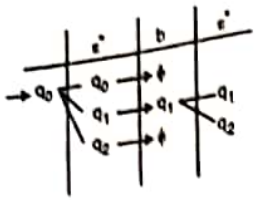


Fig. P. 3.3.1(c)

For c input

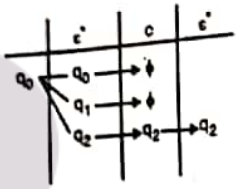


Fig. P. 3.3.1(d)

Now find out for q_1 state,

For a input

Fig. P. 3.3.1(e)

For b input

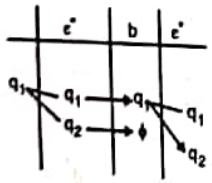


Fig. P. 3.3.1(f)

For c input

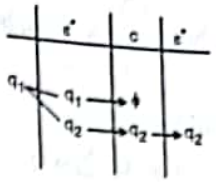


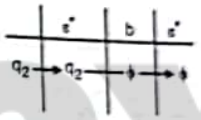
Fig. P. 3.3.1(g)

Now find out for q_2 state,

For a input

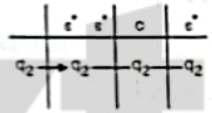
(h)

For b input



(i)

For c input



(j)

Fig. P. 3.3.1

State transition table

	a	b	c
→ q_0	(q_0,q_1,q_2)	(q_1,q_2)	q_2
q_1	ϕ	(q_1,q_2)	q_2
* q_2	ϕ	ϕ	q_2

Fig. P. 3.3.1(k)

Transition diagram

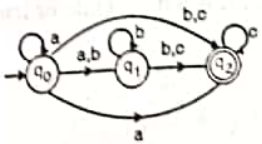


Fig. P. 3.3.1(l)

3.3.2 Conversion of NFA to DFA

- Every DFA is an NFA, but not viceversa, but there is an equivalent DFA for every NFA.
- Transition function of DFA is,

$$\delta = Q \times \Sigma \rightarrow Q$$

- Transition function of NFA is,

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

- We can convert every NFA into DFA.

For Example

L = {Set of all strings over (0, 1) that states with '0'}.

Σ = {0, 1}

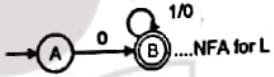


Fig. 3.3.1

State transition diagram

State s	0	1
A	B	ϕ
B	B	B

Convert the NFA into DFA

Transition State

	0	1	
A	B	C	... C is a dead state or trap state
B	B	B	
C	C	C	

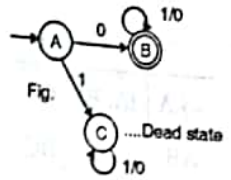


Fig.

Fig. 3.3.2

3.3.3 Conversion of DFA to NFA

Find the equivalent DFA for the NFA given by L = [{A, B, C}, (a, b), δ, A, {C}]

Which δ is given by :

	a	b
→ A	A, B	C
B	A	B
©	–	A, B

- **States :** A, B, C
- Inputs : a, b
- Transits function :δ
- Initial state : A
- Final state : C

State diagram for NFA

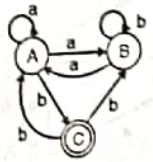


Fig. 3.3.3

Convert NFA into DFA

	a	b
→ A	{A, B}	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D

Note : To find AB states, we must look at transition state diagram of NFA and apply union operation.

In DFA we can not leave a state, so we have to write new state. In our example we have written D is a new state which is called dead state.

The input that comes in dead state remains there.

To find state of DFA

- Check the final state of NFA and after that select the state from DFA that contains the final state of NFA.
- In our example, 'C' state is a final state in NFA. So, select state from DFA which contains 'C' state in DFA. So we get 2 final states in our example.

State diagram for DFA

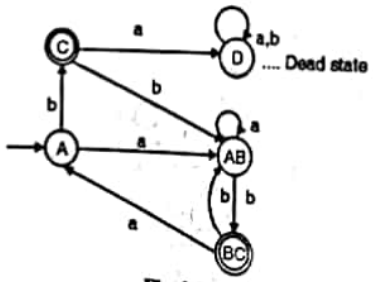


Fig. 3.3.4

Example 3.3.2 : Given below is the NFA for a language,

L = { set of all strings over (0, 1) that ends with '01' }

Construct its equivalent DFA

Solution :

⇒ First draw NFA

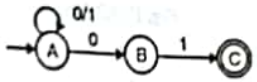


Fig. P. 3.3.2(a)

Transition table

	0	1
→ A	A, B	A
B	ϕ	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table for DFA

	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A

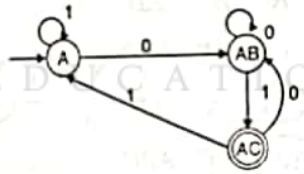


Fig. P. 3.3.2(b)

Example 3.3.3 : Design an NFA for language that accepts all strings over {0, 1} in which the second last symbol is always '1'. Then convert it to its equivalent DFA.

e.g. string 1 0 1 0 or 1 1 0 or 1 1 0 1 0 1 0

Solution. :

Design NFA

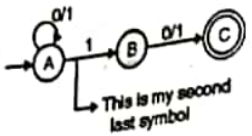


Fig. P. 3.3.3(a)

State transition table for NFA

	0	1
→ A	A	A, B
B	C	C
(C)	ϕ	ϕ

Convert into DFA

Transition table

	0	1
→ A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

Fig. P. 3.3.3(b)

3.3.4 Algebraic Methods Using Arden's Theorem

We consider following assumption regarding the transition system,

1. The transition graph or diagram does not have ε moves,
2. It has only one initial state say V_1,
3. Its vertices are $v_1, v_2 \ldots v_n$.
4. α_{ij} denotes regular expression representing the set of labels of edges from V_i to V_j.

$$V_1 = v_1\alpha_{11} + v_2\alpha_{21} \ldots + v_n\alpha_n + A$$
$$V_2 = v_1\alpha_{12} + v_2\alpha_{22} \ldots + v_n\alpha_{n2}$$
$$\vdots$$
$$V_n = v_1\alpha_{1n} + v_2\alpha_{2n} \ldots + v_n\alpha_{nn}$$

by repeatedly applying substitutions and applying Arden's theorem we can express V_i in terms at α_{ij}'s

Example 3.3.4 : Consider the transition system given in Fig.**, Prove that the strings recognized are :

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

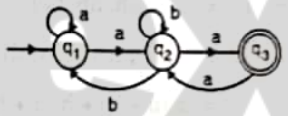


Fig. P. 3.3.4

Solution :

There is no any ε move and there is only one initial state.

3 equations for q_1, q_2, q_3 can be written as,

$$q_1 = q_1 a + q_2 b + \varepsilon \quad \ldots(1)$$
$$q_2 = q_1 a + q_2 b + q_3 a \quad \ldots(2)$$
$$q_3 = q_2 a \quad \ldots(3)$$

by substituting Equation (3) in Equation(2) we get,

$$q_2 = q_1 a + q_2 b + q_2 aa$$
$$= q_1 a + q_2 (b + aa)$$
$$= q_1 a\,(b + aa)^* \quad \ldots(4)$$

by substituting Equation (4) in Equation (1) we get,

$$q_1 = q_1 a + q_1 a\,(b + aa)^*\,b + \varepsilon$$
$$= q_1 (a + a\,(b + aa)^*b) + \varepsilon$$

Hence,

$$q_1 = \varepsilon (a + a (b + aa)^* b)^*$$

$$q_2 = (a + a (b + aa)^* b)^* a (b + aa)^*$$

$$q_3 = (a + a (b + aa)^* b)^* a (b + aa)^* a$$

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

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

3.3.5 Designing Regular Expressions

For example, design regular expression for the following language over {a, b}

1. L = accepting strings of length exactly 2

$$\Rightarrow L_1 = \{aa, ab, ab, bb\}$$

$$R = aa + ab + ba + bb$$

$$= a (a + b) + b (a + b)$$

$$= (a + b) (a + b)$$

2. L = accepting string of length at least 2

$$\Rightarrow L_1 = \{aa, ab, ba, bb, aaa \ldots.\}$$

$$R = (a + b) (a + b) (a + b)^*$$

3. L = accepting strings of length atmost 2

$$\Rightarrow L_1 = \{\varepsilon, a, b, aa, ab, ba, bb\}$$

$$R = \varepsilon + a + b + aa + ab + ba + bb$$

$$= (\varepsilon + a + b) (\varepsilon + a + b)$$

For example,

1. **Find the regular expression for the following NFA.**

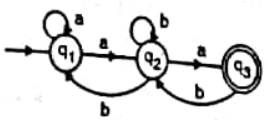


Fig. 3.3.5

We have written equation for each states of NFA and put all Equations of all states to the equation of final state and that gives regular expression for given NFA.

For writing equation, check incoming input transition.

$$q_3 = q_2 a \quad \ldots(3.3.1)$$

$$q_2 = q_1 a + q_2 b + q_3 b \quad \ldots(3.3.2)$$

$$q_1 = \varepsilon + q_1 a + q_2 b \quad \ldots(3.3.3)$$

Simplify these Equation,

$$q_3 = q_2 a$$

$$= (q_1 a + q_2 b + q_3 b) a$$

$$= q_1 aa + q_2 ba + q_3 ba \quad \ldots(3.3.4)$$

$q_2 = q_1 a + q_2 b + q_3 b$...Putting value of q_3 from Equation (3.3.1)

$$= q_1 a + q_2 b + (q_2 a) b$$

$$= q_1 a + q_2 b + q_2 ab$$

$$\underbrace{q_2}_{R} = \underbrace{q_1 a}_{Q} + \underbrace{q_2}_{R} \underbrace{(b + ab)}_{P}$$

...(Informed R = Q + RP R = QP* by Arden's theorem)

$$\therefore q_2 = (q_1 a) (b + ab)^* \quad \ldots(3.3.5)$$

$$q_1 = \varepsilon + q_1 a + q_2 b$$

Putting value of q_2 from Equation (3.3.5)

$$q_1 = \varepsilon + q_1 a + ((q_1 a) (b + ab)^*) b$$

$$\underbrace{q_1}_{R} = \underbrace{\varepsilon}_{Q} + \underbrace{q_1}_{R} \underbrace{((a + a) (b + ab)^*) b}_{P}$$

...Which is in form R = Q + RP, R = QP*

$$q_1 = \varepsilon ((a + a (b + ab)^*) b)^* \quad \ldots\text{By identities } \varepsilon \cdot R = R$$

$$q_1 = (a + a (b + ab)^* b)^* \quad \ldots(3.3.6)$$

Final state q_3,

$$q_3 = q_2 a$$

$$= q_1 a (b + ab)^* a \quad \text{...Putting value of } q_2 \text{ from (3.3.5)}$$

$$= ((a + a (b + ab)^* b)^* a (b + ab)^* a \quad \text{...Putting value of } q_1 \text{ from (3.3.6)}$$

2. Find the regular expression for the following DFA

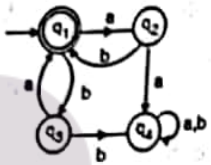


Fig. 3.3.6

Here q_1 is a final state

For writing equation check input transitions in each state

$$q_1 = \varepsilon + q_2 b + q_3 a \quad \text{...(3.3.7)}$$

$$q_2 = q_2 a \quad \text{...(3.3.8)}$$

$$q_3 = q_1 b \quad \text{...(3.3.9)}$$

$$q_4 = q_2 a + q_3 + b + q_4 a + q_4 b \quad \text{...(3.3.10)}$$

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

Putting values of q_2 and q_3 from Equations (3.3.7) and (3.3.9)

$$q_1 = \varepsilon + q_1 ab + q_1 ba$$

$$\underbrace{q_1}_{R} = \underbrace{\varepsilon}_{Q} + \underbrace{q_1}_{R} \underbrace{(ab + ba)}_{P} \quad \text{...In form of } R = Q + RP$$

$R = QP^*$ by Arden's theorem

$$q_1 = \varepsilon (ab + ba)^*$$

$$q_1 = (ab + ba)^* \quad \text{...}\varepsilon R = R$$

3.3.6 Construction of Finite Automata

Construction of finite automata equivalent to regular expression

This method we use, to construct a finite automation equivalent to a given regular expression is called subset method.

Some basic element :

(i) (a + b) means a union b.

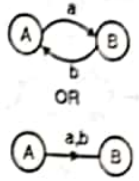


Fig. 3.3.7

(ii) (a · b) means a concatenation with b.

Fig. 3.3.8

(iii) a^* means closure of a

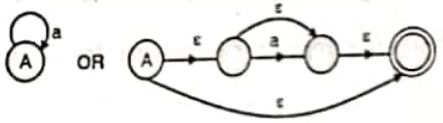


Fig. 3.3.9

(iv) ϕ

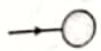


Fig. 3.3.10

(v) a

Fig. 3.3.11

(vi) ab^*

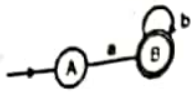


Fig. 3.3.12

(vii) $(ab)^*$

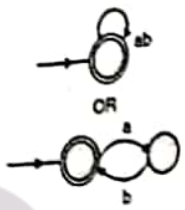


Fig. 3.3.13

(viii) $(ab + ba)^*$

Fig. 3.3.14

3.3.7 Conversion of Regular Expression to Finite Automata

Convert the following regular expressions to their equivalent finite automata.

For example

1. **ba^*b :**

String accepted like {bb, bab, baab...}

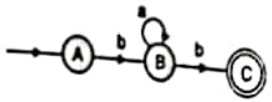


Fig. 3.3.15

2. **$(a + b)\,c$:**

In this case first is union operation followed by concatenation.

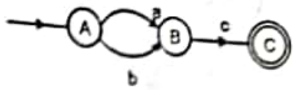


Fig. 3.3.16

String accepted by the R.E are {ac, bc}

3. **$a\,(bc)^*$**

String accepted by the R.R. are {a, abc, abcbc...}

Fig. 3.3.17

4. **$(a \mid b)^*\,(abb \mid a^*b)$**

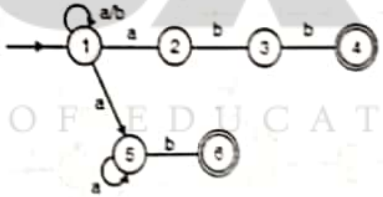


Fig. 3.3.18

$a^+ = \{a, aa, aaa....\}$

5. **$10 + (0 + 11)0^*1$**

Step I :

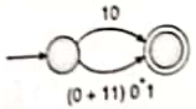


Fig. 3.3.19

Step II :

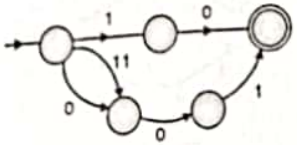


Fig. 3.3.20

Step III :

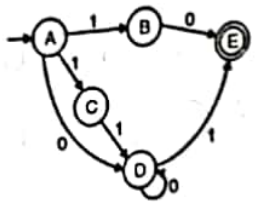
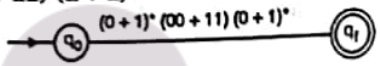


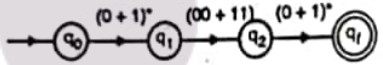
Fig. 3.3.21

6. $(0 + 1)^* (00 + 11) (a + 1)^*$

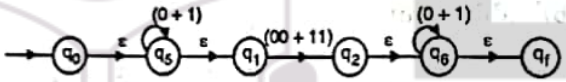
Step I :



Step II :



Step III :



Step IV :

Fig. 3.3.22

Step V :

Remove the ε moves.

Step VI :

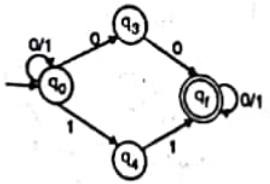


Fig. 3.3.23

Constructing a finite automaton equivalent to

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

Step VII : Construct transition table for NFA.

	0	1
$\longrightarrow q_0$	q_0, q_3	q_0, q_4
q_3	q_f	ϕ
q_4	ϕ	q_f
q_f	q_f	q_f

Step VIII : Construct transition table for DFA using transition table of NFA.

	0	1
$\longrightarrow q_0$	$\{q_0, q_3\}$	$\{q_0, q_4\}$
$\{q_0, q_3\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4\}$
$\{q_0, q_3, q_f\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4, q_f\}$
$\{q_0, q_4\}$	$\{q_0, q_3\}$	$\{q_0, q_3, q_f\}$
$\{q_0, q_4, q_f\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4, q_f\}$

Step IX : Construct transition diagram for DFA

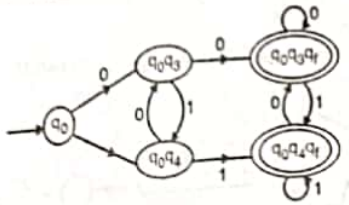


Fig. 3.3.24 : Finite automata

3.3.8 Conversion of F.A. to R.E.

State elimination method

Rules :

(i) Initial state could not have any incoming edge.

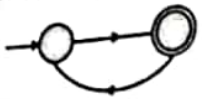


Fig. 3.3.25

If any incoming edge there, create new initial state with ∈ move.

Fig. 3.3.26

(ii) Final state should not have any outgoing edge.

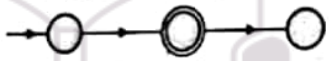


Fig. 3.3.27

If any outgoing edge of there, create new final state with ε move.

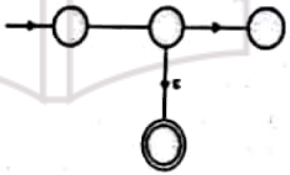


Fig. 3.3.28

If we have more than one final state then make it into one final state.

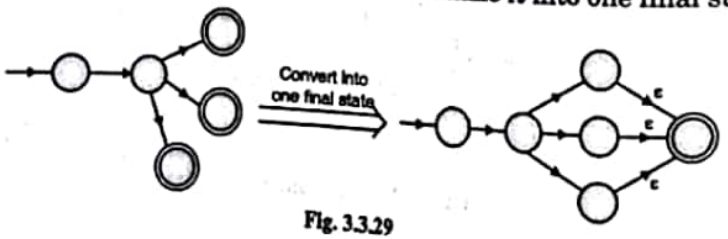


Fig. 3.3.29

(iii) Eliminate other state apart from initial and final state.

Example :

1.

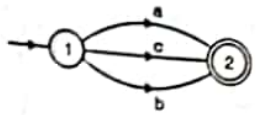


Fig. 3.3.30

StepI :

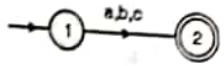


Fig. 3.3.31

R.E. = a + b + c

2.

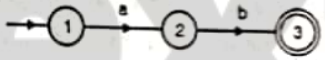


Fig. 3.3.32

Step I : Remove second (②) state.

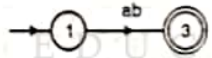


Fig. 3.3.33

RE = ab

3.3.9 Equivalence of Two Finite Automata

☞ **How to identify equivalence of two finite automata ?**

Steps to identify equivalence

1. For any pair of states $\{q_i, q_j\}$ the transition for input a $\in \Sigma$ is defined by $\{q_a, q_b\}$ where ;

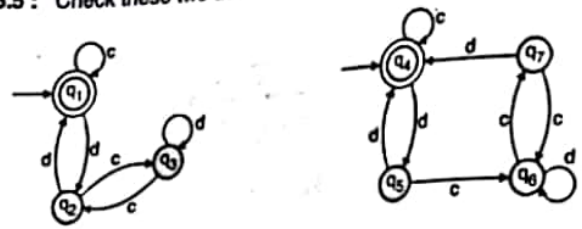
   $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$

   The two automates are not equivalent if for pair $\{q_a, q_b\}$ one is intermediate state and other is final state

2. If initial state is final state of one automation, then in second automation

also, initial state must be final state for them to be equivalent.

Example 3.3.5 : Check these two automata are equivalent or not.



(i) First Automata [A] (ii) Second Automata [B]

Fig. P. 3.3.5

Solution :

Check condition I :

- For every pair of states the pair of states generates particular input showable either both should be on intermediate state or both should be on final state.

States	Input	
	C	D
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_5) IS IS	(q_1, q_4) FS FS
(q_3, q_6)	(q_2, q_7) IS IS	(q_3, q_6) IS IS
(q_2, q_7)	(q_3, q_6) IS IS	(q1, q4) FS FS

Note FS = final state

IS = Intermediate state

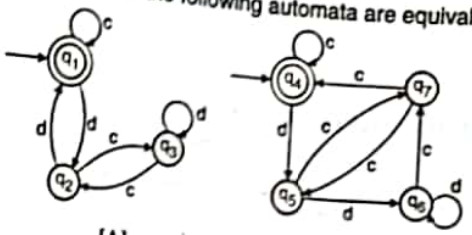
So first condition is satisfied.

Check condition II :

- Initial state and final state are same, in both automata.

- In our Example, q_1 is initial state as well as final state and in second automata q_4 is initial state as well as final state. So these two automata are equivalent.

Example 3.3.6 : Find out whether the following automata are equivalent or not :



[A] [B]

Fig. P. 3.3.6

Soln. :

In this example, second condition is satisfied because q_1 is initial and final state in A Automata and q_4 is initial state and final state in B Automata.

Check condition I :

	Inputs	
State	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS, IS
(q_2, q_5)	(q_3, q_7) IS, IS	(q_1, q_5) FS, IS

A and B are not equivalent.

Equivalence of two regular expressions

Write methods for checking equivalence of regular expression ?

There are two methods for checking equivalence of given regular expression,

1. Simplification of regular expression by using indentities.
2. Conversion of regular expression into finite automata.

Solved examples on Equivalence of two regular expressions

Example 3.3.7 : $(a + b)^* = a^* (ba^*)^*$

Check the equivalence of given regular expression by Converting of regular expression into finite automata.

Solution :

Method II

Step I : Construct finite automata for $(a + b)^*$

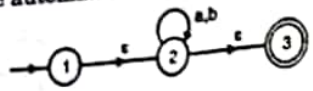
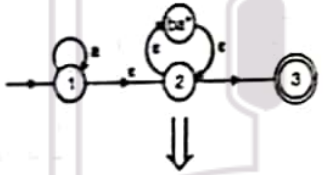


Fig. P. 3.3.7(a)

After removing ε moves we get the diagram,

Fig. P. 3.3.7(b)

Step II : Transition diagram for $a^* (ba^*)^*$



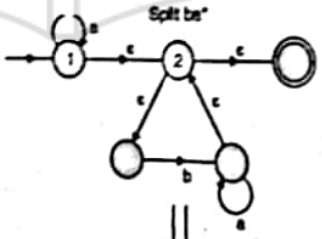


Fig. P. 3.3.7(c)

Hence prove that given regular expression are equal.

Syllabus Topic : Pumping Lemma and its Application

3.4 Pumping Lemma and its Application

Write a note on pumping lemma for regular expression ?

3.4.1 Pumping Lemma for Regular Expression

- Pumping lemma is used to prove that a language is NOT regular.
- It cannot be used to prove that a language is regular.
- If A is regular language, then A has a pumping length 'p' such that any string 's',

where $|s| \geq p$ may be divided into 3 parts.

$s = xyz$ such that the following conditions most be true-

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

(ii) $|y| > 0$

(ii) $|xy| \leq p$.

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

(i) Assume that A is regular.

(ii) It has to have a pumping length (say p).

(iii) All strings longer than p can be pumped $|s| \geq p$.

(iv) Now find a string 's' in A such that $|s| \geq p$.

(v) Divide s into x, y, z.

(vi) Show that $xy^i z \in A$ for some i.

(vii) Then consider all ways that s can be divided into xyz.

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

(ix) S cannot be pumped = contradiction means language is not regular.

Example 3.4.1 : Using pumping lemma, prove that the language,

$A = \{a^n b^n \mid n \geq 0\}$ is not regular.

Solution :

Proof :

- Assume that A is regular.
- Pumping length = p.

$$s = a^p b^p$$

- Divide it into 3 parts x, y, z.
- Assume that p = 7.
- So $s \Rightarrow$ aaaaaaabbbbbbb

Condition I :

Case 1 :

The y is in 'a' part.

a aa a a aa b b b b b b b

x y z

Case 2 :

The y is in 'b' part.

a a a a a a a b bb b b bb

x y z

case y^2 $|xy| \leq p$

Case 3 :

The Y is in the 'a' and 'b' Part.

aaaaa aabb bbbbb

X Y Z

For case 1 :

$x y^i z \Rightarrow x y^2 z$

in case 1 → x = aa

y = aaaa

z = abbbbbbb

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaaaaa bbbbbbb

In this string number of a = 11 and b = 7.

$11 \neq 7$

So this string does not lie in our language.

For case 2 :

$x y^i z \Rightarrow x y^2 z$

in case 2 → x = aaaaaaabb

y = bbbb

z = b

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaabbbbbbbbbbb

$7 \neq 11$

So this string does not lie in our language .

For case 3 :

$x y^i z \Rightarrow x y^2 z$

in case 3 → x = aaaaa

y = aabb

z = bbbbb

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaaabbaabbbbbbb

This string does not follow the $a^n b^n$ pattern, so these strings do not lie in our language.

Condition 3

$|xy| \leq p$

where, $p = 7$

Case 1 :

$|x| = 2, \quad |y| = 4$

$\therefore$ $|xy| = 2 + 4 = 6$ ...satisfied

$6 \leq 7$

Case 2 :

$|x| = 9, \quad |y| = 4$

$\therefore$ $|xy| = 9 + 4 = 13$ ...not satisfied

$13 \nleq 7$

Case 3 :

$|x| = 5, \quad |y| = 4$

$\therefore$ $|xy| = 5 + 4 = 9$ ...not satisfied

$9 \nleq 7$

So we can say our language A is not regular.

Example 3.4.2 : Using pumping Lemma prove that the language, $A = \{ y\,y \mid y \in (0, 1)^* \}$ is not regular.

Solution :

Consider,

$\underbrace{01}_{y}\ \underbrace{01}_{y}$

Proof

- Assume that A is a regular.
- Then it must have a pumping length = P
- Choose string S;

$S = \underline{0^P 1 0^P 1}$

- Divide S into three parts X, Y, Z
- Consider P = 7
- String, S = 0000000100000001
- Divide this string into X, Y, Z

$\underbrace{00}_{X}\ \underbrace{0000}_{Y}\ \underbrace{0100000001}_{Z}$

Condition I

- Showing string in form of $X Y^i Z$

Let, take $i = 2$

$\therefore$ $X Y^i Z \Rightarrow X Y^2 Z$

- In our case,

$X = 00, \ Y = 0000, \ Z = 0100000001$

$X Y^2 Z = 000000000001000000001$

This string does not lie in language A.

Condition II

$|Y| > 0$

$4 > 0$ ...satisfied

Condition III

$|XY| \leq P$

Where $P = 7, \ X = 02, Y = 04$

$|XY| = 02 + 04 = 06$

$|XY| \leq 7$

$06 \leq 7$ ...satisfied

But condition is not satisfied so our language is not a regular.

Syllabus Topic : Closure Properties

3.5 Closure Properties of Regular Sets

3.5.1 Regular set

Any set that represents the value of regular expression is called **regular set.**

Properties of regular sets :

1.	Union	2.	Intersection
3.	Complement	4.	Difference
5.	Reversal	6.	Closure of regular
7.	Concatenation		

Property 1 : Union of two regular set is regular.

Proof :

Lets consider two regular expressions.

$R_1 = a(aa)^*$

$R_2 = (aa)^*$

For R_1 :

$L(R_1) = \{a, aaa, aaaaa, ...\}$

$= \{$ Strings of odd length excluding Null $\}$

For R_2 :

$L(R_2) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

$= \{$ Strings of even length including null$\}$

$\therefore L(R_1) \cup L(R_2) = \{\varepsilon, a, aa, aaa, aaaa, ...\}$

$\therefore RE = (L(R_1) \cup L(R_2)) = a^*$ which is regular expression itself.

Property 2 : Intersection of two regular sets is regular.

Proof : $R_1 = a(a^*)$, $R_2 = (aa)^*$

So,

$L(R_1) = \{a, aa, aaa, ...\}$

$L(R_2) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

$\therefore L(R_1) \cap L(R_2) = \{aa, aaaa, aaaaaa, ...\}$

$\therefore RE(L(R_1) \cap L(R_2)) = aa(aa)^*$ which is a regular expression itself.

Property 3 : Complement of regular set is regular

Proof :

$R_1 = (aa)^*$

so $L(R_1) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

Complement of $L(R_1)$ is all strings that is not in L

$L'(R_1) = \{a, aaa, aaaaa, ...\}$

$\therefore RE\ L' = a(aa)^*$ which is regular expression itself

Property 4 : The difference of two regular set is regular.

Proof :

$R_1 = a(a)^*$, $R_2 = (aa)^*$

$\therefore L(R_1) = \{a, aa, aaa, aaaa, ...\}$

$L(R_2) = \{\varepsilon, aa, aaaa, ...\}$

$L(R_1) - L(R_2) = \{a, aaa, aaaaa, ...\}$

$\therefore RE(L(R_1) - L(R_2)) = a(aa)^*$ which is regular expression itself

Property 5 : The reversal of regular set is regular

Proof :

$\therefore L(R_1) = \{01, 10, 11, 10\}$

$RE(L(R_1)) = 01 + 10 + 11 + 10$

$\therefore L^R = \{10, 01, 11, 01\}$

$RE(L^R) = 01 + 10 + 11 + 10$ which is regular expression

Property 6 : Closure of regular set is regular

Proof :

$R_1 = a(aa)^*$

$\therefore L(R_1) = \{a, aaa, aaaaa, ...\}$

$\therefore L^* = \{a, aa, aaa, aaaa, ...\}$

$\therefore RE(L^*) = a(a^*)$ which is regular expression

Property 7 : Concatenation of two regular sets is regular

Proof :

$R_1 = (0+1)^* 0$, $R_2 = 01(0+1)^*$

$\therefore$ $L(R_1) = \{0, 00, 10, 000, 010, \ldots\}$ all string ending in 0

$\therefore$ $L(R_2) = \{01, 010, 011, \ldots\}$ all string beginning with 01

then after concatenation of these two strings we get,

{ 001, 0010, 0011, 0001, 00010, 00011, ... }

Set of string containing 001 as a substring

Which can be represented by,

$$RE = (0+1)^* 001 (0+1)^*$$

Syllabus Topic : Regular Sets and Regular Grammars

3.6 Regular Sets and Regular Grammars

- In this section, we show that class of regular sets over Σ is precisely the regular languages over the terminal set Σ.
- Construction of regular grammar generating T(M) for a given DFA M.

Let, $M = (\{q_0, \ldots\ldots q_n\}, \Sigma, \delta, q_0, F)$

If w is in T(M), then it is obtained by concatenating labels corresponding to several transitions, the first from q_0 and the last terminating at some final state. So for grammar G to be constructed, productions should correspond to transition.

Now construct the Grammar ;

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

Where production P :

(1) $A_i \rightarrow aA_j$ is included in P,

if $\delta(q_i, a) = A_j \in F$.

(2) $A_i \rightarrow a A_j$ and $Ai \rightarrow a$ are included in P

if $\delta(q_i, a) = q_j \in F$.

We can show that L(G) = T(M) by string construction of P, such a construction gives,

$A_i \Rightarrow aA_j$ if $\delta(q_i, a) = q_j$

$A_i \Rightarrow a$ if $\delta(q_i, a) \in F$

So,

$A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots\ldots \Rightarrow a_1, \ldots\ldots, a_{k-1} A_k \Rightarrow a_1 a_2 \ldots\ldots a_k$

If $\delta(q_0, a_1) = q_1$

$\delta(q_1, a_2) = q_2$

:

:

$\delta(q_k, a_k) \in f$

This proves that $w = a_1 \ldots\ldots a_k \in L(G)$

If $\delta(q_0, a_1 \ldots\ldots a_k) \in F$ i.e. if $w \in T(M)$

For example,

1. **Construct regular grammar G generating the regular set represent by**

$$P = a^* b (a+b)^*$$

Step I : Construct finite automata of given regular expression

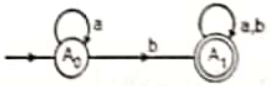


Fig. 3.6.1

Step II : Convert this finite automata into regular grammar

Grammar consists of non-terminal, terminal, productions and initial symbol.

$G = (N, T, P, S) = (\{A_0, A_1\}, \{a, b\}, P, A_0)$

Non-terminal = $\{A_0, A_1\}$

Terminals = $\{a\ b\}$

Initial symbol = A_0

Production P : $A_0 \rightarrow a A_0$, $A_0 \rightarrow bA_1$, $A_0 \rightarrow b$

$A_1 \rightarrow b A_1$, $A_1 \rightarrow a A_1$, $A_1 \rightarrow a$, $A_1 \rightarrow b$

2. Construct regular grammar G at given finite

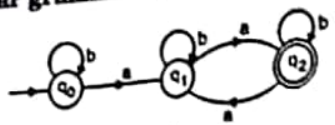


Fig. 3.6.2

$G = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$

Production P :

$q_0 \rightarrow b\, q_0$, $q_1 \rightarrow a\, q_2$, $q_2 \rightarrow \in / b$

$q_0 \rightarrow a\, q_1$, $q_2 \rightarrow b\, q_2$, $q_1 \rightarrow a$

$q_1 \rightarrow b\, q_1$, $q_2 \rightarrow a\, q_1$

3.6.1 Construction of Transition System

Construction of transition system M accepting L(G) for given regular grammar G.

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

   We construct transition system M whose,

1. States correspond to variable
2. Initial state corresponds to A_0.
3. Transition corresponds to production P.
4. Last production applied in any derivation is of the form $A_j \rightarrow a$.
5. Transition terminals at final state.

   M is defined as, $(\{q_0, q_1, \ldots\ldots, q_f\}, \Sigma, \delta, q_0, \{q_f\})$

   Where δ is transition function is defined as,

1. Each production $A_i \rightarrow aA_j$ induces a transition from q_i to q_f with label a
2. Each production $A_k \rightarrow a$ induces a transition from q_k to q_f with label a

Suppose,

$A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots\ldots$

$\Rightarrow a_1 \ldots\ldots, a_{n-1} A_{n-1} \Rightarrow a_1 \ldots\ldots a_n$ is a derivation of $a_1 a_2, \ldots\ldots$ an. If there is a path in M starting from q_0 and terminating q_f with path value $a_1 a_2 \ldots\ldots a_n$

$\therefore$ L (G) = T(m)

Example 3.6.1 : $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$ when

P : $A_0 \rightarrow a A_1$

$A_1 \rightarrow b A_1$

$A_1 \rightarrow a$

$A_1 \rightarrow b A_0$

Construct a transition system M accepting L(G)

Solution :

Lets consider,

$M = (\{q_0 q_1 q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$

when q_0 and q_1 correspond to A_0 and A_1 respectively and q_f is a final state introduced,

$A_0 \rightarrow a A_1$ .... induces a transition from q_0 to q_1 with label a

$A_1 \rightarrow b A_1$ .... induces a transition from q_1 to q_1 with label b

$A_1 \rightarrow b A_0$ .... induces a transition from q_1 to q_0 with label b

$A_1 \rightarrow a$ .... induces a transition from q_1 to q_f with label a

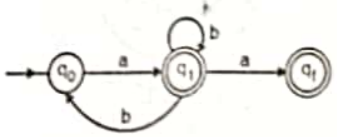


Fig. P. 3.6.1

Example 3.6.2 : If regular grammar G is given by,

$S \rightarrow as \mid a$ find M accepting L(G)

Solution :

Let q_0 correspond to S and q_f is final state.

$$M = (\{q_0, q_f\}, \{a\}, \delta, q_0, \{q_f\})$$

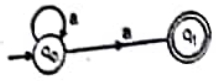


Fig. P. 3.6.2

Exercise

Q. 1. Construct a regular expression corresponding to the state diagram described by Fig. 1

Fig. 1

Q. 2. Find regular expression :

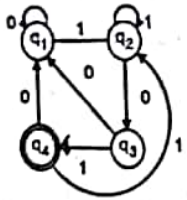


Fig. 2

Q. 3 Construct DFA with reduced state equivalent to regular expression,

$10 + (0 + 11)\,0^*1$.

□□□

CHAPTER 4

Context Free Languages

University Prescribed Syllabus

Context Free Languages : Context-free Languages, Derivation Tree, Ambiguity of Grammar, CFG simplification, Normal Forms, Pumping Lemma for CFG.

Syllabus Topic : Context Free Languages

4.1 Context Free Languages

Write a note on context free languages.

- A grammar is a set of rules for putting strings together and so corresponds to a language.
- A type 2 grammar is called as **context-free grammar** (as A can be replaced by α in any context).
- A language generated by context free grammar is called **a type 2 language** or **a context free language.**

Definition of Context Free Grammar (CFG)

Context Free Grammar consisting a finite set of grammar rules is a quadruple (N, Σ, P, S) where,

N = A set of non-terminal symbols.

Σ = A set of terminals; where, $N \cap T$ = NULL (Set N intersection set T is always equal to NULL)

P = Set of production rules.